Семинарни по ДАА
(чернова)

Траян Господинов

7 юни 2023 г.

Съдържание

Глава 1

Асимптотични нотации

1.1 Теория

Започваме с припомняне на две дефинции от математическия анализ.

Дефиниция 1.1: Асимптотично неотрицателна функция

Ще казваме, че $f : \mathbb{R}_0^+ \to \mathbb{R}$ е асимптотично неотрицателна, т.с.т.к. $\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(f(n) \geq 0\big)$, където $\mathbb{R}_0^+$ са неотрицателнителните реални числа.

Дефиниция 1.2: Асимптотично положителна функция

Ще казваме, че $f : \mathbb{R}_0^+ \to \mathbb{R}$ е асимптотично положителна, т.с.т.к. $\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(f(n) > 0\big)$.

С $\mathscr{F}^+$ и $\mathscr{F}_0$ ще означаваме съответно множествата от всички асимптотично положителни и множествата от всички асимптотично неотрицателни функции. Функции в тези класове ще имат семантика на *брой стъпки до приключване изпълнението на дадена програма*, което очевидно няма как да ни върне отрицателен резултат. Също така ще се интересуваме да сравняваме (асимптотично) такива функции. За целта въвеждаме пет (несобствени) класа от фунции спрямо асимптотиката на дадена функция.

Дефиниция 1.3: Основни класове от функции, спрямо асимптотиката на фукнция

Нека $g \in \mathscr{F}^+$. Дефинираме следните пет класа:

- $O(g) \leftrightharpoons \{f \in \mathscr{F}^+ | \big(\exists c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq f(n) \leq cg(n)\big)\}$
- $o(g) \leftrightharpoons \{f \in \mathscr{F}^+ | \big(\forall c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq f(n) \leq cg(n)\big)\}$
- $\Omega(g) \leftrightharpoons \{f \in \mathscr{F}^+ | \big(\exists c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq cg(n) \leq f(n)\big)\}$
- $\omega(g) \leftrightharpoons \{f \in \mathscr{F}^+ | \big(\forall c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq cg(n) \leq f(n)\big)\}$
- $\Theta(g) \leftrightharpoons \{f \in \mathscr{F}^+ | \big(\exists c_1 > 0\big)\big(\exists c_2 > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\big)\}$

Забележка. Поради исторически причини е прието да се пише $f = O(g)$ вместо $f \in O(g)$. Аналогично и за другите четири класа.

Пример 1.1. $O(n^2) = \{n^2, 100n^2, \frac{n^2}{5}, n, 10^6 n, 1, 5n^2 - 1000n - 1000, \dots\}$.
Нека разгледаме примерни свидетели c и n_0 за $f = O(n^2)$ съответно за $g(n) = n^2$ и $f(n)$:

- $f(n) = n^2$

  Директно се вижда, че $c = 1$ и $n_0 = 0$ ни вършат работа:
  $$\big(\forall n \geq \underbrace{0}_{n_0}\big)\big(0 \leq \underbrace{n^2}_{f(n)} \leq \underbrace{1}_{c} * \underbrace{n^2}_{g(n)}\big)$$

- $f(n) = 100n^2$

  Отново лесно се преценя, че $c = 100$ и $n_0 = 0$ ни вършат работа:
  $$\big(\forall n \geq \underbrace{0}_{n_0}\big)\big(0 \leq \underbrace{100n^2}_{f(n)} \leq \underbrace{100}_{c} * \underbrace{n^2}_{g(n)}\big)$$

- $f(n) = \frac{n^2}{5}$

  Отново $c = 1$ и $n_0 = 0$ ни вършат работа (разбира се и $c = \frac{1}{5}$ $n_0 = 0$ също):
  $$\big(\forall n \geq \underbrace{0}_{n_0}\big)\big(0 \leq \underbrace{\frac{n^2}{5}}_{f(n)} \leq \underbrace{1}_{c} * \underbrace{n^2}_{g(n)}\big)$$

- $f(n) = 5n^2 - 1000n - 1000$

  Макар и сташно на първи поглед, отново лесно се преценя, че $c = 10^7$ и $n_0 = 0$ ни вършат работа (разбира се това не е минимален избор на c и n_0.. интересува ни само да намерим кои да е):
  $$\big(\forall n \geq \underbrace{0}_{n_0}\big)\big(0 \leq \underbrace{5n^2 - 1000n - 1000}_{f(n)} \leq \underbrace{10^7}_{c} * \underbrace{n^2}_{g(n)}\big)$$

Упражнете се с другите четири класа.

Нотация 1.1: Основни релации над асимптотично положителни функции

Въвеждаме следните пет релации за удобство:

- $f \preccurlyeq g \overset{\text{def}}{\leftrightarrow} f = O(g)$
- $f \prec g \overset{\text{def}}{\leftrightarrow} f = o(g)$
- $f \succcurlyeq g \overset{\text{def}}{\leftrightarrow} f = \Omega(g)$
- $f \succ g \overset{\text{def}}{\leftrightarrow} f = \omega(g)$
- $f \asymp g \overset{\text{def}}{\leftrightarrow} f = \Theta(g)$ (асимптотично равенство с точност до константа)

Нотация 1.2: Асимптотично равенство

$f \sim g \overset{\text{def}}{\leftrightarrow} \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 1$

Основни свойства: Асимптотични релации

Нека $f, g \in \mathscr{F}^+$. Тогава:

1. $f \,\sigma\, g \wedge g \,\sigma\, h \to f \,\sigma\, h$, $\sigma \in \{\preccurlyeq, \prec, \succcurlyeq, \succ, \asymp\}$
2. $f \,\sigma\, f$, $\sigma \in \{\preccurlyeq, \succcurlyeq, \asymp\}$
3. $f \preccurlyeq g \wedge g \preccurlyeq f \to f \asymp g$
4. $f \asymp g \leftrightarrow g \asymp f$
5. $f \preccurlyeq g \leftrightarrow g \succcurlyeq f$ (аналогично $f \prec g \leftrightarrow g \succ f$)
6. $f + g \asymp max(f, g)$
   Док: $\frac{f(n)+g(n)}{2} \leq max(f(n), g(n)) \leq f(n) + g(n)$
7. $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0 \leftrightarrow f \prec g \,(f = o(g))$
   Док: От деф. на лимит имаме: $\big(\forall \varepsilon > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(-\varepsilon < \frac{f(n)}{g(n)} < \varepsilon\big)$. Сега умножаваме двете страни по $g(n)$ и получаваме $-g(n) * \varepsilon < 0 < f(n) < g(n) * \varepsilon$, откъдето имаме $\big(\forall \varepsilon > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq f(n) \leq \varepsilon g(n)\big)$, което е точно дефиницията на $o(g)$. Обратната посока е аналогична.
8. $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = c > 0 \to f \asymp g \,(f = \Theta(g))$
   Док: От деф. на лимит имаме: $\big(\forall \varepsilon > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(c - \varepsilon < \frac{f(n)}{g(n)} < c + \varepsilon\big)$. Сега умножаваме двете страни по $g(n)$ и получаваме $g(n) * (c - \varepsilon) < f(n) < g(n) * (c + \varepsilon)$. Тъй като $c > 0$, то е ясно че има $\varepsilon > 0 : c - \varepsilon > 0$. Тоест получихме $\big(\exists \varepsilon > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 < \underbrace{(c-\varepsilon)}_{c_1} g(n) < f(n) < \underbrace{(c+\varepsilon)}_{c_2} g(n)\big)$, което е точно дефиницията на $\Theta(g)$.
   Обратната посока **не** е вярна.
9. Нека g не е ограничена отгоре и нека $a > 1$. Тогава:
   (а) $f \prec g \to a^{f(n)} \prec a^{g(n)}$ (нестрогия аналог **не** е верен)
   (б) $log_a f(n) \prec log_a g(n) \to f(n) \prec g(n)$ (нестрогия аналог **не** е верен)
10. $\big(\forall a > 1\big)\big(\forall t > 0\big)\big(\forall \varepsilon > 0\big)\big(log_a^t(n) \prec n^\varepsilon\big)$

Факт 1.1: Апроксимация на Стирлинг

$n! = \sqrt{2\pi n}\big(\frac{n}{e}\big)^n(1 + \frac{1}{12n} + \frac{1}{288n^2} + \dots)$

Следствие 1.1: Асимптотична апроксимация на Стирлинг

$n! \sim \sqrt{2\pi n}\big(\frac{n}{e}\big)^n$

Приложение.

- $log(n!) \asymp nlog(n)$ (докажете за упражнение)
- $\binom{2n}{n} = \frac{(2n)!}{n!n!} \sim \frac{\sqrt{4\pi n}\big(\frac{2n}{e}\big)^{2n}}{2\pi n\big(\frac{n}{e}\big)^{2n}} = \frac{2^{2n}n^{2n}}{\sqrt{\pi n}\,n^{2n}} = \frac{2^{2n}}{\sqrt{\pi n}} = \frac{4^n}{\sqrt{\pi n}}$

Факт 1.2: Логаритми и техните свойства

0. $\big(\forall a \in \mathbb{R}^+\backslash\{1\}\big)\big(\forall b \in \mathbb{R}^+\big)\big(a^x = b \leftrightarrow x = log_a(b)\big)$
1. $a^{log_a(b)} = b$
2. $log_{a^n}(b^m) = \frac{m}{n} log_a(b)$
3. (а) $log_a(x) + log_a(y) = log_a(xy)$
   (б) $log_a(x) - log_a(y) = log_a(\frac{x}{y})$
4. $log_a(x) = \frac{log_b(x)}{log_b(a)}$, $b \in \mathbb{R}^+\backslash\{1\}$
   (а) $log_a(b) = \frac{1}{log_b(a)}$
   (б) $log_b(a) * log_a(x) = log_b(x)$
5. $a^{log_b(x)} = x^{log_b(a)}$

1.2 Задачи

Задача 1.1. Нека $p(x) = a_0x^k + \cdots + a_k$ е асимптотично положителен полином (т.е. $a_0 > 0$). Да се докаже, че $p(n) \asymp n^k$.

Решение. $\lim\limits_{n\to\infty} \frac{p(n)}{n^k} = \lim\limits_{n\to\infty} \frac{a_0x^k+\cdots+a_k}{n^k} = \underbrace{\lim\limits_{n\to\infty} \frac{a_0n^k}{n^k}}_{a_0} + \underbrace{\lim\limits_{n\to\infty} \frac{a_1n^{k-1}}{n^k}}_{0} + \cdots + \underbrace{\lim\limits_{n\to\infty} \frac{a_n}{n^k}}_{0} = a_0 > 0.$

От основно свойство 8 следва, че $p(n) \asymp n^k$.

Задача 1.2. Нека $k \in \mathbb{N}^+$. Да се докаже, че $\binom{n}{k} \asymp n^k$.

Решение. $\lim\limits_{n\to\infty} \frac{\binom{n}{k}}{n^k} = \lim\limits_{n\to\infty} \frac{n(n-1)\ldots(n-k+1)}{n^k k!} = \frac{1}{k!} > 0$. От основно свойство 8 следва, че $\binom{n}{k} \asymp n^k$.

Задача 1.3. Да се докаже, че $(n+1)^n \asymp n^n$.

Решение. $\lim\limits_{n\to\infty} \frac{(n+1)^n}{n^n} = \lim\limits_{n\to\infty} \big(\frac{n+1}{n}\big)^n = e > 0$. От основно свойство 8 следва, че $(n+1)^n \asymp n^n$.

Задача 1.4. Нека $f(n) = \begin{cases} 1 & , \lfloor n \rfloor \equiv 0(2) \\ n^2 & , \lfloor n \rfloor \equiv 1(2) \end{cases}$ и $g(n) = n$. Да се докаже, че те са асимптотично несравними.

Решение. За упражнение...

Задача 1.5. Вярна ли е следната импликация: $f = O(g) \to (f = o(g) \vee f = \Theta(g))$?

Решение. Не е вярна! Проверете за упражнение със следния контрапример:
$f(n) = \begin{cases} n & , \lfloor n \rfloor \equiv 0(2) \\ 1/n & , \lfloor n \rfloor \equiv 1(2) \end{cases}$ и $g(n) = n$.

Задача 1.6. Да се сортират по асимптотика следните функции:

$f_1(n) = n^3$	$f_2(n) = \sqrt{n}$	$f_3(n) = log(n)$	$f_4(n) = log^2(n)$
$f_5(n) = log^{(2)}(n)$	$f_6(n) = n!$	$f_7(n) = a^n$	$f_8(n) = a$
$f_9(n) = n^n$	$f_{10}(n) = n^{-2}$	$f_{11}(n) = n^2$	$f_{12}(n) = n^{log(n)}$

Решение.
$n^n \succ n! \succ a^n \succ n^{log(n)} \succ n^3 \succ n^2 \succ \sqrt{n} \succ log^2(n) \succ log(n) \succ log^{(2)}(n) \succ a \succ n^{-2}$
Ще опишем подробно решенията:

1. $\boldsymbol{n^n \succ n!}$

   От анализа знаем, че $\lim\limits_{n\to\infty} \frac{n!}{n^n} = 0$. Тогава от основно свойство 7 следва, че $n! \prec n^n$.

2. $\boldsymbol{n! \succ a^n}$

   Ще приложим основно свойство 9 ... т.е. логаритмуваме двете страни и получаваме

   $$log(n!) \,?\, log(a^n)$$

   $$nlog(n) \,?\, nlog(a)$$

   Знаем, че $\lim\limits_{n\to\infty} \frac{nlog(a)}{nlog(n)} = 0$. Тогава от основно свойство 7 следва, че $nlog(a) \prec nlog(n)$. Оттук и от основно свойство 9 следва, че $a^n \prec n!$.

3. $\boldsymbol{a^n \succ n^{log(n)}}$

   Ще приложим основно свойство 9 ... т.е. логаритмуваме двете страни и получаваме

   $$log(a^n) \,?\, log(n^{log(n)})$$

   $$nlog(a) \,?\, log(n)log(n)$$

   От основно свойство 10 имаме $log(n)log(n) \prec nlog(a)$. Оттук и от основно свойство 9 следва, че $n^{log(n)} \prec a^n$.

4. $\boldsymbol{n^{log(n)} \succ n^3}$

   Ще приложим основно свойство 9 . . . т.е. логаритмуваме двете страни и получаваме

   $$log(n^{log(n)}) \,?\, log(n^3)$$

   $$log^2(n) \,?\, 3log(n)$$

   Знаем, че $\lim\limits_{n\to\infty} \frac{3log(n)}{log^2(n)} = 0$. Тогава от основно свойство 7 следва, че $3log(n) \prec log^2(n)$. Оттук и от основно свойство 9 следва, че $n^3 \prec n^{log(n)}$.

5. $\boldsymbol{n^3 \succ n^2}$

   Знаем, че $\lim\limits_{n\to\infty} \frac{n^2}{n^3} = 0$. Тогава от основно свойство 7 следва, че $n^2 \prec n^3$.

6. $\boldsymbol{n^2 \succ \sqrt{n}}$

   Знаем, че $\lim\limits_{n\to\infty} \frac{\sqrt{n}}{n^2} = 0$. Тогава от основно свойство 7 следва, че $\sqrt{n} \prec n^2$.

7. $\boldsymbol{\sqrt{n} \succ log^2(n)}$

   От основно свойство 9 директно следва, че $log^2(n) \prec \sqrt{n}$.

8. $\boldsymbol{log^2(n) \succ log(n)}$

   Знаем, че $\lim\limits_{n\to\infty} \frac{log(n)}{log^2(n)} = 0$. Тогава от основно свойство 7 следва, че $log(n) \prec log^2(n)$.

9. $\boldsymbol{log(n) \succ log(log(n))}$

   Полагаме $m = log(n)$ и получаваме

   $$m \,?\, log(m)$$

   От основно свойство 10 следва, че $log(m) \prec m$. Сега като върнем полагането получаваме $log(log(n)) \prec log(n)$.

10. $\boldsymbol{log(log(n)) \succ a}$

    Знаем, че $\lim\limits_{n\to\infty} \frac{a}{log^{(2)}(n)} = 0$. Тогава от основно свойство 7 следва, че $a \prec log^{(2)}(n)$.

11. $\boldsymbol{a \succ n^{-2}}$

    Знаем, че $\lim\limits_{n\to\infty} \frac{n^{-2}}{a} = 0$. Тогава от основно свойство 7 следва, че $n^{-2} \prec a$.

Задача 1.7. Да се сортират по асимптотика следните функции:

$f_1(n) = (\sqrt{2})^{log(n)}$	$f_2(n) = n^3$	$f_3(n) = n!$	$f_4(n) = (log(n))!$
$f_5(n) = e^{-2ln(n)}$	$f_6(n) = log^2(n)$	$f_7(n) = log(n!)$	$f_8(n) = 2^{2^n}$
$f_9(n) = n^{\frac{1}{log(n)}}$	$f_{10}(n) = log^{(2)}(n)$	$f_{11}(n) = \left(\frac{3}{2}\right)^n$	$f_{12}(n) = n2^n$
$f_{13}(n) = 4^{log(n)}$	$f_{14}(n) = (n+1)!$	$f_{15}(n) = \sqrt{log(n)}$	$f_{16}(n) = 2^{\sqrt{2log(n)}}$
$f_{17}(n) = n^{log(log(n))}$	$f_{18}(n) = log(n)$	$f_{19}(n) = 2^{log(n)}$	$f_{20}(n) = (log(n))^{log(n)}$

Решение.
$e^{-2ln(n)} \prec n^{\frac{1}{log(n)}} \prec log^{(2)}(n) \prec \sqrt{log(n)} \prec log(n) \prec log^2(n) \prec 2^{\sqrt{2log(n)}} \prec (\sqrt{2})^{log(n)} \prec 2^{log(n)} \prec log(n!) \prec 4^{log(n)} \prec n^3 \prec (log(n))! \prec (log(n))^{log(n)} \asymp n^{log(log(n))} \prec \left(\frac{3}{2}\right)^n \prec n2^n \prec n! \prec (n+1)! \prec 2^{2^n}$
Ще опишем подробно решенията:

1. $\boldsymbol{e^{-2ln(n)} \prec n^{\frac{1}{log(n)}}}$

   Ще преобразуваме функциите използвайки основните свойства на логаритмите:

   $$(e^{ln(n)})^{-2} \,?\, n^{log_n(2)}$$

   $$(n^{ln(e)})^{-2} \,?\, 2^{log_n(n)}$$

   $$n^{-2} \,?\, 2$$

   Сега знаем, че $\lim\limits_{n\to\infty} \frac{n^{-2}}{2} = 0$. Тогава от основно свойство 7 следва, че $n^{-2} \prec 2$.

2. $\boldsymbol{n^{\frac{1}{log(n)}} \prec log^{(2)}(n)}$

   Вече показахме, че $n^{\frac{1}{log(n)}} = 2$. Сега знаем, че $\lim\limits_{n\to\infty} \frac{2}{log^{(2)}(n)} = 0$. Тогава от основно свойство 7 следва, че $2 \prec log^{(2)}(n)$.

3. $\boldsymbol{log^{(2)}(n) \prec \sqrt{log(n)}}$

   Полагаме $m = log(n)$ и получаваме

   $$log(m) \,?\, \sqrt{m}$$

   От основно свойство 10 следва, че $log(m) \prec m$. Сега като върнем полагането получаваме $log^{(2)}(n) \prec \sqrt{log(n)}$.

4. $\boldsymbol{\sqrt{log(n)} \prec log(n)}$

   Директно прилагаме основно свойство 7.

5. $\boldsymbol{log(n) \prec log^2(n)}$

   Директно прилагаме основно свойство 7.

6. $\boldsymbol{(log(n))^2 \prec 2^{\sqrt{2log(n)}}}$

   Ще приложим основно свойство 9 . . . т.е. логаритмуваме двете страни и получаваме

   $$log((log(n))^2) \,?\, log(2^{\sqrt{2log(n)}})$$

   $$2log(log(n)) \,?\, \sqrt{2log(n)}log(2)$$

   Полагаме $m = log(n)$ и получаваме

   $$2log(m) \,?\, \sqrt{2m}$$

   От основно свойство 10 имаме $2log(m) \prec \sqrt{2m}$. Сега като върнем полагането имаме $2log(log(n)) \prec \sqrt{2log(n)}$. Оттук и от основно свойство 9 следва, че $(log(n))^2 \prec 2^{\sqrt{2log(n)}}$.

7. $\boldsymbol{2^{\sqrt{2log(n)}} \prec (\sqrt{2})^{log(n)}}$

Ще приложим основно свойство 9 . . . т.е. логаритмуваме двете страни и получаваме

$$log(2^{\sqrt{2log(n)}}) \,?\, log(\sqrt{n})$$

$$\sqrt{2log(n)} \,?\, \frac{1}{2}log(n)$$

Знаем, че $\lim\limits_{n\to\infty} \frac{\sqrt{2log(n)}}{\frac{1}{2}log(n)} = 0$. Тогава от основно свойство 7 следва, че $\sqrt{2log(n)} \prec \frac{1}{2}log(n)$. Оттук и от основно свойство 9 следва, че $2^{\sqrt{2log(n)}} \prec (\sqrt{2})^{log(n)}$.

8. $\boldsymbol{(\sqrt{2})^{log(n)} \prec 2^{log(n)}}$

Ще преобразуваме функциите използвайки основните свойства на логаритмите:

$$n^{log(\sqrt{2})} \,?\, n^{log(2)}$$

$$\sqrt{n} \,?\, n$$

Сега знаем, че $\lim\limits_{n\to\infty} \frac{\sqrt{n}}{n} = 0$. Тогава от основно свойство 7 следва, че $\sqrt{n} \prec n$.

9. $\boldsymbol{2^{log(n)} \prec log(n!)}$

Вече показахме, че $2^{log(n)} = n$ и $log(n!) \asymp nlog(n)$. Сега знаем, че $\lim\limits_{n\to\infty} \frac{n}{nlog(n)} = 0$. Тогава от основно свойство 7 следва, че $n \prec nlog(n)$.

10. $\boldsymbol{log(n!) \prec 4^{log(n)}}$

Ще преобразуваме $4^{log(n)}$ използвайки основните свойства на логаритмите:

$$log(n!) \,?\, n^{log(4)}$$

$$nlog(n) \,?\, n^2$$

Тогава от основно свойство 10 следва, че $nlog(n) \prec n^2$.

11. $\boldsymbol{4^{log(n)} \prec n^3}$

Вече показахме, че $4^{log(n)} = n^2$. Сега знаем, че $\lim\limits_{n\to\infty} \frac{n^2}{n^3} = 0$. Тогава от основно свойство 7 следва, че $n^2 \prec n^3$.

12. $\boldsymbol{n^3 \prec (log(n))!}$

Ще преобразуваме $(log(n))!$ използвайки апроксимацията на Стирлинг:

$$(log(n))! \sim \sqrt{2\pi log(n)}\left(\frac{log(n)}{e}\right)^{log(n)}$$

Ще приложим основно свойство 9 . . . т.е. логаритмуваме двете страни и получаваме

$$log(n^3) \,?\, log\left(\sqrt{2\pi log(n)}\left(\frac{log(n)}{e}\right)^{log(n)}\right)$$

$$3log(n) \,?\, \underbrace{log\left(\sqrt{2\pi log(n)}\right)}_{\asymp log(log(n))} + log(n)\,log(log(n)) - log(n)log(e)$$

$$log(n) \,?\, log(n)\,log(log(n))$$

Знаем, че $\lim\limits_{n\to\infty} \frac{log(n)}{log(n)\,log(log(n))} = 0$. Тогава от основно свойство 7 следва, че $log(n) \prec log(n)\,log(log(n))$. Оттук и от основно свойство 9 следва, че $n^3 \prec (log(n))!$.

13. $\boldsymbol{(log(n))! \prec (log(n))^{log(n)}}$

Полагаме $m = log(n)$ и получаваме

$$m! \,?\, m^m$$

От анализа знаем, че $\lim\limits_{n\to\infty} \frac{m!}{m^m} = 0$. Тогава от основно свойство 7 следва, че $m! \prec m^m$. Сега като върнем полагането получаваме $(log(n))! \prec (log(n))^{log(n)}$.

14. $\boldsymbol{(log(n))^{log(n)} \asymp n^{log(log(n))}}$

Директно прилагаме основно свойство 5 на логаритмите.

15. $\boldsymbol{n^{log(log(n))} \prec \left(\frac{3}{2}\right)^n}$

Ще приложим основно свойство 9 ... т.е. логаритмуваме двете страни и получаваме

$$log(n)\,log(log(n)) \,?\, n$$

От основно свойство 10 следва, че $log(n)\,log(log(n)) \prec log^2(n) \prec n$. Оттук и от основно свойство 9 следва, че $n^{log(log(n))} \prec \left(\frac{3}{2}\right)^n$.

16. $\boldsymbol{\left(\frac{3}{2}\right)^n \prec n2^n}$

Директно прилагаме основно свойство 7. $//\left(\frac{3}{2}\right)^n \prec 2^n \prec n2^n$

17. $\boldsymbol{n2^n \prec n!}$

Ще приложим основно свойство 9 ... т.е. логаритмуваме двете страни и получаваме

$$log(n2^n) \,?\, log(n!)$$

$$log(n) + nlog(2) \,?\, nlog(n)$$

Знаем, че $\lim\limits_{n\to\infty} \frac{n}{nlog(n)} = 0$. Тогава от основно свойство 7 следва, че $n \prec log(n)$. Оттук и от основно свойство 9 следва, че $n2^n \prec n!$.

18. $\boldsymbol{n! \prec (n+1)!}$

Директно прилагаме основно свойство 7

19. $\boldsymbol{(n+1)! \prec 2^{2^n}}$

Ще приложим основно свойство 9 ... т.е. логаритмуваме двете страни и получаваме

$$(n+1)log(n+1) \,?\, 2^n$$

Логаритмуваме още веднъж

$$log(n+1)\,log(log(n+1)) \,?\, n$$

От основно свойство 10 следва, че $log(n)\,log(log(n)) \prec log^2(n) \prec n$. Оттук и от основно свойство 9 следва, че $log(n)\,log(log(n)) \prec 2^n$. И отново от основно свойство 9 следва, че $(n+1)! \prec 2^{2^n}$.

Глава 2

Анализ на коректността на алгоритми

2.1 Итеративни алгоритми

Коректност на итеративни алгоритми ще доказваме чрез ***инвариант***. Това е пособие, което много наподобява доказателство чрез индукция. Аналогично на индукцията имаме ***база***. Индуктивната хипотеза и индуктивната стъпка тук са обединени в една фаза - ***поддръжка***. Имаме нова фаза наречена ***терминация***.

Забележка 2.1

Важно е да отбележим, че инвариантът е пособие за доказателство за коректност на итеративен цикъл, а **не** за цял алгоритъм!

Задача 2.1. Даден е следният алгоритъм (зададен чрез ***псевдокод***):

```
1. Alg1(n) : // n ∈ ℕ0
2.     s ← 1;
3.     for i ← 1 to n
4.         s ← s * 2;
5.     return s;
```

Какво връща той? Да се докаже формално.

Решение. Ще докажем, че $Alg1(n) = 2^n$.

Инвариант

При всяко k-то достигане на ред 3 имаме, че $s = 2^{k-1}$ (k броим от 1)

База. При $k = 1$-во достигане на ред 3 имаме, че $s \overset{\text{ред 3}}{=} 1 = 2^0 = 2^{k-1}$.

Поддръжка. Нека е вярно за някое k-то непоследно достигане на ред 3, т.е. имаме $s = 2^{k-1}$. Сега се изпълнява тялото на цикъла, т.е. $s_{new} \overset{\text{ред 4}}{=} s_{old} * 2 \overset{\text{ИП}}{=} 2^{k-1} * 2 = 2^k = 2^{(k+1)-1}$. След изпълнение на тялото на цикъла отново се връщаме на ред 3, като вече го достигаме за $(k+1)$-ви път (може да е последно достигане) и имаме $s = 2^{(k+1)-1}$.

Терминация. При $k = (n+1)$-то достигане на ред 3 тялото на цикъла няма да се изпълни (за първи път). От инварианта имаме, че $s = 2^{(n+1)-1} = 2^n$. Директно връщаме $s = 2^n$ на ред 5.

Задача 2.2. Даден е следният алгоритъм:

```
1. Alg2(A[1..n]) : // n ∈ ℕ⁺ , A ∈ ℝⁿ
2.     s ← 0;
3.     for i ← 1 to n
4.         s ← s + A[i];
5.     return s;
```

Какво връща той? Да се докаже формално.

Решение. Ще докажем, че $Alg2(A[1..n]) = \sum_{j=1}^{n} A[j]$.

Инвариант

При всяко k-то достигане на ред 3 имаме:

- $i = k$ (в рамките на курса ще го считаме за тривиално при *стандартни* for-цикли и няма да го доказваме)
- $s = \sum_{j=1}^{k-1} A[j]$

База. При $k = 1$-во достигане на ред 3 имаме, че $s \overset{\text{ред 2}}{=} 0 = \sum_{j=1}^{0} A[j] = \sum_{j=1}^{k-1} A[j]$.

Поддръжка. Нека е вярно за някое непоследно достигане на ред 3, т.е. имаме $s = \sum_{j=1}^{k-1} A[j]$.

Сега се изпълнява тялото на цикъла $s_{new} \overset{\text{ред 4}}{=} s_{old} + A[i] = s_{old} + A[k] \overset{\text{ИП}}{=} \sum_{j=1}^{k-1} A[j] + A[k] = \sum_{j=1}^{k} A[j]$.

Терминация. При $k = (n+1)$-то достигане на ред 3 тялото на цикъла няма да се изпълни (за първи път). От инварианта имаме, че $s = \sum_{j=1}^{n} A[j]$. Директно връщаме s на ред 5.

2.2 Рекурсивни алгоритми

За разлика от итеративните алгоритми, коректност на рекурсивни алгоритми ще доказваме чрез добре познатата ни ***пълна математическа индукция*** състояща се от три фази - ***база***, ***индуктивна хипотеза*** и ***индуктивна стъпка***.

Забележка 2.2

Индукцията е пособие за доказателство за коректност на цял алгоритъм!

Забележка 2.3: Рекурсивен алгоритъм, съдържащ итеративен цикъл

За да докажем рекурсивен алгоритъм, съдържащ итеративен (един или повече) цикъл, то трябва да направим доказателство за коректност на всеки итеративен цикъл (използвайки ***инвариант***) след което да използваме ***пълна математическа индукция*** за доказателство за коректност на целия алгоритъм.

Задача 2.3. Даден е следният алгоритъм:

```
1. Alg3(a, n) : // a ∈ ℝ, n ∈ ℕ_0
2.   if n = 0 then
3.     return 1;
4.   if n ≡ 0 (mod 2) then
5.     return Alg3(a * a, n/2);
6.   return a * Alg3(a, n − 1);
```

Какво връща той? Да се докаже формално.

Решение. Ще докажем, че $Alg3(a,n) = a^n$ чрез пълна математическа индукция по n. Забележете, че $0^0 = 1$ по теоретико-множествена дефиниция (броят функции от $\emptyset$ в $\emptyset$ е единица).

База. $(n = 0)$ Директно имаме $Alg3(a,0) \overset{\text{2-3}}{\underset{\text{ред}}{=}} 1$.

Индуктивна хипотеза. Нека $\big(\forall m \leq n\big)\big(Alg3(a,m) = a^m\big)$.

Индуктивна стъпка. Ще докажем за $n+1 > 0$. Разглеждаме два случая:

сл.1 $(n+1) \equiv 0\,(mod\,2)$
Тъй като $n+1 > 0$, то не влизаме в тялото на if на ред 2-3. Знаем, че $n+1 > 0$ и $(n+1) \equiv 0\,(mod\,2)$, значи имаме $n+1 \geq 2$. Откъдето заключваме, че $\frac{n+1}{2} > 0$. Влизаме в тялото на if на ред 4-5, като директно връщаме $Alg3(a*a, \underbrace{\frac{n+1}{2}}_{>0}) \overset{\text{ИХ}}{=} (a*a)^{\frac{n+1}{2}} = a^{n+1}$.

сл.2 $(n+1) \equiv 1\,(mod\,2)$
Тъй като $n+1 > 0$, то не влизаме в тялото на if на ред 2-3. От друга страна, не влизаме в тялото на if на ред 4-5 заради $(n+1) \equiv 1\,(mod\,2)$. Тоест достигаме ред 6, където директно връщаме $a * Alg3(a,(n+1)-1) = a * Alg3(a,n) \overset{\text{ИХ}}{=} a * a^n = a^{n+1}$.

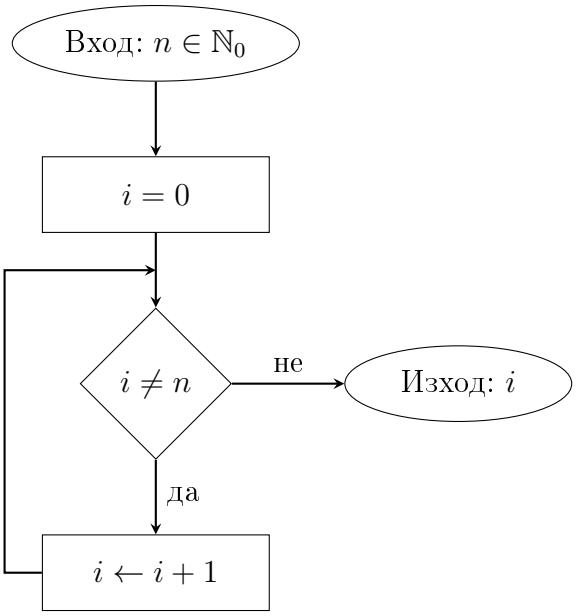
2.3 Блок схеми

В Забележка 2.1 споменахме, че ***инвариант*** ни служи за доказване само на итеративни цикли, а не на цели алгоритми, а в Забележка 2.2 казахме, че използваме ***индукция*** за доказване на коректността на цели алгоритми. Това поражда въпроса: „С какво пособие доказваме итеративни алгоритми?“. Отговорът: чрез ***блок схеми***.

Забележка 2.4

Разглежданите итеративни алгоритми в настоящия курс са достатъчно прости, т.е. целият алгоритъм всъщност е само един (най-много два) цикъл(а). За тези алгоритми е напълно приемливо да се използва ***инвариант*** за доказателство за коректност на целия алгоритъм, но когато алгоритъмът не е просто един цикъл, то тогава се налага да доказваме коректност чрез ***блок схеми***. Поради тази причина в рамките на текущия курс не се изискват познания за ***блок схеми***.

Задача 2.4. Даден е следният алгоритъм (зададен чрез ***блок схема***):



Какво връща той? Да се докаже формално.

Решение. Ще докажем, че алгоритъмът връща n.

Дефинираме три предиката (по един за всяка *не правоъгълна кутийка*):

- $I(n) \overset{def}{\leftrightarrow} n \in \mathbb{N}_0$
- $L(n, i) \overset{def}{\leftrightarrow} (n \in \mathbb{N}_0 \wedge i \in \mathbb{N}_0)$
- $O(n, i) \overset{def}{\leftrightarrow} i = n$

Ще докажем следните три импликации (връзките между *не правоъгълните кутийки*):

- $(I(n) \wedge i = 0) \to L(n, i)$
- $(L(n, i) \wedge i \neq n) \to L(n, i + 1)$
- $(L(n, i) \wedge \neg(i \neq n)) \to O(n, i)$

Ето и пълното доказателство на горните три импликации:

- $(I(n) \wedge i = 0) \to L(n, i)$
  Нека са изпълнени $\underbrace{n \in \mathbb{N}_0}_{I(n)}$ и $i = 0$. Чудим се дали е изпълнено $\underbrace{n \in \mathbb{N}_0 \text{ и } i \in \mathbb{N}_0}_{L(n,i)}$.
  Очевидно е изпълнено.
- $(L(n, i) \wedge i \neq n) \to L(n, i + 1)$
  Нека са изпълнени $\underbrace{n \in \mathbb{N}_0 \text{ и } i \in \mathbb{N}_0}_{L(n,i)}$ и $i \neq n$. Чудим се дали е изпълнено $\underbrace{n \in \mathbb{N}_0 \text{ и } (i+1) \in \mathbb{N}_0}_{L(n,i)}$.
  Очевидно е изпълнено.
- $(L(n, i) \wedge \neg(i \neq n)) \to O(n, i)$
  Нека са изпълнени $\underbrace{n \in \mathbb{N}_0 \text{ и } i \in \mathbb{N}_0}_{L(n,i)}$ и $\underbrace{\neg(i \neq n)}_{\text{т.е. } i=n}$. Чудим се дали е изпълнено $\underbrace{i = n}_{O(n,i)}$.
  Очевидно е изпълнено.

Тоест доказахме, че достигайки *кутийката за изход* е изпълнен предикатът $\underbrace{i = n}_{O(n,i)}$. Тоест връщаният резултат наистина е n.

2.4 Често срещани грешки

2.4.1 Бъркане на позицията на инварианта при do-while цикъл

Без да се замисли човек, е лесно да се направи тази грешка от невнимание. Да разгледаме следния псевдокод за илюстрация:

```
1. Alg4(n) : // n ∈ ℕ₀
2.     i ← 0;
3.     do
4.         i ← i + 1;
5.     while i < n;
6.     return n;
```

Инвариант: ГРЕШЕН

При всяко k-то достигане на ред **3**...

Инвариант: КОРЕКТЕН

При всяко k-то достигане на ред **5**...

Съобразете защо!

2.4.2 Некоректна терминация

За разлика от предходната грешка, тази е значително по-трудна за съобразяване. Да разгледаме следния алгоритъм:

```
1. Alg5(n) : // n ∈ ℕ_0
2.     i ← 0;
3.     while i < n do
4.         i ← i + 1;
5.     return n;
```

Ясно е, че $Alg5(n) = n$. Проблем настъпва, когато се опитаме да съставим инвариант. Първоначалната ни идея би била да направим следния инвариант:

Инвариант: НЕПЪЛЕН

При всяко k-то достигане на ред 3 имаме, че $i = k - 1$

Този инвариант е абсолютно верен. Нека го докажем.

База. При $k = 1$-во достигане на ред 3 имаме, че $i = 0 = 1 - 1 = k - 1$.

Поддръжка. Нека е изпълнено за някое k-то непоследно достигане на ред 3. Тъй като е непоследно достигане, то тялото на цикъла се изпълнява и достигаме отново ред 3, като имаме $i_{new} \overset{\text{ред } 4}{=} i_{old} + 1 \overset{\text{ИХ}}{=} (k-1) + 1 = k = (k+1) - 1$.

Терминация. При $(n+1)$-вото достигане на ред 3 за първи път не влизаме в тялото на while-а. От инварианта знаем, че $i = (n+1) - 1 = n$, което и връщаме директно след цикъла.

Забелязахте ли проблема?

Проблемът е, че не знаем кое е де факто **последното** достигане на ред 3. При текущия алгоритъм е очевидно за съобразяване, но аналогичен проблем може да възникне при много по-сложни алгоритми. Инвариантът е абсолютно коректно доказан (съобразете защо поддръжката е коректно доказана), но терминацията е грешна! Ето как би изглеждал правилен инвариант за алгоритъма:

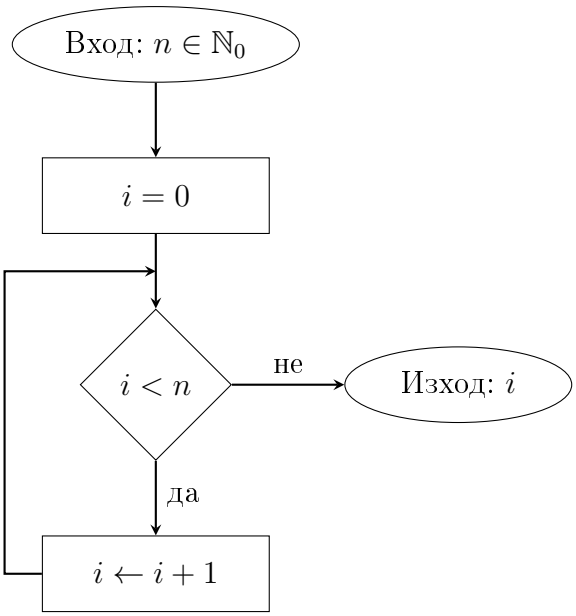
Инвариант

При всяко k-то достигане на ред 3 имаме:

- $i = k - 1$
- $i < n + 1$

Докажете, използвайки този инвариант!

Тъй като е трудно да се съобрази точно къде е проблемът, нека да разгледаме същия проблем, но зададен чрез блок схема:



Нека да разгледаме грешната идея първо:

Дефинираме три предиката (по един за всяка *не правоъгълна кутийка*):

- $I(n) \overset{def}{\leftrightarrow} n \in \mathbb{N}_0$
- $L(n,i) \overset{def}{\leftrightarrow} (n \in \mathbb{N}_0 \wedge i \in \mathbb{N}_0)$
- $O(n,i) \overset{def}{\leftrightarrow} i = n$

Ще докажем следните три импликации (връзките между *не правоъгълните кутийки*):

- $(I(n) \wedge i = 0) \to L(n,i)$
- $(L(n,i) \wedge i < n) \to L(n, i+1)$
- $(L(n,i) \wedge \neg(i < n)) \to O(n,i)$

Ето и опит за пълно доказателство на горните три импликации:

- $(I(n) \wedge i = 0) \to L(n,i)$
  Нека са изпълнени $\underbrace{n \in \mathbb{N}_0}_{I(n)}$ и $i = 0$. Чудим се дали е изпълнено $\underbrace{n \in \mathbb{N}_0 \text{ и } i \in \mathbb{N}_0}_{L(n,i)}$.
  Очевидно е изпълнено.
- $(L(n,i) \wedge i < n) \to L(n, i+1)$
  Нека са изпълнени $\underbrace{n \in \mathbb{N}_0 \text{ и } i \in \mathbb{N}_0}_{L(n,i)}$ и $i < n$. Чудим се дали е изпълнено $\underbrace{n \in \mathbb{N}_0 \text{ и } (i+1) \in \mathbb{N}_0}_{L(n,i)}$.
  Очевидно е изпълнено.
- $(L(n,i) \wedge \neg(i < n)) \to O(n,i)$
  Нека са изпълнени $\underbrace{n \in \mathbb{N}_0 \text{ и } i \in \mathbb{N}_0}_{L(n,i)}$ и $\underbrace{\neg(i < n)}_{\text{т.е. } i \geq n}$. Чудим се дали е изпълнено $\underbrace{i = n}_{O(n,i)}$. Не знаем! Единствено имаме, че $i \geq n$ (и че е естествено число), но не знаем дали е n или $n+1$, или дори може да е $n + 10000$.

Тоест не се получи, използвайки тези 3 предиката.. ще трябва да опитаме нещо по-силно.

Дефинираме три предиката (по един за всяка *не правоъгълна кутийка*):

- $I(n) \overset{def}{\leftrightarrow} n \in \mathbb{N}_0$
- $L(n,i) \overset{def}{\leftrightarrow} (n \in \mathbb{N}_0 \wedge i \in \mathbb{N}_0 \wedge \boldsymbol{i \leq n})$
- $O(n,i) \overset{def}{\leftrightarrow} i = n$

Ще докажем следните три импликации (връзките между *не правоъгълните кутийки*):

- $(I(n) \wedge i = 0) \to L(n,i)$
- $(L(n,i) \wedge i < n) \to L(n, i+1)$
- $(L(n,i) \wedge \neg(i < n)) \to O(n,i)$

Ето и пълното доказателство на горните три импликации:

- $(I(n) \wedge i = 0) \to L(n,i)$
  Нека са изпълнени $\underbrace{n \in \mathbb{N}_0}_{I(n)}$ и $i = 0$. Чудим се дали е изпълнено $\underbrace{n \in \mathbb{N}_0 \text{ и } i \in \mathbb{N}_0 \text{ и } i \leq n}_{L(n,i)}$.
  Тъй като $n \in \mathbb{N}_0$, то знаем, че $n \geq 0$, но $i = 0$, тоест имаме $n \geq i$.

- $(L(n,i) \wedge i < n) \to L(n, i+1)$
  Нека са изпълнени $\underbrace{n \in \mathbb{N}_0 \text{ и } i \in \mathbb{N}_0 \text{ и } i \leq n}_{L(n,i)}$ и $i < n$. Чудим се дали е изпълнено следното
  $\underbrace{n \in \mathbb{N}_0 \text{ и } (i+1) \in \mathbb{N}_0 \text{ и } (i+1) \leq n}_{L(n,i)}$. Тъй като имаме, че $i < n$ (и $i \in \mathbb{N}_0$), то е еквивалентно с $(i+1) \leq n$. Очевидно е, че $(i+1) \in \mathbb{N}_0$.

- $(L(n,i) \wedge \neg(i < n)) \to O(n,i)$
  Нека са изпълнени $\underbrace{n \in \mathbb{N}_0 \text{ и } i \in \mathbb{N}_0 \text{ и } i \leq n}_{L(n,i)}$ и $\underbrace{\neg(i < n)}_{\text{т.е. } i \geq n}$. Чудим се дали е изпълнено $\underbrace{i = n}_{O(n,i)}$.
  Очевидно е, тъй като имаме $i \geq n$ и $i \leq n$, то е ясно, че $i = n$.

Доказахме, че алгоритъмът връща n (при вход n).

2.5 Задачи

Задача 2.5. Даден е следният алгоритъм:

```
1.  Kadane(A[1..n]) : // n ∈ ℕ⁺ , A ∈ ℤⁿ
2.  |  c ← A[1];
3.  |  m ← A[1];
4.  |  for i ← 2 to n
5.  |  |  if A[i] + c > A[i] then
6.  |  |  |  c ← c + A[i];
7.  |  |  else
8.  |  |  |  c ← A[i];
9.  |  |  if c > m then
10. |  |  |  m ← c;
11. |  return m;
```

Какво връща той? Да се докаже формално.

Решение. Ще докажем, че $Kadane(A[1..n])$ връща най-голяма сума на непразен подмасив (последователни елементи) на $A[1..n]$.

Инвариант

При всяко k-то достигане на ред 4 имаме:

- $i = k + 1$
- c е най-голяма сума на непразен подмасив на $A[1..k]$, завършващ в индекс k
- m е най-голяма сума на непразен подмасив на $A[1..k]$

База. При $k = 1$-во достигане на ред 4 имаме:

- $i \overset{\text{ред 4}}{=} 2 = k + 1$
- c изпълнява инварианта
- m изпълнява инварианта

Поддръжка. Нека са изпълнени за някое k-то непоследно достигане на ред 4 следните:

- $i_{old} = k + 1$
- c_{old} е най-голяма сума на непразен подмасив на $A[1..k]$, завършващ в индекс k
- m_{old} е най-голяма сума на непразен подмасив на $A[1..k]$

Ще докажем за $(k + 1)$-во достигане на ред 4:

- $i_{new} = i_{old} + 1 \overset{\text{ИХ}}{=} (k + 1) + 1$

- Ще използваме следния факт наготово: $c_{new} = max\{c_{old} + A[i_{old}], A[i_{old}]\}$ (докажете). Допускаме, че c_{new} не е най-голяма сума на непразен подмасив на $A[1..k+1]$, завършваща в индекс $k+1$, и нека означим с t тази най-голяма сума. Тоест имаме, че $t > c_{new} = max\{c_{old} + A[i_{old}], A[i_{old}]\} \geq c_{old} + A[i_{old}]$. Като прехвърлим $A[i_{old}]$ от другата страна получаваме $t - A[i_{old}] > c_{old}$. Допуснахме, че t е най-голяма сума сума на непразен подмасив на $A[1..k+1]$, завършващ в индекс $k+1$. Тогава $t - A[i_{old}] \overset{\text{ИХ}}{=} t - A[k+1]$ е сума на **(потенциално празен)** подмасив на $A[1..k]$, завършващ в индекс k. Разглеждаме два случая:

  сл.1 (подмасивът е празен)
  Тогава имаме, че t е сума само на последния елемент, т.е. $t = A[k+1]$. Знаем, че $c_{new} = max\{c_{old} + A[i_{old}], A[i_{old}]\}$. Нека разгледаме два подслучая:

  сл.1.1 ($c_{old} > 0$)
  Тогава имаме $c_{new} = \{c_{old} + A[i_{old}], A[i_{old}]\} = c_{old} + A[i_{old}] > A[i_{old}] \overset{\text{ИХ}}{=} A[k+1] = t$. Тоест излезе, че $c_{new} > t$ - противоречие (с това, че $t > c_{new}$).

  сл.1.2 ($c_{old} \leq 0$)
  Тогава имаме $c_{new} = \{c_{old} + A[i_{old}], A[i_{old}]\} = A[i_{old}] \overset{\text{ИХ}}{=} A[k+1] = t$. Тоест излезе, че $c_{new} = t$ – противоречие (с това, че $t > c_{new}$).

  сл.2 (подмасива не е празен)
  Тогава имаме, че $t - A[k+1]$ е сума на непразен подмасив на $A[1..k]$, завършваща в индекс k, и знаем още, че $t - A[k+1] > c_{old}$. От друга страна, от ИХ имаме, че c_{old} е максима сума на непразен подмасив на $A[1..k]$, завършваща в индекс k, т.е. $c_{old} \geq t - A[k+1] = t - A[i_{old}]$ – противоречие (с това, че $t - A[i_{old}] > c_{old}$).

  Следователно c_{new} най-голяма сума на непразен подмасив на $A[1..k+1]$, завършващ в индекс $k+1$.

- Ще използваме следния факт наготово: $m_{new} = max\{c_{new}, m_{old}\}$ (докажете). Опитваме се да докажем, че m_{new} е максимална сума на непразен подмасив на $A[1..k+1]$. Разглеждаме следните два случая:

  сл.1 ($A[k+1]$ участва в някоя максимална сума на непразен подмасив на $A[1..k+1]$)
  Тогава е ясно, че c_{new} е тази сума. За пълнота разглеждаме два подслучая:

  сл.1.1 (има точно една максимална сума на непразен подмасив на $A[1..k+1]$)
  Тогава ще влезем в тялото на if-а на ред 9-10 и ще имаме $m_{new} = c_{new}$.

  сл.1.2 (има повече от една)
  Тогава няма да влезем в тялото на същия if и ще имаме $m_{new} = m_{old} (= c_{new})$.

  сл.2 ($A[k+1]$ не участва в никоя максимална сума на непразен подмасив на $A[1..k+1]$)
  Тогава имаме $m_{new} = m_{old}$, но от ИХ за m_{old} директно имаме твърдението за m_{new}.

  Следователно m_{new} е най-голяма сума на непразен подмасив на $A[1..k+1]$

Терминация. При $k = n$ за първия път не влизаме в тялото на for-а. Директно връщаме m, което от инварианта знаем, че е максимална сума на непразен подмасив на $A[1..n]$.

Задача 2.6. Дадена е кутия с 53 сини топки и 42 червени топки. В кутията няма други топки. Извън кутията имаме неограничен брой сини и червени топки. Даден е следният алгоритъм:

```
1. AlgBall() :
2.     while не остане една топка в кутията do
3.         извади две случайни топки;
4.         if двете топки са еднакъв цвят then
5.             добави една червена;
6.         else
7.             добави една синя;
```

Какъв е цветът на топката, която е останала самичка?

Решение. Ще докажем, че цветът на топката, която е останала, е син.

Инвариант

При всяко k-то достигане на ред 2 имаме:

- броят топки в кутията е $max\{95 - k + 1, 1\}$
- броят сини топки в кутията е нечетен

Докажете за упражнение инварианта и покажете как от него следва цветът на останалата самичка топка!

Задача 2.7. Даден е следният алгоритъм:

```
1. Pred(A[1..n]) : // n ∈ ℕ_0 , A ∈ ℝ^n
2.     for i ← 1 to n − 1
3.         for j ← i + 1 to n
4.             if A[i] = A[j] then
5.                 return TRUE;
6.     return FALSE;
```

Какво връща той? Да се докаже формално.

Решение. Ще докажем, че $Pred(A[1..n])$ връща истина тогава и само тогава, когато има повтарящи се елементи (и лъжа иначе) в $A[1..n]$. За целта ще ни трябват два инварианта – един за външния цикъл и един за вътрешния цикъл.

Инвариант: Вътрешен цикъл

При всяко k-то достигане на ред 3 (релативно за текущото изпълнение на външният цикъл) имаме, че $A[i+1], \ldots, A[i+k-1]$ са различни от $A[i]$ и $i < j$

Инвариант: Външен цикъл

При всяко k-то достигане на ред 2 имаме, че $A[1], \ldots, A[k-1]$ са уникални в $A[1..n]$

Доказателството на двата инварианта остава за упражнение. Сега остана да използваме инвариантите за да докажем коректността. Имаме две възможни места за изход от програмата - ред 5 и ред 6. Нека да ги разгледаме:

сл.1 (излизаме през ред 5)
От вътрешния инвариант, знаем че $i < j$. Също така знаем, че $A[i] = A[j]$ тъй като сме в тялото на if от ред 4-5. Тоест имаме два еднакви елемента в масива и връщаме истина, к.т.д.д. (Тук **измамихме**! Можете ли да откриете проблема и да го оправите?)

сл.2 (излизаме през ред 5)
От външния инвариант директно следва, че в масива всички елементи са уникални и връщаме лъжа, к.т.д.д.

Забележка. При ограничено време на контролно приоритизирайте доказателството на най-външния инвариант.

Забележка 2.5: Инвариантът не е всичко

Понякога трябва бонус работа (след инварианта), за да се докаже коректност на алгоритъм! Примери за това са **Задача 2.6** и **Задача 2.7**.

Глава 3

Анализ на времевата сложност на алгоритми

3.1 Итеративни алгоритми

Изложените по-долу суми, може да се ползват без доказателство на контролни и домашни:

$$\sum_{i=1}^{n} 1 = n = \Theta(n) \qquad \sum_{i=1}^{n} \Theta(1) = \Theta(n) \qquad \sum_{i=c}^{n} = n - c + 1 = \Theta(n)$$

$$\sum_{i=c}^{n} \Theta(1) = \Theta(n) \qquad \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2) \qquad \sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3) \qquad \sum_{i=1}^{n} \Theta(i^2) = \Theta(n^3) \qquad \sum_{\substack{i=1 \\ i=i+k}}^{n} 1 = \lfloor \frac{n}{k} \rfloor = \Theta(n)$$

$$\sum_{\substack{i=1 \\ i=i+k}}^{n} \Theta(1) = \Theta(n) \qquad \sum_{i=1}^{n} \frac{1}{i} = \Theta(ln(n)) \qquad \sum_{i=1}^{n} \Theta(\frac{1}{i}) = \Theta(ln(n))$$

Забележка. Употребата на тета-нотацията вляво на $=$ се нарича ***анонимна функция***

3.1.1 Интегрален критерий

В настоящия курс ще разглеждаме само частен случай на интегралния критерий, който ще ни бъде достатъчен. Това е един от най-силните инструменти, с които разполагаме, за да намираме асимптотиката на дадена сума.

Теорема 3.1: Интегрален критерий (частен случай)

Нека f е асимптотично положителна функция и нека $n_0 \in \mathbb{N}_0$ е свидетел за това. Нека $\big(\exists m > 0\big)\big(\exists M > 0\big)\big(\forall n > n_0\big)\big(mf(n) \leq \min\limits_{x \in [n,n+1]} f(x) \leq \max\limits_{x \in [n,n+1]} f(x) \leq Mf(n)\big)$.

Тогава е изпълнено: $\sum\limits_{i=n_0}^{n} f(i) \asymp \int_{n_0}^{n} f(x)dx$.

Забележка. Приложен интегрален критерий без посочени свидетели m, M и n_0 и/или без обосновка на трите неравенства, носи **нула** точки!

3.1.1.1 Добре разпределени функции

Една функция ще наричаме ***добре разпределена*** (между естествените и реалните числа), ако отговаря на допълнителното условие в **Теорема 3.1** за съществуване на такива константи m и M. Нека да разгледаме малко примери за да придобием интуиция кои функции са ***добре разпределени***:

Пример 3.1. $f(x) = x^2$
Ще проверим, че условието

$$mn^2 \leq \min_{x\in[n,n+1]} f(x) \leq \max_{x\in[n,n+1]} f(x) \leq Mn^2 \tag{3.1}$$

е изпълнено за $m = 1, M = 4$ и произволно $n \geq n_0 = 1$:

$$1n^2 \leq n^2 \leq (n+1)^2 \leq 4n^2$$

Първите две неравенства очевидно са верни. Остава да проверим:

$$(n+1)^2 \leq 4n^2$$

След като разкрием скобите получаваме:

$$n^2 + 2n + 1 \leq 4n^2$$

или още

$$3n^2 - 2n - 1 \geq 0$$

Сега намираме корените $x_1 = -\frac{1}{3}, x_2 = 1$ и виждаме, че $3n^2 - 2n - 1 \geq 0$ за $n \geq 1$, което искахме да докажем.

Разбира се, не е задължително да има единствени такива m, M, n_0. Даже напротив - ако има едно решение, то има безкрай много решения. Да разгледаме друго решение. Ще се убедим, че е изпълнено за $m = \frac{1}{3}, M = 1.005$ и произволно $n \geq n_0 = 41$:

$$\frac{1}{3}n^2 \leq n^2 \leq (n+1)^2 \leq 1.005n^2$$

Отново първите две неравенства очевидно са верни. Остава да проверим:

$$(n+1)^2 \leq 1.005n^2$$

или още

$$0.05n^2 - 2n - 1 \geq 0$$

Сега намираме корените $x_1 = 20 - 2\sqrt{105} \approx -0.4939, x_2 = 20 + 2\sqrt{105} \approx 40.4939$ и виждаме, че $0.05n^2 - 2n - 1 \geq 0$ за $n \geq n_0 = 41 > x_2 \approx 40.4939$.

В случая кое да е $m \in (0, 1]$ и кое да е $M \in (1, +\infty)$ ни вършат работа, като в зависимост от избора ни на M, трябва да изберем подходящо n_0.

Пример 3.2. $f(x) = x^\alpha$
В зависимост от α ще проверим, че условието 3.1 е изпълнено за

сл.1 $(\alpha \geq 0)$ $m = 1, M = 2^\alpha, n_0 = 1$
Тоест трябва да се убедим, че следните неравенства са вярни:

$$1n^\alpha \leq n^\alpha \leq (n+1)^\alpha \leq 2^\alpha n^\alpha$$

Отново първите две са очевидни. Остава да се убедим само в последното неравенство:

$$(n+1)^\alpha \leq (2n)^\alpha$$

Тъй като $n_0 > 0$ и работим с $n \geq n_0$, то $2n > 0$ и можем да разделим на него (т.е. не делим на нула и не се сменя посоката на неравенството):

$$\left(\frac{n+1}{2n}\right)^\alpha \leq 1$$

Лесно съобразяваме, че при $n \geq n_0 = 1$ и $\alpha \geq 0$ неравенството е изпълнено:

$$\left(\underbrace{\frac{n+1}{2n}}_{\leq 1}\right)^\alpha \leq 1$$

сл.2 $(\alpha < 0)$ $m = 2^\alpha, M = 1, n_0 = 1$
Тоест трябва да се убедим, че следните неравенства са вярни:

$$2^\alpha n^\alpha \leq (n+1)^\alpha \leq n^\alpha \leq 1n^\alpha$$

Ще проверим първото неравенство, другите две са очевидни:

$$(2n)^\alpha \leq (n+1)^\alpha$$

Тъй като $n_0 > 0$ и работим с $n \geq n_0$, то $2n > 0$ и можем да разделим на него (т.е. не делим на нула и не се сменя посоката на неравенството):

$$\left(\frac{n+1}{2n}\right)^\alpha \geq 1$$

Лесно съобразяваме, че при $n \geq n_0 = 1$ и $\alpha < 0$ неравенството е изпълнено:

$$\left(\underbrace{\frac{n+1}{2n}}_{\leq 1}\right)^\alpha = \left(\underbrace{\frac{2n}{n+1}}_{\geq 1}\right)^{|\alpha|} \geq 1$$

Пример 3.3. $f(x) = \alpha^n,\ \alpha > 0$
В зависимост от α ще проверим, че условието 3.1 е изпълнено за

сл.1 $(\alpha \geq 1)$ $m = 1, M = \alpha, n_0 = 1$
Тоест трябва да се убедим, че следните неравенства са вярни:

$$1\alpha^n \leq \alpha^n \leq \alpha^{n+1} \leq \alpha\alpha^n$$

Очевидно и трите неравенства са верни.

сл.2 $(0 < \alpha < 1)$ $m = \alpha, M = 1, n_0 = 1$
Тоест трябва да се убедим, че следните неравенства са верни:

$$\alpha\alpha^n \leq \alpha^{n+1} \leq \alpha^n \leq 1\alpha^n$$

Очевидно и трите неравенства са верни.

Пример 3.4. $f(x) = x^x$
Това е пример за функция, която **не** е добре разпределена. Убедете се защо!

Пример 3.5. $f(x) = x!$
Това е друг пример за функция, която **не** е добре разпределена. Убедете се защо!

3.1.1.2 Приложения

Приложение 3.1. $f(x) = x^\alpha$
Очевидно функията е асимптотично положителна и в **Пример 3.2** доказахме, че тя е добре разпределена. Тогава може да приложим интегралния критерий и в зависимост какво n_0 сме избрали получаваме:

$$\sum_{i=n_0}^{n} i^\alpha \asymp \int_{n_0}^{n} x^\alpha dx = \begin{cases} \Theta(n^{\alpha+1}) & , \alpha > -1 \\ \Theta(ln(n)) & , \alpha = -1 \\ \Theta(1) & , \alpha < -1 \end{cases}$$

Приложение 3.2. $f(x) = \alpha^x,\ \alpha > 0$
Очевидно функията е асимптотично положителна и в **Пример 3.3** доказахме, че тя е добре разпределена. Тогава може да приложим интегралния критерий и в зависимост какво n_0 сме избрали получаваме:

$$\sum_{i=n_0}^{n} \alpha^i \asymp \int_{n_0}^{n} \alpha^x \, dx$$

Сега разглеждаме 3 случая в зависимост от α:

сл.1 $(\alpha > 1)$

$$\int_{n_0}^{n} \alpha^x \, dx = \frac{\alpha^n - \alpha^{n_0}}{ln(\alpha)} = \Theta(\alpha^n)$$

сл.2 $(\alpha = 1)$

$$\int_{n_0}^{n} \alpha \, dx = \alpha(n - n_0) = \Theta(n)$$

сл.3 $(0 < \alpha < 1)$

$$\int_{n_0}^{n} \alpha^x \, dx = \frac{\alpha^n - \alpha^{n_0}}{ln(\alpha)} = \Theta(1)$$

Тъй като числителя $\alpha^n - \alpha^{n_0}$ и знаменателя $ln(\alpha)$ са строго по-малки от 0, то цялата дроб е строго положителна. Тогава $\lim_{n\to\infty} \frac{\alpha^n - \alpha^{n_0}}{ln(\alpha)} = \frac{-\alpha^{n_0}}{ln(\alpha)} > 0$ е просто константа, откъдето имаме асимптотика $\Theta(1)$.

Тоест получихме:

$$\sum_{i=n_0}^{n} \alpha^i = \begin{cases} \Theta(\alpha^n) & , \alpha > 1 \\ \Theta(n) & , \alpha = 1 \\ \Theta(1) & , 0 < \alpha < 1 \end{cases}$$

Приложение 3.3. Каква е асимптотиката на:

(а) $\sum_{i=1}^{n} \frac{1}{i}$ (б) $\sum_{i=1}^{n} \frac{1}{\sqrt{i}}$ (в) $\sum_{i=1}^{n} \frac{1}{i^2}$

Решение. Това е частен случай на **Приложение 3.1**. Тоест имаме:

(а) $\sum_{i=1}^{n} \frac{1}{i} = \Theta(ln(n))$ (б) $\sum_{i=1}^{n} \frac{1}{\sqrt{i}} = \Theta(\sqrt{n})$ (в) $\sum_{i=1}^{n} \frac{1}{i^2} = \Theta(1)$

Приложение 3.4. Каква е асимптотиката на $\sum_{i=1}^{n} ln(i)$?

Решение. Очевидно е, че функцията $f(x) = ln(x)$ е асимптотично положителна. Остана да покажем, че тя е добре разпределена. Тоест трябва да намерим такива $m, M > 0$ и $n_0 \in \mathbb{N}_0$, че да е вярно неравенство 3.1. Ще покажем, че $m = 1, M = 2, n_0 = 2$ са свидели:

$$1ln(n) \leq ln(n) \leq ln(n+1) \leq 2ln(n)$$

Очевидно първите две неравенства са изпълнени. Остава да проверим третото:

$$ln(n+1) \leq 2ln(n) = ln(n^2)$$

Нека се убедим първо, че следното неравенство е изпълнено за $n \geq n_0 = 2$:

$$n + 1 \leq n^2$$

или още

$$n^2 - n - 1 \geq 0$$

Сега намираме корените $x_1 = \frac{1-\sqrt{5}}{2} \approx -0.618, x_2 = \frac{1+\sqrt{5}}{2} \approx 1.618$ и виждаме, че $n^2 - n - 1 \geq 0$ за $n \geq n_0 = 2 > x_2 \approx 1.618$. Знаем, че логаритмуването на асимптотично положителни и неограничени отгоре функции запазва посоката на неравенството, откъдето получаваме:

$$ln(n+1) \leq ln(n^2) = 2ln(n)$$

Сега вече може да приложим интегралния критерий, откъдето получаваме:

$$\sum_{i=2}^{n} ln(i) \asymp \int_2^n ln(x)\,dx = ln(x)x\Big|_2^n - \int_2^n x\,d(ln(x)) = nln(n) - 2ln(2) - \int_2^n \frac{x}{x}\,dx =$$
$$= nln(n) - 2ln(2) - x\Big|_2^n = nln(n) - n + 2 - 2ln(2) = \Theta(n.ln(n))$$

Забележете, че в оригиналната задача, сумата започваше от 1, а интегралния критерий ни даде асимптотика от 2 нагоре. Това обаче не е проблем, тъй като сме изпуснали $n_0 - 1$ на брой крайни събираеми (което е константа). Тоест имаме:

$$\sum_{i=1}^{n} ln(i) = const + \sum_{i=2}^{n} ln(i) = const + \Theta(n.ln(n)) = \Theta(n.ln(n))$$

В случая даже изпуснатото събираемо е $ln(1) = 0$, но в общия случай това не е така.

Приложение 3.5. Каква е асимптотиката на $\sum_{i=1}^{n} \frac{ln(i)}{i}$?

Решение. За упражнение докажете (използвайки интегралния критерий), че:

$$\sum_{i=1}^{n} \frac{ln(i)}{i} \asymp \Theta(ln^2(n))$$

3.1.2 Задачи

Основната идея е да заместим всяка атомарна операция с константа (или за по-лесно единица - не се отразява на асимптотиката), а циклите с математическа сума. Нека разгледаме няколко примерни задачи:

Задача 3.1. Даден е следният алгоритъм:

```
1. Func(n) : // n ∈ ℕ⁺
2. |  print 'a';
3. |  for i ← 1 to n
4. |  |  print 'b';
5. |  fastprint 'c';
6. |  print 'd';
```

Каква е сложността му по време (спрямо n)?

Решение. Сложността му по време е $c_{print} + \sum_{i=1}^{n}(c_{print} + c_{checkend} + c_{increment}) + c_{fastprint} + c_{print} = 2c_{print} + n(c_{print} + c_{checkend} + c_{increment}) + c_{fastprint} = \Theta(n)$.

Забележка. За чистота се разбираме да пишем 1 вместо c_{name} за всички атомарни операции.

Задача 3.2. Даден е следният алгоритъм:

```
1. Func(n) : // n ∈ ℕ⁺
2. |  for i ← 1 to n
3. |  |  for j ← 1 to n
4. |  |  |  print 'a';
```

Каква е сложността му по време (спрямо n)?

Решение. Сложността му по време е $\sum_{i=1}^{n}\sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n\sum_{i=1}^{n} 1 = n\,n = \Theta(n^2)$.

Задача 3.3. Даден е следният алгоритъм:

```
1. Func(n) : // n ∈ ℕ⁺
2. |  for i ← 1 to n
3. |  |  for j ← 1 to i
4. |  |  |  print 'a';
```

Каква е сложността му по време (спрямо n)?

Решение. Сложността му по време е $\sum_{i=1}^{n}\sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{(n+1)n}{2} = \Theta(n^2)$.

Задача 3.4. Даден е следният алгоритъм:

```
1. Func(n) : // n ∈ ℕ⁺
2. |  for i ← 1 to n
3. |  |  for j ← 1 to n with step i
4. |  |  |  print 'a';
```

Каква е сложността му по време (спрямо n)?

Решение. Сложността му по време е $\sum_{i=1}^{n}\sum_{\substack{j=1\\ j=j+i}}^{n} 1 = \sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor \asymp \sum_{i=1}^{n}\frac{n}{i} = n\sum_{i=1}^{n}\frac{1}{i} \asymp nln(n) = \Theta(n\,ln(n))$.

Забележка 3.1: Сложност спрямо големината на входа

Забележете, че големината на входа на горните четири задачи (а и на долната) е все $log(n)$. Тоест сложността по време (спрямо големината на входа $m = log(n)$) на горните четири задачи е съответно $\Theta(2^m), \Theta((2^m)^2) = \Theta(4^m), \Theta(4^m)$ и $\Theta(2^m log(2^m)) = \Theta(m2^m)$.

Задача 3.5. Даден е следният алгоритъм:

```
1. Func(n) : // n ∈ ℕ⁺
2.     for i ← 1 to n
3.         for j ← i to 2^i
4.             if j < n then
5.                 for k ← 1 to n with step i
6.                     print 'a';
7.             print 'b';
```

Каква е сложността му по време (спрямо n, а спрямо големината на входа $m = log(n)$)?

Решение. Сложността му по време е:

$$\underbrace{\sum_{i=1}^{n}\sum_{j=i}^{2^i} 1}_{\text{print 'b'}} + \underbrace{\sum_{i=1}^{\lfloor log(n)\rfloor}\sum_{j=i}^{2^i}\sum_{\substack{k=1\\k=k+i}}^{n} 1}_{\text{print 'a' } (i\le\lfloor log(n)\rfloor)} + \underbrace{\sum_{i=\lfloor log(n)\rfloor+1}^{n}\sum_{j=i}^{n}\sum_{\substack{k=1\\k=k+i}}^{n} 1}_{\text{print 'a' } (i>\lfloor log(n)\rfloor)} \tag{3.2}$$

Нека да разгледаме трите суми поотделно. Да започнем с първата:

$$\sum_{i=1}^{n}\sum_{j=i}^{2^i} 1 = \sum_{i=1}^{n}\Big(2^i - i + 1\Big) = \sum_{i=1}^{n} 2^i - \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1 \asymp 2^n - n^2 + n = \Theta(2^n)$$

Сега да разгледаме втората (като премахнем закръглянето - не променя асимптотиката):

$$\begin{aligned}\sum_{i=1}^{log(n)}\sum_{j=i}^{2^i}\sum_{\substack{k=1\\k=k+i}}^{n} 1 &\asymp \sum_{i=1}^{log(n)}\sum_{j=i}^{2^i}\frac{n}{i} = n\sum_{i=1}^{log(n)}\left(\frac{1}{i}\sum_{j=i}^{2^i} 1\right) = n\sum_{i=1}^{log(n)}\left(\frac{1}{i}\Big(2^i - i + 1\Big)\right) = \\ &= n\left(\sum_{i=1}^{log(n)}\frac{2^i}{i} - \sum_{i=1}^{log(n)}\frac{i}{i} + \sum_{i=1}^{log(n)}\frac{1}{i}\right) \le n\left(\sum_{i=1}^{log(n)} 2^i - \sum_{i=1}^{log(n)} 1 + \sum_{i=1}^{log(n)}\frac{1}{i}\right) \asymp \\ &\asymp n\Big(2^{log(n)} - log(n) + ln(log(n))\Big) \asymp n\Big(n - log(n) + log^{(2)}(n)\Big) \asymp n^2\end{aligned}$$

В крайна сметка за втората сума получихме:

$$\sum_{i=1}^{log(n)}\sum_{j=i}^{2^i}\sum_{\substack{k=1\\k=k+i}}^{n} 1 = O(n^2)$$

Обърнете внимание, че е $O(n^2)$, а не $\Theta(n^2)$ - това е заради употребата на $\le$. Разбира се може в действителност да е $\Theta(n^2)$, но $O(n^2)$ ни е достатъчно (след малко ще видим защо) да определим сложността по време на 3.2. Остана да разгледаме третата сума (отново махаме закръглянето и константата $+1$ - не променят асимптотиката):

$$\begin{aligned}\sum_{i=log(n)}^{n}\sum_{j=i}^{n}\sum_{\substack{k=1\\k=k+i}}^{n} 1 &\asymp \sum_{i=log(n)}^{n}\sum_{j=i}^{n}\frac{n}{i} = n\sum_{i=log(n)}^{n}\frac{1}{i}\sum_{j=i}^{n} 1 = n\sum_{i=log(n)}^{n}\left(\frac{1}{i}\Big(n - i + 1\Big)\right) = \\ &= n\left(n\sum_{i=log(n)}^{n}\frac{1}{i} - \sum_{i=log(n)}^{n}\frac{log(n)}{i} + \sum_{i=log(n)}^{n}\frac{1}{i}\right) = \cdots = \Theta(n^2 log(n))\end{aligned}$$

Тоест сложността по време на функцията е $\Theta(2^n) + O(n^2) + \Theta(n^2 log(n)) = \Theta(2^n) = \Theta(2^{2^m})$.

3.2 Рекурсивни алгоритми

Сложността на рекурсивни алгоритми ще определяме посредством рекурентни уравнения. По тази причина рекурентните уравнения, които ще разглеждаме ще са със строго намаляващ аргумент, всички нехомогенни събираеми отдясно на $=$ ще се срещат с положителен знак и всички основи на експоненти в нехомогенната част ще бъдат положителни. Също така да обърнем внимание, че за всяко фиксирано $n_0 \in \mathbb{N}_0$ е изпълнено $\big(\forall n \leq n_0\big)\big(T(n) = \Theta(1)\big)$, където $T(n)$ е рекурентно уравнение. Може да си мислим за това n_0 като базата на рекурсивния алгоритъм. За удобство ще нагласяме n_0 да е някое малко число - например $0, 1, 2$ - в зависимост от конкретната задача. Съставянето на рекурентно уравнение за сложността на рекурсивен алгоритъм е тривиално в повечето случаи. Поради тази причина ще се съсредоточим върху методи за намиране на асимптотиката на рекурентни уравнения.

3.2.1 Характеристично уравнение

Започваме с метод, който ви е добре познат от ДСТР. За разлика от ДСТР, тук не ни интересува точното решение, а само асимптотиката.

Задача 3.6. Каква е асимптотиката на $T(n) = 4T(n-2) + n2^n + 4 \cdot 3^n$?

Решение.

- (Хомогенна част)
  $x^2 = 4 \ \mapsto \ x_{1,2} = \pm 2 \ \mapsto \ \{2, -2\}_M$
- (Нехомогенна част)
  - $n2^n \ \mapsto \ \{2, 2\}_M$
  - $4 \cdot 3^n \ \mapsto \ \{3\}_M$

Като обединим всички мултимножества получаваме: $\{-2, 2, 2, 2, 3\}_M$. Оттук получаваме, че $T(n) = c_1(-2)^n + c_2 2^n + c_3 n 2^n + c_4 n^2 2^n + c_5 3^n = \Theta(3^n)$.

Забележка. Забележете, че поради естеството на рекурентните уравнения (строго намаляващ аргумент, положителен знак отдясно на $=$ и положителна основа на експонентата в нехомогенната част), то ни е гарантирано, че най-голямото (по модул) число в крайното мултимножество ще е положително и при това ще се среща със строго положителна константа в явния запис на рекурентното уравнение. Поради тази причина в **Задача 3.6** сме сигурни, че $c_5 > 0$ (и че $\max\{|-2|, |2|, |2|, |2|, |3|\} = 3 \in \{-2, 2, 2, 2, 3\}$).

Задача 3.7. Каква е асимптотиката на $T(n) = 2T(n-1) + T(n-2)$?

Решение. Тук имаме само хомогенна част: $x^2 = 2x+1 \ \mapsto \ x_{1,2} = 1 \pm \sqrt{2} \ \mapsto \ \{1+\sqrt{2}, 1-\sqrt{2}\}_M$. Оттук получаваме $T(n) = c_1(1+\sqrt{2})^n + c_2(1-\sqrt{2})^n = \Theta((1+\sqrt{2})^n)$.

Задача 3.8. Каква е асимптотиката на $T(n) = T(n-2) + T(n-4) + \cdots + \underbrace{T(n\%2)}_{T(0) \text{ или } T(1)}$?

Решение. Това характеристично уравнение се решава със следния трик: $T(n) - T(n-2) = \underbrace{T(n-2) + T(n-4) + \cdots + T(n\%2)}_{T(n)} - \underbrace{(T(n-4) + T(n-6) + \cdots + T(n\%2))}_{T(n-2)} = T(n-2)$. Тоест получихме, че $T(n) - T(n-2) = T(n-2)$ или още $T(n) = 2T(n-2)$. Решаваме характеристичното уравнение $x^2 = 2$ и получаваме $x_{1,2} = \pm\sqrt{2} \mapsto \{\sqrt{2}, -\sqrt{2}\}_M$. Оттук имаме $T(n) = c_1(\sqrt{2})^n + c_2(-\sqrt{2})^n = \Theta((\sqrt{2})^n)$.

3.2.2 Развиване

Метода за намиране асимптотика на рекурентно уравнение чрез развиване е най-мощният, който ще разгледаме в текущия курс, но и най-хамалският. Изисква сравнително много писане и известна доза *умен съм бил сетил съм се* (зависи от задачата). Самия метод не е формален. Формализацията изисква доказателство чрез индукция, което е трудната част. Идеята е много проста - развиваме докажо не заподозреем сложността и след това доказваме формално.

Задача 3.9. Каква е асимптотиката на $T(n) = T(n-1) + n$?

Решение. Тази задача може да се реши чрез характеристично уравнение, но сега ще го докажем чрез развиване и индукция.

$$\begin{aligned} T(n) &= T(n-1) + n = T(n-2) + (n-1) + n = \\ &= T(n-3) + (n-2) + (n-1) + n = \cdots = \\ &= T(0) + 1 + 2 + \cdots + (n-1) + n = T(0) + \Theta(n^2) = \Theta(n^2) \end{aligned}$$

Забележка

Дотук **НЕ** сме доказали нищо! Само сме заподозрели отговора! Проблема е в това, че употребата на "..." е изцяло неформална.. може да сме сбъркали нещо на ум!

Нека $n_{st} \in \mathbb{N}_0$ е достатъчно голямо[1].

- $T(n) = O(n^2)$

  По дефиниция имаме $O(n^2) = \{f \in \mathscr{F}^+ | (\exists c > 0)(\exists n_0 \in \mathbb{N}_0)(\forall n \geq n_0)(0 \leq f(n) \leq cn^2)\}$. Тоест търсим константи $c > 0$ и $n_0 \in \mathbb{N}_0$ такива, че $(\forall n \geq n_0)(0 \leq T(n) \leq cn^2)$. Както се разбрахме в началото на 3.2, сложността на рекурсивните алгоритмите, които ще разглеждаме в текущия курс, изпълняват условието $T(n) \geq 0$. Тоест търсим константи $c > 0$ и $n_0 \in \mathbb{N}_0$ такива, че $(\forall n \geq n_0)(T(n) \leq cn^2)$.

  Нека $b = \max\left\{\frac{T(1)}{1^2}, \frac{T(2)}{2^2}, \frac{T(3)}{3^2}, \ldots, \frac{T(n_{\text{st}}-1)}{(n_{\text{st}}-1)^2}, 1^2\right\}$. (виж **Забележка 3.2**)

  Ще докажем с индукция по n, че за $c = b$ и някое n_0 (което ще установим по-надолу) е изпълнено $(\forall n \geq n_0)(T(n) \leq cn^2)$.

[1] Надолу в доказателството ще въведем n_0 и ще изискваме $n_0 < n_{st}$.

[2] Надолу ще установим, защо добавяме тази единица.

База. Нека $k \in \{1, 2, \ldots, n_{st} - 1\}$. Ще докажем, че $T(k) \leq bk^2$.

От дефиницията на $b = \max\left\{\frac{T(1)}{1^2}, \frac{T(2)}{2^2}, \frac{T(3)}{3^2}, \ldots, \frac{T(n_{\text{st}}-1)}{(n_{\text{st}}-1)^2}, 1\right\}$ знаем, че $b \geq \frac{T(k)}{k^2}$ откъдето $bk^2 \geq T(k)$ или още $T(k) \leq bk^2$.

Индуктивна хипотеза. Нека допуснем, че е изпълнено $\big(\forall m < n\big)\big(T(m) \leq bm^2\big)$.

Индуктивна стъпка. Ще докажем, че е изпълнено за n, тоест че $T(n) \leq bn^2$.

$$T(n) \overset{\text{def}}{=} T(n-1) + n \overset{\text{ИХ}}{\leq} b(n-1)^2 + n = bn^2 - 2bn + b + n \overset{?}{\leq} bn^2$$

$$bn^2 - 2bn + b + n \overset{?}{\leq} bn^2$$

$$n(1 - 2b) + b \overset{?}{\leq} 0$$

Може да забележим, че при $1 - 2b \leq -b$ горното неравенство ще е изпълнено за всяко $n \geq 1$. Оттук си избираме $n_0 = 1$.

Остана да видим кога е изпълнено $1 - 2b \leq -b$. Очевидно е изпълнено когато $b \geq 1$. Това сме си го подсигурили от дефиницията на $b = \max\{\ldots, 1\} \Rightarrow b \geq 1$.

Тоест доказахме, че за $c = b \wedge n_0 = 1$ е изпълнено $\big(\forall n \geq n_0\big)\big(0 \leq T(n) \leq cn^2\big)$. Казано с други думи $T(n) = O(n^2)$.

- $T(n) = \Omega(n^2)$

  По дефиниция имаме $\Omega(n^2) = \{f \in \mathscr{F}^+ | \big(\exists c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq cn^2 \leq f(n)\big)\}$. Тоест търсим константи $c > 0$ и $n_0 \in \mathbb{N}_0$ такива, че $\big(\forall n \geq n_0\big)\big(cn^2 \leq T(n)\big)$.

  Нека $b = \min\left\{\frac{T(1)}{1^2}, \frac{T(2)}{2^2}, \frac{T(3)}{3^2}, \ldots, \frac{T(n_{\text{st}}-1)}{(n_{\text{st}}-1)^2}, \frac{1}{2}\right\}$. (виж **Забележка 3.2**)

  Ще докажем с индукция по n, че за $c = b$ и някое n_0 (което ще установим по-надолу) е изпълнено $\big(\forall n \geq n_0\big)\big(cn^2 \leq T(n)\big)$.

  База. Нека $k \in \{1, 2, \ldots, n_{st} - 1\}$. Ще докажем, че $bk^2 \leq T(k)$.

  От дефиницията на $b = \min\left\{\frac{T(1)}{1^2}, \frac{T(2)}{2^2}, \frac{T(3)}{3^2}, \ldots, \frac{T(n_{\text{st}}-1)}{(n_{\text{st}}-1)^2}, \frac{1}{2}\right\}$ знаем, че $b \leq \frac{T(k)}{k^2}$ откъдето $bk^2 \leq T(k)$.

  Индуктивна хипотеза. Нека допуснем, че е изпълнено $\big(\forall m < n\big)\big(bm^2 \leq T(m)\big)$.

  Индуктивна стъпка. Ще докажем, че е изпълнено за n, тоест че $bn^2 \leq T(n)$.

  $$T(n) \overset{\text{def}}{=} T(n-1) + n \overset{\text{ИХ}}{\geq} b(n-1)^2 + n = bn^2 - 2bn + b + n \overset{?}{\geq} bn^2$$

  $$n(1 - 2b) + b \overset{?}{\geq} 0$$

  Може да забележим, че при $1 - 2b \geq 0$ горното неравенство ще е изпълнено за всяко $n \geq 0$. Оттук си избираме $n_0 = 1$ (ако изберем $n_0 = 0$ ще имаме $\frac{T(0)}{0}$).

  Остана да видим кога е изпълнено $1 - 2b \geq 0$. Очевидно е изпълнено когато $b \leq \frac{1}{2}$. Това сме си го подсигурили от дефиницията на $b = \min\{\ldots, \frac{1}{2}\} \Rightarrow b \leq \frac{1}{2}$.

  Тоест доказахме, че за $c = b \wedge n_0 = 1$ е изпълнено $\big(\forall n \geq n_0\big)\big(0 \leq cn^2 \leq T(n)\big)$. Казано с други думи $T(n) = \Omega(n^2)$.

Забележка 3.2: Константата c при доказване с индукция

Когато доказваме, че асимптотиката на рекурентно уравнение е $\varphi(n)$ чрез индукция, то константата винаги е от вида max (за O) или min (за Ω) на $\{\frac{T(n_0)}{\varphi(n_0)}, \frac{T(n_0+1)}{\varphi(n_0+1)}, \ldots, \frac{T(n_{st}-1)}{\varphi(n_{st}-1)}\}$, като внимаваме за избора на n_0 - да не разделим на нула (примерно ако $\varphi(n) = log(n)$, то тогава $\frac{T(1)}{\varphi(1)} = \frac{T(1)}{0}$). Потенциално може да трябва още един елемент към множеството. Нека разгледаме няколко примерни константи:

- $T(n) = O(n^2) \mapsto \max\left\{\frac{T(1)}{1^2}, \frac{T(2)}{2^2}, \frac{T(3)}{3^2}, \ldots, \frac{T(n_{\text{st}}-1)}{(n_{\text{st}}-1)^2}, ?\right\}$
- $T(n) = \Omega(log(n)) \mapsto \min\left\{\frac{T(2)}{log(2)}, \frac{T(3)}{log(3)}, \ldots, \frac{T(n_{\text{st}}-1)}{log(n_{st}-1)}, ?\right\}$
- $T(n) = O(2^n) \mapsto \max\left\{\frac{T(0)}{2^0}, \frac{T(1)}{2^1}, \frac{T(2)}{2^2}, \ldots, \frac{T(n_{\text{st}}-1)}{2^{(n_{st}-1)}}, ?\right\}$
- $T(n) = \Omega(n^3-4n^2+3n) \mapsto \min\left\{\frac{T(4)}{4^3-4\cdot4^2+3\cdot4}, \frac{T(5)}{5^3-4\cdot5^2+3\cdot5}, \ldots, \frac{T(n_{\text{st}}-1)}{(n_{st}-1)^3-4(n_{st}-1)^2+3(n_{st}-1)}, ?\right\}$

Важно е да обърнем внимание, че при образуването на базата има разлика между $\Omega(n^3 - 4n^2 + 3n)$, $\Omega(n^3)$ и $\Omega(5n^3)$. Пример защо това е важно в O-посоката на **Задача 3.13**.

Задача 3.10. Каква е асимптотиката на $T(n) = T(n-1) + \frac{1}{n}$?

Решение. Нека да развием, докато не заподозреем асимптотиката.

$$\begin{aligned} T(n) &= T(n-1) + \frac{1}{n} = T(n-2) + \frac{1}{n-1} + \frac{1}{n} = \\ &= T(n-3) + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n} = \cdots = \\ &= T(0) + \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n-1} + \frac{1}{n} = T(0) + \Theta(ln(n)) = \Theta(log(n)) \end{aligned}$$

Нека $n_{st} \in \mathbb{N}_0$ е достатъчно голямо.

- $T(n) = O(log(n))$

  По дефиниция $O(log(n)) = \{f \in \mathscr{F}^+ | \big(\exists c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq f(n) \leq c\,log(n)\big)\}$. Тоест търсим $c > 0$ и $n_0 \in \mathbb{N}_0$ такива, че $\big(\forall n \geq n_0\big)\big(T(n) \leq c\,log(n)\big)$.

  Нека $b = \max\left\{\frac{T(2)}{log(2)}, \frac{T(3)}{log(3)}, \frac{T(4)}{log(4)}, \ldots, \frac{T(n_{\text{st}}-1)}{log(n_{\text{st}}-1)}, 1\right\}$.

  Ще докажем с индукция по n, че за $c = b$ и някое n_0 (което ще установим по-надолу) е изпълнено $\big(\forall n \geq n_0\big)\big(T(n) \leq c\,log(n)\big)$.

  База. Нека $k \in \{2, 3, \ldots, n_{st} - 1\}$. Ще докажем, че $T(k) \leq b\,log(k)$.

  От дефиницията на $b = \max\left\{\frac{T(2)}{log(2)}, \frac{T(3)}{log(3)}, \frac{T(4)}{log(4)}, \ldots, \frac{T(n_{\text{st}}-1)}{log(n_{\text{st}}-1)}, 1\right\}$ знаем, че $b \geq \frac{T(k)}{log(k)}$ откъдето $b\,log(k) \geq T(k)$ или още $T(k) \leq b\,log(k)$.

  Индуктивна хипотеза. Нека допуснем, че е изпълнено $\big(\forall m < n\big)\big(T(m) \leq b\,log(m)\big)$.

Индуктивна стъпка. Ще докажем, че е изпълнено за n, тоест че $T(n) \leq b\,log(n)$.

$$T(n) \overset{\text{def}}{=} T(n-1) + \frac{1}{n} \overset{\text{ИХ}}{\leq} b\,log(n-1) + \frac{1}{n} \overset{?}{\leq} b\,log(n)$$

$$b\,log(n-1) + \frac{1}{n} \overset{?}{\leq} b\,log(n)$$

Тъй като $n \geq n_0 \in \mathbb{N}_0$, то може да умножим по n без да се обърне знака на неравенството.

$$bn\,log(n-1) + 1 \overset{?}{\leq} bn\,log(n)$$

$$1 \overset{?}{\leq} bn\,log(n) - bn\,log(n-1)$$

$$1 \overset{?}{\leq} bn\,log\left(\frac{n}{n-1}\right)$$

Антилогаритмуваме двете страни (със основа 2)

$$2 \overset{?}{\leq} \left(\frac{n}{n-1}\right)^{bn}$$

От дефиницията на $b = \max\{\ldots, 1\} \Rightarrow b \geq 1$. Ясно е, че $\big(\forall n \geq 1\big)\big(\frac{n}{n-1} \geq 1\big)$. Тоест имаме:

$$2 \overset{?}{\leq} \left(\frac{n}{n-1}\right)^{n} \leq \left(\frac{n}{n-1}\right)^{bn}$$

$$2 \overset{?}{\leq} \left(\frac{n}{n-1}\right)^{n-1} \leq \left(\frac{n}{n-1}\right)^{n}$$

От анализа знаем, че $\lim\limits_{n\to\infty} \big(\frac{n}{n-1}\big)^{n-1} = e$ и че функцията $\big(\frac{n}{n-1}\big)^{n-1}$ е строго растяща. Тогава директно проверяваме, че $n_0 = 2$ ни върши работа

$$\big(\forall n \geq n_0\big)\left(2 \leq \left(\frac{n}{n-1}\right)^{n-1} \leq \left(\frac{n}{n-1}\right)^{bn}\right)$$

или още

$$\big(\forall n \geq n_0\big)\left(2 \leq \left(\frac{n}{n-1}\right)^{bn}\right)$$

Тъй като логаритмуването е монотонно растяща функция, то можем да запазим неравенството след логаритмуване на двете страни

$$\big(\forall n \geq n_0\big)\left(1 \leq bn\,log\left(\frac{n}{n-1}\right)\right)$$

което трябваше да докажем.

- $T(n) = \Omega(log(n))$

  По дефиниция $\Omega(log(n)) = \{f \in \mathscr{F}^+ | \big(\exists c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq c\,log(n)\big) \leq f(n)\}$. Тоест търсим $c > 0$ и $n_0 \in \mathbb{N}_0$ такива, че $\big(\forall n \geq n_0\big)\big(c\,log(n) \leq T(n)\big)$.

  Нека $b = \min\left\{\frac{T(2)}{log(2)}, \frac{T(3)}{log(3)}, \frac{T(4)}{log(4)}, \ldots, \frac{T(n_{\text{st}}-1)}{log(n_{\text{st}}-1)}, \frac{1}{2}\right\}$.

  Ще докажем с индукция по n, че за $c = b$ и някое n_0 (което ще установим по-надолу) е изпълнено $\big(\forall n \geq n_0\big)\big(c\,log(n) \leq T(n)\big)$.

База. Нека $k \in \{2, 3, \ldots, n_{st} - 1\}$. Ще докажем, че $b\,log(k) \leq T(k)$.

От дефиницията на $b = \min\left\{\frac{T(2)}{log(2)}, \frac{T(3)}{log(3)}, \frac{T(4)}{log(4)}, \ldots, \frac{T(n_{\text{st}}-1)}{log(n_{\text{st}}-1)}, \frac{1}{2}\right\}$ знаем, че $b \leq \frac{T(k)}{log(k)}$ откъдето $b\,log(k) \leq T(k)$.

Индуктивна хипотеза. Нека допуснем, че е изпълнено $\big(\forall m < n\big)\big(b\,log(m) \leq T(m)\big)$.

Индуктивна стъпка. Ще докажем, че е изпълнено за n, тоест че $b\,log(n) \leq T(n)$.

$$T(n) \overset{\text{def}}{=} T(n-1) + \frac{1}{n} \overset{\text{ИХ}}{\geq} b\,log(n-1) + \frac{1}{n} \overset{?}{\geq} b\,log(n)$$

Аналогично на O-посоката стигаме до

$$2 \overset{?}{\geq} \left(\frac{n}{n-1}\right)^{bn}$$

От дефиницията на $b = \min\{\ldots, \frac{1}{2}\} \Rightarrow b \leq \frac{1}{2}$. Ясно е, че $\big(\forall n \geq 1\big)\big(\frac{n}{n-1} \geq 1\big)$. Тоест имаме:

$$2 \overset{?}{\geq} \left(\frac{n}{n-1}\right)^{n/2} \geq \left(\frac{n}{n-1}\right)^{bn}$$

Вдигаме на квадрат двете страни

$$4 \overset{?}{\geq} \left(\frac{n}{n-1}\right)^{n} = \left(\frac{n}{n-1}\right)\left(\frac{n}{n-1}\right)^{n-1}$$

От анализа имаме

$$4 \overset{?}{\geq} \left(\frac{n}{n-1}\right)e \geq \left(\frac{n}{n-1}\right)\left(\frac{n}{n-1}\right)^{n-1}$$

$$4(n-1) \overset{?}{\geq} ne$$

$$n(4-e) \overset{?}{\geq} 4$$

Директно проверяваме, че $n_0 = 4$ ни върши работа

$$\big(\forall n \geq n_0\big)\big(n(4-e) \geq 4\big)$$

$$\big(\forall n \geq n_0\big)\left(4 \geq \left(\frac{n}{n-1}\right)e \geq \left(\frac{n}{n-1}\right)^{n}\right)$$

Тъй като коренуването е монотонно растяща функция, то можем да запазим неравенството след коренуване на двете страни

$$\big(\forall n \geq n_0\big)\left(2 \geq \left(\frac{n}{n-1}\right)^{n/2} \geq \left(\frac{n}{n-1}\right)^{bn}\right)$$

Тъй като логаритмуването е монотонно растяща функция, то можем да запазим неравенството след логаритмуване на двете страни

$$\big(\forall n \geq n_0\big)\left(1 \geq bn\,log\left(\frac{n}{n-1}\right)\right)$$

което трябваше да докажем.

Задача 3.11. Каква е асимптотиката на $T(n) = 2T(n-1) + \frac{1}{n}$?

Решение. Нека да развием, докато не заподозреем асимптотиката.

$$\begin{aligned} T(n) &= 2T(n-1) + \frac{1}{n} = 4T(n-2) + \frac{2}{n-1} + \frac{1}{n} = \\ &= 8T(n-3) + \frac{4}{n-2} + \frac{2}{n-1} + \frac{1}{n} = \cdots = \\ &= 2^n T(0) + \frac{2^{n-1}}{1} + \frac{2^{n-2}}{2} + \cdots + \frac{2}{n-1} + \frac{1}{n} = \\ &= 2^n \Theta(1) + \underbrace{2^n \sum_{i=1}^{n} \frac{1}{i2^i}}_{O(2^n)} = \Theta(2^n) \end{aligned}$$

Докажете го формално чрез индукция.

Задача 3.12. Каква е асимптотиката на $T(n) = \frac{n}{n+1}T(n-1) + 1$?

Решение. Нека да развием, докато не заподозреем асимптотиката.

$$\begin{aligned} T(n) &= \frac{n}{n+1}T(n-1) + 1 = \frac{n-1}{n+1}T(n-2) + \frac{n}{n+1} + \frac{n+1}{n+1} = \\ &= \frac{n-2}{n+1}T(n-3) + \frac{n-1}{n+1} + \frac{n}{n+1} + \frac{n+1}{n+1} = \cdots = \\ &= \frac{1}{n+1}T(0) + \frac{2}{n+1} + \frac{3}{n+1} + \cdots + \frac{n}{n+1} + \frac{n+1}{n+1} = \\ &= \Theta(n^{-1}) + \frac{1}{n+1}\underbrace{(2 + 3 + \cdots + (n+1))}_{\Theta(n^2)} = \Theta(n) \end{aligned}$$

Докажете го формално чрез индукция.

3.2.2.1 Засилване на индуктивната хипотеза

В някои ситуации доказването на индуктивната стъпка би било невъзможно без така наречената ***засилване на индуктивната хипотеза***. Какво представлява то се разбира най-добре с пример.

Задача 3.13. Каква е асимптотиката на $T(n) = 2T(n-1) + T(log(n)) + n$?

Решение. Заподозряваме, че асимптотиката е $T(n) = \Theta(2^n)$.
Нека $n_{st} \in \mathbb{N}_0$ е достатъчно голямо.

- $T(n) = O(2^n)$

  По дефиниция имаме $O(2^n) = \{f \in \mathscr{F}^+ | \big(\exists c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq f(n) \leq c2^n\big)\}$.
  Тоест търсим константи $c > 0$ и $n_0 \in \mathbb{N}_0$ такива, че $\big(\forall n \geq n_0\big)\big(T(n) \leq c2^n\big)$.

Нека $b = \max\left\{\frac{T(0)}{2^0}, \frac{T(1)}{2^1}, \frac{T(2)}{2^2}, \ldots, \frac{T(n_{\text{st}}-1)}{2^{n_{\text{st}}-1}}\right\}$.

Ще (се опитаме да) докажем с индукция по n, че за $c = b$ и някое n_0 (което ще установим по-надолу) е изпълнено $\big(\forall n \geq n_0\big)\big(T(n) \leq c2^n\big)$.

База. Нека $k \in \{0, 1, \ldots, n_{st} - 1\}$. Ще докажем, че $T(k) \leq b2^k$.

От дефиницията на $b = \max\left\{\frac{T(0)}{2^0}, \frac{T(1)}{2^1}, \frac{T(2)}{2^2}, \ldots, \frac{T(n_{\text{st}}-1)}{2^{n_{\text{st}}-1}}\right\}$ знаем, че $b \geq \frac{T(k)}{2^k}$ откъдето $b2^k \geq T(k)$ или още $T(k) \leq b2^k$.

Индуктивна хипотеза. Нека допуснем, че е изпълнено $\big(\forall m < n\big)\big(T(m) \leq b2^m\big)$.

Индуктивна стъпка. Ще докажем, че е изпълнено за n, тоест че $T(n) \leq b2^n$.

$$T(n) \overset{\text{def}}{=} 2T(n-1) + T(log(n)) + n \overset{\text{ИХ}}{\leq} 2b2^{n-1} + b2^{log(n)} + n = b2^n + bn + n \overset{?}{\leq} b2^n$$

$$b2^n + bn + n \overset{?}{\leq} b2^n$$

$$bn + n \overset{?}{\leq} 0$$

Очевидно не е изпълнено за никои $b > 0$ и $n > 0$. Не стана.. ще трябва да засилим индуктивната хипотеза.

- $T(n) = O(2^n - \alpha^n)$, $\alpha \in (1, 2)$

  По дефиниция $O(2^n - \alpha^n) = \{f \in \mathscr{F}^+ | \big(\exists c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq f(n) \leq c(2^n - \alpha^n)\big)\}$. Тоест търсим константи $c > 0$ и $n_0 \in \mathbb{N}_0$ такива, че $\big(\forall n \geq n_0\big)\big(T(n) \leq c(2^n - \alpha^n)\big)$.

  Нека $b = \max\left\{\frac{T(1)}{2^1 - \alpha^1}, \frac{T(2)}{2^2 - \alpha^2}, \ldots, \frac{T(n_{\text{st}}-1)}{2^{n_{\text{st}}-1} - \alpha^{n_{\text{st}}-1}}, 1\right\}$. (при нула знаменателя е $2^0 - \alpha^0 = 0$)

  Ще докажем с индукция по n, че за $c = b$ и някое n_0 (което ще установим по-надолу) е изпълнено $\big(\forall n \geq n_0\big)\big(T(n) \leq c(2^n - \alpha^n)\big)$.

  База. Нека $k \in \{1, 2, \ldots, n_{st} - 1\}$. Ще докажем, че $T(k) \leq b(2^k - \alpha^k)$.

  От дефиницията на $b = \max\left\{\frac{T(1)}{2^1 - \alpha^1}, \frac{T(2)}{2^2 - \alpha^2}, \ldots, \frac{T(n_{\text{st}}-1)}{2^{n_{\text{st}}-1} - \alpha^{n_{\text{st}}-1}}, 1\right\}$ знаем, че $b \geq \frac{T(k)}{2^k - \alpha^k}$ откъдето $b(2^k - \alpha^k) \geq T(k)$ или още $T(k) \leq b(2^k - \alpha^k)$.

  Индуктивна хипотеза. Нека допуснем, че е изпълнено $\big(\forall m < n\big)\big(T(m) \leq b(2^m - \alpha^m)\big)$.

  Индуктивна стъпка. Ще докажем, че е изпълнено за n, тоест че $T(n) \leq b(2^n - \alpha^n)$.

$$T(n) \overset{\text{def}}{=} 2T(n-1) + T(log(n)) + n \overset{\text{ИХ}}{\leq} 2b(2^{n-1} - \alpha^{n-1}) + b(2^{log(n)} - \alpha^{log(n)}) + n \overset{?}{\leq} b(2^n - \alpha^n)$$

$$2b(2^{n-1} - \alpha^{n-1}) + b(2^{log(n)} - \alpha^{log(n)}) + n \overset{?}{\leq} b(2^n - \alpha^n)$$

$$b2^n - 2b\alpha^{n-1} + bn - bn^{log(\alpha)} + n \overset{?}{\leq} b2^n - b\alpha^n$$

$$b(\alpha - 2)\alpha^{n-1} + bn - bn^{log(\alpha)} + n \overset{?}{\leq} 0 \tag{3.3}$$

Тъй като $\alpha \in (1,2)$, то $log(\alpha) = log_2(\alpha) \in (0,1)$. Тогава имаме

$$b(\underbrace{\alpha - 2}_{<0})\alpha^{n-1} + \underbrace{bn - bn^{log(\alpha)} + n}_{\Theta(n)} \overset{?}{\leq} 0$$

Ясно е, че асимптотиката се определя от α^{n-1} и че за кое да е $b > 0$ има подходящо n_0. Ако искаме да посочим конкретно n_0 трябва да фиксираме α. Ще го направим за пълнота. Нека $\alpha = \sqrt{2} \in (1,2)$ и заместим в неравенство (3.3)

$$b(\sqrt{2} - 2)(\sqrt{2})^{n-1} + bn - b\sqrt{n} + n \overset{?}{\leq} 0$$

От дефиницията на $b = max\{\ldots, 1\} \Rightarrow b \geq 1$

$$b(\sqrt{2} - 2)(\sqrt{2})^{n-1} + bn - b\sqrt{n} + n \leq b(\sqrt{2} - 2)(\sqrt{2})^{n-1} + bn - b\sqrt{n} + bn \overset{?}{\leq} 0$$

Сега от това, че $b > 0$ можем да разделим двете страни на b

$$(\sqrt{2} - 2)(\sqrt{2})^{n-1} + n - \sqrt{n} + n \overset{?}{\leq} 0$$

Ясно е, че $\big(\forall n > 1\big)\big(n - \sqrt{n} \leq n\big)$

$$(\sqrt{2} - 2)(\sqrt{2})^{n-1} + n - \sqrt{n} + n \leq (\sqrt{2} - 2)(\sqrt{2})^{n-1} + 2n \overset{?}{\leq} 0$$

$$(\sqrt{2} - 2)(\sqrt{2})^{n-1} + 2n \overset{?}{\leq} 0$$

Вече е достатъчно просто да забележим, че $n_0 = 12$ ни върши работа

$$\big(\forall n \geq n_0\big)\big((\sqrt{2} - 2)(\sqrt{2})^{n-1} + 2n \leq 0\big)$$

$$\big(\forall n \geq n_0\big)\big(b(\sqrt{2} - 2)(\sqrt{2})^{n-1} + bn - b\sqrt{n} + n \leq 0\big)$$

което трябваше да докажем.

Тоест доказахме, че $T(n) = O(2^n - (\sqrt{2})^n)$, т.е. $T(n) = O(2^n)$.

- $T(n) = \Omega(2^n)$

  По дефиниция имаме $\Omega(2^n) = \{f \in \mathscr{F}^+ | \big(\exists c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq c2^n \leq f(n)\big)\}$. Тоест търсим константи $c > 0$ и $n_0 \in \mathbb{N}_0$ такива, че $\big(\forall n \geq n_0\big)\big(c2^n \leq T(n)\big)$.

  Нека $b = \max\left\{\frac{T(0)}{2^0}, \frac{T(1)}{2^1}, \frac{T(2)}{2^2}, \ldots, \frac{T(n_{\text{st}}-1)}{2^{n_{\text{st}}-1}}\right\}$.

  Ще докажем с индукция по n, че за $c = b$ и някое n_0 (което ще установим по-надолу) е изпълнено $\big(\forall n \geq n_0\big)\big(c2^n \leq T(n)\big)$.

  База. Нека $k \in \{0, 1, \ldots, n_{st} - 1\}$. Ще докажем, че $b2^k \leq T(k)$.

  От дефиницията на $b = \min\left\{\frac{T(0)}{2^0}, \frac{T(1)}{2^1}, \frac{T(2)}{2^2}, \ldots, \frac{T(n_{\text{st}}-1)}{2^{n_{\text{st}}-1}}\right\}$ знаем, че $b \leq \frac{T(k)}{2^k}$ откъдето $b2^k \leq T(k)$.

  Индуктивна хипотеза. Нека допуснем, че е изпълнено $\big(\forall m < n\big)\big(b2^m \leq T(m)\big)$.

Индуктивна стъпка. Ще докажем, че е изпълнено за n, тоест че $T(n) \leq b2^n$.

$$T(n) \overset{\text{def}}{=} 2T(n-1) + T(log(n)) + n \overset{\text{ИХ}}{\geq} 2b2^{n-1} + b2^{log(n)} + n = b2^n + bn + n \overset{?}{\geq} b2^n$$

$$b2^n + bn + n \overset{?}{\geq} b2^n$$

$$bn + n \overset{?}{\geq} 0$$

Очевидно е изпълнено за кое да е $b > 0$ и $n \geq 0$. Оттук си избираме $n_0 = 0$.

Забележка. Обърнете внимание, че този път не добавихме бонус стойност в множеството като образувахме b.. не е нужно винаги!

3.2.3 Полагане

Този метод използваме в комбинация с други методи. Първо полагаме и свеждаме задачата до по-лесна, след което прилагаме друг метод за да намерим асимптотиката.

Задача 3.14. Каква е асимптотиката на $T(n) = 2T(\sqrt{n}) + 1$?

Решение. Полагаме $n = 2^{2^m}$, т.е. $m = log(log(n))$. Нека $S(m) = T(2^{2^m}) = T(n)$. Тогава

$$S(m) = T(2^{2^m}) = 2T(2^{\frac{2^m}{2}}) + 1 = 2T(2^{2^{m-1}}) + 1 = 2S(m-1) + 1$$

$$S(m) = 2S(m-1) + 1$$

Сега използвайки някой от предните методи (а може и някой от бъдещите) доказваме, че $S(m) = \Theta(2^m)$. Тоест $T(n) = S(m) = \Theta(2^m) = \Theta(2^{log(log(n))}) = \Theta(log(n))$.

Задача 3.15. Каква е асимптотиката на $T(n) = T(\sqrt{n}) + 1$?

Решение. Полагаме $n = 2^{2^m}$, т.е. $m = log(log(n))$. Нека $S(m) = T(2^{2^m}) = T(n)$. Тогава

$$S(m) = T(2^{2^m}) = T(2^{\frac{2^m}{2}}) + 1 = T(2^{2^{m-1}}) + 1 = S(m-1) + 1$$

$$S(m) = S(m-1) + 1$$

Сега използвайки някой от предните методи (а може и някой от бъдещите) доказваме, че $S(m) = \Theta(m)$. Тоест $T(n) = S(m) = \Theta(m) = \Theta(log(log(n)))$.

Допълнение 3.1: Рекурсивно извикване с корен от аргумента

Нека разгледаме $T(n) = T(\sqrt{n}) + 1$. То извършва константна работа при всяко извикване и извиква $T(\sqrt{n})$. Тоест се интересуваме колко на брой коренувания ще направим преди n да стане 1. Нека вземем едно произволно число - да кажем

30000. Нека сега разгледаме какво се случва когато го коренуваме многократно: $30000 \mapsto 173 \mapsto 13 \mapsto 3 \mapsto 1$. Нека сега запишем числата в двоична бройна система: $111010100110000_{(2)} \mapsto 10101100_{(2)} \mapsto 1101_{(2)} \mapsto 11_{(2)} \mapsto 1_{(2)}$ и да разгледаме тяхната дължина: $15 \mapsto 8 \mapsto 4 \mapsto 2 \mapsto 1$. Може да забележим, че коренуването на дадено число де факто намалява двойно (закръглено нагоре) дължината му в двоична бройна система. Тоест правим $log(\textit{дължината на } n \textit{ в двоична бройна система}) = log(log(n))$ брой стъпки.

3.2.4 Мастър теорема

Макар и силна, не трябва да се подлъгваме по гръмкото име на тази теорема - тя **не** може да намира асимптотиката на какви да е рекурентни уравнения! Въпреки това е много силно пособие, което ще използваме в текущия курс.

Теорема 3.2: Мастър теорема

Нека $a \geq 1, b > 1, f(n)$ - положителна и положим $k = log_b(a)$. Тогава за рекурентното уравнение $T(n) = aT(\frac{n}{b}) + f(n)$ имаме:

$$T(n) = \begin{cases} \Theta(n^k) & , \big(\exists \varepsilon > 0\big)\big(f(n) = O(n^{k-\varepsilon})\big) \\ \Theta(n^k log(n)) & , f(n) = \Theta(n^k) \\ \Theta(f(n)) & , \big(\exists \varepsilon > 0\big)\big(f(n) = \Omega(n^{k+\varepsilon})\big) \wedge \\ & \underbrace{\big(\exists c \in (0,1)\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(af(\frac{n}{b}) \leq cf(n)\big)}_{\text{условие за регулярност}} \\ \textit{не знаем} & , \text{иначе} \end{cases}$$

Задача 3.16. Каква е асимптотиката на $T(n) = 4T(\frac{n}{2}) + n$?

Решение. Ще използваме мастър теоремата (случай 1):

$$\left| \begin{array}{l} a = 4 \\ b = 2 \\ f(n) = n \\ k = log_b(a) = log_2(4) = 2 \end{array} \right.$$

Нека $\varepsilon = \frac{1}{10}$. Тогава имаме $f(n) = n = O(n^{2-0,1}) = O(n^{1,9}) \overset{MT1}{\Longrightarrow} T(n) = \Theta(n^2)$.

Задача 3.17. Каква е асимптотиката на $T(n) = 4T(\frac{n}{\sqrt{2}}) + n^3$?

Решение. Ще използваме мастър теоремата (случай 1):

$$\left|\begin{array}{l} a = 4 \\ b = \sqrt{2} \\ f(n) = n^3 \\ k = log_b(a) = log_{\sqrt{2}}(4) = 4 \end{array}\right.$$

Нека $\varepsilon = \frac{1}{10}$. Тогава имаме $f(n) = n = O(n^{4-0,1}) = O(n^{3,9}) \overset{MT\,1}{\Longrightarrow} T(n) = \Theta(n^4)$.

Задача 3.18. Каква е асимптотиката на $T(n) = T(\frac{n}{2}) + 1$?

Решение. Ще използваме мастър теоремата (случай 2):

$$\left|\begin{array}{l} a = 1 \\ b = 2 \\ f(n) = 1 \\ k = log_b(a) = log_2(1) = 0 \end{array}\right.$$

Имаме $f(n) = 1 = \Theta(1) = \Theta(n^0) = \Theta(n^k) \overset{MT\,2}{\Longrightarrow} T(n) = \Theta(n^k log(n)) = \Theta(log(n))$.

Задача 3.19. Каква е асимптотиката на $T(n) = 2T(\frac{n}{8}) + n$?

Решение. Ще използваме мастър теоремата (случай 3):

$$\left|\begin{array}{l} a = 2 \\ b = 8 \\ f(n) = n \\ k = log_b(a) = log_8(2) = \frac{1}{3} \end{array}\right.$$

Нека $\varepsilon = \frac{1}{6}$. Тогава имаме $f(n) = n = \Omega(n^{\frac{1}{3}+\frac{1}{6}}) = \Omega(\sqrt{n})$. Остана да проверим условието за регулятност: $\big(\exists c \in (0,1)\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(af(\frac{n}{b}) \leq cf(n)\big)$. Тоест търсим $c \in (0,1)$ и $n_0 \in \mathbb{N}_0$:

$$\big(\forall n \geq n_0\big)\left(2f\left(\frac{n}{8}\right) \leq cf(n)\right)$$

$$\big(\forall n \geq n_0\big)\left(2\frac{n}{8} \leq cn\right)$$

Ясно е, че $c = \frac{1}{4}$ и $n_0 = 0$ ни вършат работа. Тоест от МТ 3 имаме, че $T(n) = \Theta(f(n)) = \Theta(n)$.

Задача 3.20. Каква е асимптотиката на $T(n) = 4T(\frac{n}{2}) + n^2\sqrt{n}$?

Решение. Ще използваме мастър теоремата (случай 3):

$$\left|\begin{array}{l} a = 4 \\ b = 2 \\ f(n) = n^{\frac{5}{2}} \\ k = log_b(a) = log_2(4) = 2 \end{array}\right.$$

Нека $\varepsilon = \frac{1}{10}$. Тогава имаме $f(n) = n^{2,5} = \Omega(n^{2+0,1}) = \Omega(n^{2,1})$. Остана да проверим условието за регулятност: $\big(\exists c \in (0,1)\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(af(\frac{n}{b}) \leq cf(n)\big)$. Тоест търсим $c \in (0,1)$ и $n_0 \in \mathbb{N}_0$:

$$\big(\forall n \geq n_0\big)\left(4f\left(\frac{n}{2}\right) \leq cf(n)\right)$$

$$\big(\forall n \geq n_0\big)\left(4\frac{n^2\sqrt{n}}{4\sqrt{2}} \leq cn^2\sqrt{n}\right)$$

Ясно е, че $c = \frac{1}{\sqrt{2}}$ и $n_0 = 0$ ни вършат работа. Тоест от МТ 3 имаме, че $T(n) = \Theta(n^2\sqrt{n})$.

3.2.4.1 Разширение на Мастър теоремата

Макар и да обхваща по-голям набор от функции, тази теорема все още е твърде ограничена. Въпреки това може да улесни решаването на някои задачи многократно.

Теорема 3.3: Мастър теорема (разширена)

Нека $a \geq 1, b > 1, f(n)$ - положителна и положим $k = log_b(a)$. Тогава за рекурентното уравнение $T(n) = aT(\frac{n}{b}) + f(n)$ имаме:

$$T(n) = \begin{cases} \Theta(n^k) & , \big(\exists \varepsilon > 0\big)\big(f(n) = O(n^{k-\varepsilon})\big) \\ \Theta(n^k) & , \big(\exists \alpha < -1\big)\big(f(n) = \Theta(n^k log^{\alpha}(n))\big) \wedge \\ \Theta(n^k log^{(2)}(n)) & , \big(\exists \alpha = -1\big)\big(f(n) = \Theta(n^k log^{\alpha}(n))\big) \wedge \\ \Theta(n^k log^{\alpha+1}(n)) & , \big(\exists \alpha > -1\big)\big(f(n) = \Theta(n^k log^{\alpha}(n))\big) \wedge \\ \Theta(f(n)) & , \big(\exists \varepsilon > 0\big)\big(f(n) = \Omega(n^{k+\varepsilon})\big) \wedge \\ & \underbrace{\big(\exists c \in (0,1)\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(af(\frac{n}{b}) \leq cf(n)\big)}_{\text{условие за регулярност}} \\ \textit{не знаем} & , \text{иначе} \end{cases}$$

където $\alpha \in \mathbb{R}$.

Задача 3.21. Каква е асимптотиката на $T(n) = T(\frac{n}{2}) + \frac{1}{log(n)}$?

Решение. Ще използваме разширената мастър теорема (случай 3):

$$\begin{vmatrix} a = 1 \\ b = 2 \\ f(n) = log^{-1}(n) \\ k = log_b(a) = log_2(1) = 0 \end{vmatrix}$$

Имаме $f(n) = log^{-1}(n) = \Theta(n^0 log^{-1}(n)) \overset{EMT3}{\Longrightarrow} T(n) = \Theta(log(log(n)))$.

Задача 3.22. Каква е асимптотиката на $T(n) = 2T(\frac{n}{4}) + 2\sqrt{n}\,log^3(n)$?

Решение. Ще използваме разширената мастър теорема (случай 4):

$$\begin{vmatrix} a = 2 \\ b = 4 \\ f(n) = 2\sqrt{n}\,log^3(n) \\ k = log_b(a) = log_4(2) = \frac{1}{2} \end{vmatrix}$$

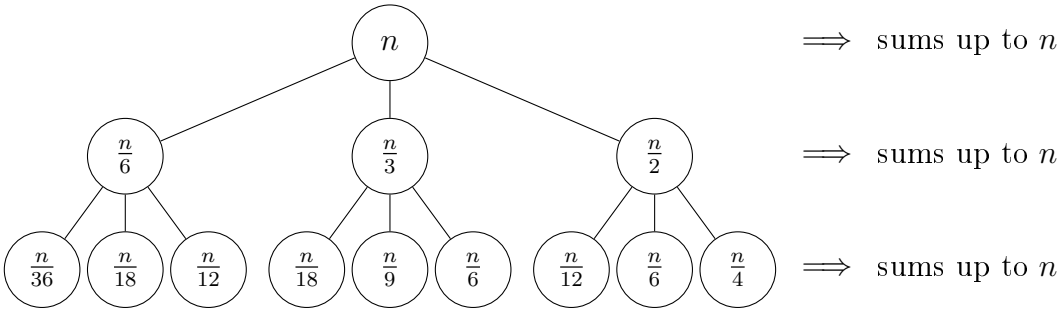
Имаме $f(n) = 2\sqrt{n}\,log^3(n) = \Theta(\sqrt{n}\,log^3(n)) \overset{EMT\,4}{\Longrightarrow} T(n) = \Theta(\sqrt{n}\,log^4(n))$.

3.2.5 Дърво на рекурсия

Аналогично на метода с развиване, текущия метод не е формален. Той служи само за придобиване на интуиция. Необходимо е формално доказателство с индукция след като се заподозрее асимптотиката.

Задача 3.23. Каква е асимптотиката на $T(n) = T\left(\frac{n}{6}\right) + T\left(\frac{n}{3}\right) + T\left(\frac{n}{2}\right) + n$?

Решение. Нека разгледаме дървото на рекурсия на $T(n)$:



Може да забележим, че на всяко ниво сбора на нехомогенните части дава точно n. Освен това дървото е пълно до ниво $\lceil log_6(n) \rceil$[3] (при база $n = 1$), съответно дървото е с височина $\lceil log_2(n) \rceil$[3]. Тоест заподозряхме, че $T(n) = \Theta(n\,log_2(n)) + O\big(n(log_6(n) - log_2(n))\big) = \Theta(n\,log(n))$. Сега ще го докажем формално. Нека $n_{st} \in \mathbb{N}_0$ е достатъчно голямо.

- $T(n) = O(n\,log(n))$

  По дефиниция $O(n\,log(n)) = \{f \in \mathscr{F}^+ | \big(\exists c > 0\big)\big(\exists n_0 \in \mathbb{N}_0\big)\big(\forall n \geq n_0\big)\big(0 \leq f(n) \leq cn\,log(n)\big)\}$. Тоест търсим $c > 0$ и $n_0 \in \mathbb{N}_0$ такива, че $\big(\forall n \geq n_0\big)\big(T(n) \leq n\,log(n)\big)$.

  Нека $b = \max\left\{\frac{T(2)}{2log(2)}, \frac{T(3)}{3log(3)}, \ldots, \frac{T(n_{\text{st}}-1)}{(n_{\text{st}}-1)log(n_{\text{st}}-1)}, 1\right\}$. (виж **Забележка 3.2**)

  Ще докажем с индукция по n, че за $c = b$ и някое n_0 (което ще установим по-надолу) е изпълнено $\big(\forall n \geq n_0\big)\big(T(n) \leq n\,log(n)\big)$.

  База. Нека $k \in \{2, 3, \ldots, n_{st} - 1\}$. Ще докажем, че $T(k) \leq b\,k\,log(k)$.

  От дефиницията на $b = \max\left\{\frac{T(2)}{2log(2)}, \frac{T(3)}{3log(3)}, \ldots, \frac{T(n_{\text{st}}-1)}{(n_{\text{st}}-1)log(n_{\text{st}}-1)}, 1\right\}$ знаем, че $b \geq \frac{T(k)}{k\,log(k)}$ откъдето $b\,k\,log(k) \geq T(k)$ или още $T(k) \leq b\,k\,log(k)$.

[3] Може да докажем тези две твърдения с индукция и да докажем формално сумата от всички върхове, че е в интервала $[n\,log_6(n), n\,log_2(n)]$, откъдето да получим асимптотиката $\Theta(n\,log(n))$. В текущия курс няма да правим индукции от такъв характер.

Индуктивна хипотеза. Нека допуснем, че е изпълнено $\big(\forall m < n\big)\big(T(m) \leq b\,m\,log(m)\big)$.

Индуктивна стъпка. Ще докажем, че е изпълнено за n, тоест че $T(n) \leq bn\,log(n)$.

$$T(n) \overset{\text{def}}{=} T\Big(\frac{n}{6}\Big) + T\Big(\frac{n}{3}\Big) + T\Big(\frac{n}{2}\Big) + n \overset{\text{ИХ}}{\leq} \frac{bn}{6}log\Big(\frac{n}{6}\Big) + \frac{bn}{3}log\Big(\frac{n}{3}\Big) + \frac{bn}{2}log\Big(\frac{n}{2}\Big) + n \overset{?}{\leq} bn\,log(n)$$

$$\frac{bn}{6}\big(log(n) - log(6)\big) + \frac{bn}{3}\big(log(n) - log(3)\big) + \frac{bn}{2}\big(log(n) - log(2)\big) + n \overset{?}{\leq} bn\,log(n)$$

$$bn\,log(n) + \left(1 - \frac{b\,log(6)}{6} - \frac{b\,log(3)}{3} - \frac{b\,log(2)}{2}\right)n \overset{?}{\leq} bn\,log(n)$$

$$\left(1 - \frac{b\,log(6)}{6} - \frac{b\,log(3)}{3} - \frac{b\,log(2)}{2}\right)n \overset{?}{\leq} 0$$

Може да забележим, че при $\frac{b\,log(6)}{6} + \frac{b\,log(3)}{3} + \frac{b\,log(2)}{2} \geq 1$ горното неравенство ще е изпълнено за всяко $n \geq 1$. Оттук си избираме $n_0 = 2$ (при $n_0 = 1$ имаме $\frac{T(1)}{1log(1)} = \frac{T(1)}{0}$).

Остана да видим кога е изпълнено $\frac{b\,log(6)}{6} + \frac{b\,log(3)}{3} + \frac{b\,log(2)}{2} \geq 1$. Очевидно е изпълнено когато $b \geq 1$. Това сме си го подсигурили от дефиницията на $b = \max\{\ldots, 1\} \Rightarrow b \geq 1$.

Тоест доказахме, че за $c = b$ и $n_0 = 1$ е изпълнено $\big(\forall n \geq n_0\big)\big(0 \leq T(n) \leq cn\,log(n)\big)$. Казано с други думи $T(n) = O(n\,log(n))$.

- $T(n) = \Omega(n\,log(n))$ - Докажете за упражнение.

3.2.6 Теорема на Akra-Bazzi

Сега ще разгледаме метод, който е строго по-силен от Мастър теоремата, която е твърде ограничена - има една единствена поява вдясно. Теоремата на Akra-Bazzi позволява решаването на частен случай рекурентни уравнения с една или повече появи вдясно.

Теорема 3.4: Akra-Bazzi

Нека $T(n) = \begin{cases} \theta(1) & , 1 \leq n \leq n_0 \\ a_1T(b_1n) + \cdots + a_kT(b_kn) + f(n) & , n > n_0 \end{cases}$, където

1. $1 \leq n \in \mathbb{R}$
2. $\big(\forall i \in \{1, \ldots, k\}\big)\big(n_0 \geq \frac{1}{b_i} \wedge n_0 \geq \frac{1}{1-b_i}\big)$
3. $\big(\forall i \in \{1, \ldots, k\}\big)\big(a_i > 0\big)$
4. $\big(\forall i \in \{1, \ldots, k\}\big)\big(b_i \in (0, 1)\big)$
5. $k \in \mathbb{N}^+$
6. $f(n)$ е неотрицателна функция, удволетворяваща условието за полиномиално нарастване
7. p е уникално число, за което $\sum_{i=1}^{k} a_i b_i^p = 1$

Тогава

$$T(n) \asymp n^p\left(1 + \int_1^n \frac{f(t)}{t^{p+1}}dt\right)$$

Дефиниция 3.1: Условие за полиномиално нарастване (в контекста на Теорема 3.4)

Ще казваме, че $f(n)$ удволетворява условието за полиномиално нарастване, т.с.т.к. $\big(\exists c_1 > 0\big)\big(\exists c_2 > 0\big)\big(\forall n \geq 1\big)\big(\forall i \in \{1, \ldots, k\}\big)\big(\forall t \in [b_i n, n]\big)\big(c_1 f(n) \leq f(t) \leq c_2 f(n)\big)$.

Задача 3.24. Каква е асимптотиката на $T(n) = \frac{1}{4}T\big(\frac{n}{4}\big) + \frac{3}{4}T\big(\frac{3n}{4}\big) + 1$?

Решение. Ще покажем, че седемте условия от теоремата на Akra-Bazzi са изпълнени:

1. Това условие няма какво да го проверяваме.
2. Проверяваме, че $n_0 = 4$ ни върши работа: $\big(\forall i \in \{1,2\}\big)\Big(4 \geq \frac{1}{b_i} \wedge 4 \geq \frac{1}{1-b_i}\Big)$.
3. Директно проверяваме $\big(\forall i \in \{1,2\}\big)\big(a_i > 0\big)$.
4. Директно проверяваме $\big(\forall i \in \{1,2\}\big)\big(b_i \in (0,1)\big)$.
5. Това условие няма какво да го проверяваме.
6. Проверяваме, че $c_1 = c_2 = 1$ са свидетели: $\big(\forall n \geq 1\big)\big(\forall i \in \{1,2\}\big)\big(\forall t \in [b_i n, n]\big)\big(1 \leq 1 \leq 1\big)$.
7. Проверяваме, че $p = 0$ ни върши работа: $\frac{1}{4}\big(\frac{1}{4}\big)^0 + \frac{3}{4}\big(\frac{3}{4}\big)^0 = 1$.

Тогава прилагаме теоремата на Akra-Bazzi и получаваме $T(n) \asymp n^0\Big(1 + \int_1^n \frac{1}{t}dt\Big) = \Theta(log(n))$.

3.2.7 Задачи

Основната идея за строене на рекурентно уравнение за сложността на рекурсивен алгоритъм е за всяко рекурсивно извикване с параметър k да добавим отдясно на равенството $T(k)$. Останалата *работа* ще представлява нехомогенната част на рекурентното уравнение.

Задача 3.25. Даден е следният алгоритъм:

```
1. Func(n) : // n ∈ ℕ⁺
2.     s ← 0;
3.     for i ← 1 to n - 1
4.         s ← s + 2 * Func(i) + 1;
5.     return s;
```

Каква е сложността му по време (спрямо n)?

Решение. Имаме $n-1$ на брой рекурсивни извиквания.. това означава, че за всяко едно от тях трябва да добавим отдясно на равенството $T(\textit{подходящ параметър})$. Останалата *работа* в случая е инициализацията на s, инкрементацията на i и актуализацията на s, т.е. $\Theta(n)$. Тя може да се погълне от рекурсивните извиквания (по единица на извикване). Тоест рекурентното уравнение е $T(n) = T(n-1) + T(n-2) + \cdots + T(1) + \Theta(1)$. Вече намерихме асимптотиката на подобно рекурентно уравнение в **Задача 3.8**.

$$\left|\begin{array}{l} T(n) = T(n-1) + T(n-2) + \cdots + T(1) + \Theta(1) \\ T(n-1) = T(n-2) + T(n-3) + \cdots + T(1) + \Theta(1) \end{array}\right.$$

Тогава $T(n) - T(n-1) = T(n-1) + \Theta(1) - \Theta(1)$ или още $T(n) = 2T(n-1) + \Theta(1)$. Вече може да го решим чрез метода с характеристично уравнение.

- (Хомогенна част)
  $x = 2 \ \mapsto \ \{2\}_M$
- (Нехомогенна част)
  $\Theta(1) \ \mapsto \ \{1\}_M$

Оттук получаваме, че $T(n) = c_1 1^n + c_2 2^n = \Theta(2^n)$.

Задача 3.26. Даден е следният алгоритъм:

```
1. Func(n) : // n ∈ ℕ⁺
2.     s ← 0;
3.     for i ← 1 to n - 1
4.         s ← s + Func(i) + Func(i) + 1;
5.     return s;
```

Каква е сложността му по време (спрямо n)?

Решение. Имаме $2(n-1)$ на брой рекурсивни извиквания.. това означава, че за всяко едно от тях трябва да добавим отдясно на равенството $T(\textit{подходящ параметър})$. Останалата *работа* в случая е инициализацията на s, инкрементацията на i и актуализацията на s, т.е. $\Theta(n)$, което отново може да се погълне от рекурсивните извиквания. Тоест рекурентното уравнение е $T(n) = 2T(n-1) + 2T(n-2) + \cdots + 2T(1) + \Theta(1)$.

$$\left|\begin{array}{l} T(n) = 2T(n-1) + 2T(n-2) + \cdots + 2T(1) + \Theta(1) \\ T(n-1) = 2T(n-2) + 2T(n-3) + \cdots + 2T(1) + \Theta(1) \end{array}\right.$$

Тогава $T(n) - T(n-1) = 2T(n-1) + \Theta(1) - \Theta(1)$ или още $T(n) = 3T(n-1) + \Theta(1)$. Вече може да го решим чрез метода с характеристично уравнение.

- (Хомогенна част)
  $x = 3 \ \mapsto \ \{3\}_M$
- (Нехомогенна част)
  $\Theta(1) \ \mapsto \ \{1\}_M$

Оттук получаваме, че $T(n) = c_1 1^n + c_2 3^n = \Theta(3^n)$.

Глава 4

Дизайн на алгоритми

4.1 Уводни алгоритми

Ще започнем с това да изтъкнем разликата между изчислителна задача и алгоритъм. Грубо казано, изчислителна задача е нещо общо, а алгоритъмът е прозрачна кутия (т.е. знаем ѝ механизъма на работа), която *"решава"* дадена изчислителна задача. Може да имаме много алгоритми, решаващи една и съща изчислителна задача. За да правим разлика, когато говорим за изчислителна задача ще използваме термините *екземпляр (instance)* и *решение (solution)* вместо *вход* и *изход*. Вторите ще използваме, когато говорим за алгоритъм.

Задача 4.1. Даден е масив $A[1..n] \in (\mathbb{N}_0)^n$ и много на брой заявки от вида:

$$\begin{cases} \textbf{Instance: } i, j \in \{1, \ldots, n\} \\ \textbf{Solution: } \sum_{k=1}^{j} A[k] \end{cases}$$

Предложете двойка алгоритми (индекс и заявка), решаващи изчислителната задача. Индексирането бива извиквано точно веднъж. Заявки биват извиквани многократно. Заявка не може бъде извикана преди (единственото) извикване на индекса. Разгледайте следния пример, имащ две заявки:

$$\begin{cases} \textbf{Given: } A[1..7] = [4, 2, 5, 7, 1, 1, 10] \\ \begin{cases} \textbf{Instance: } i = 2, j = 5 \\ \textbf{Solution: } 15 \text{ //since } 2 + 5 + 7 + 1 = 15 \\ \textbf{Instance: } i = 5, j = 2 \\ \textbf{Solution: } 0 \text{ //since the neutral element of } + \text{ is } 0 \end{cases} \end{cases}$$

Решение. Нека да разгледаме първо индекирането:

```
1. Index(A[1..n]) : // A ∈ (ℕ₀)ⁿ, n ∈ ℕ⁺
2.     Sum[1..n] ← [0, ..., 0];
3.     Sum[1] ← A[1];
4.     for i ← 1 to n
5.         Sum[i] ← Sum[i − 1] + A[i];
6.     return Sum[1..n];
```

След като направихме индекс (връщаният масив $Sum[1..n]$), трябва да го използваме по подходящ начин в заявката:

```
1. Query(Sum[1..n], i, j) : // Sum ∈ (ℕ₀)ⁿ, i, j ∈ {1, ..., n}, n ∈ ℕ⁺
2.     Sum[1..n] ← [0, ..., 0];
3.     if i < 1 or j > n then
4.         return −1;
5.     if i > j then
6.         return 0;
7.     if i = 1 then
8.         return Sum[j];
9.     return Sum[j] − Sum[i − 1];
```

Алгоритмите са със сложност по време съответно $\langle \underbrace{\Theta(n)}_{Index(A[1..n])}, \underbrace{\Theta(1)}_{Query(Sum[1..n],i,j)} \rangle$.

Задача 4.2. Даден е непразен сортиран масив $A[1..n] \in (\{0, 1, \ldots, n\})^n$ съставен от уникални елементи. Да се намери $k \in \{0, 1, \ldots, n\}$ такова, че k не е елемент на $A[1..n]$. Предложете колкото можете по-бърз алгоритъм, решаващ изчислителната задача. Разгледайте следния пример:

$$\begin{cases} \textbf{Instance: } A[1..7] = [0, 1, 2, 4, 5, 6, 7] \\ \textbf{Solution: } 3 \text{ //since 3 is not an element of } A[1..7] \end{cases}$$

Решение. За разлика от предходната задача, тук нямаме много заявки, а точно една. С други думи може да обединим индекса и заявката в едно общо решение:

```
1. Task2(A[1..n]) : // A ∈ (ℕ₀)ⁿ, n ∈ ℕ⁺
2.     left ← 1;
3.     right ← n;
4.     while left < right do
5.         mid ← ⌊(left+right)/2⌋;
6.         if A[mid] + 1 = mid then
7.             left ← mid + 1;
8.         else
9.             right ← mid;
10.    if A[n] + 1 = n then
11.        return left;
12.    return left − 1;
```

Алгоритъмът е със сложност по време $\Theta(\log n)$.

Задача 4.3. Даден е непразен масив $A[1..n] \in \mathbb{Z}^n$ съставен от уникални елементи. Да се намери множеството от наредени тройки $\{\langle i, j, k\rangle \in \{1, \ldots, n\}^3 \mid i < j < k \wedge A[i] + A[j] + A[k] = 0\}$. Предложете колкото можете по-бърз алгоритъм, решаващ изчислителната задача. Разгледайте следния пример:

$$\begin{cases} \textbf{Instance: } A[1..7] = [-10, 1, 2, 3, 7, 8, 9] \\ \textbf{Solution: } \langle 1, 2, 7\rangle, \langle 1, 3, 6\rangle, \langle 1, 4, 5\rangle \end{cases}$$

Решение. Отново, аналогично на **Задача 4.2** нямаме нужда от отделен алгоритъм за индексация и заявка, тъй като имаме само една заявка:

```
1.  Task3(A[1..n]) : // A ∈ ℤ^n, n ∈ ℕ^+
2.      sort(A[1..n], order = ascending);
3.      Ans ← List.Init();                                  // лист от наредени тройки
4.      for i ← 1 to n − 2
5.          if A[i] ≥ 0 then
6.              break;
7.          j ← i + 1;
8.          k ← n;
9.          while j < k do
10.             if A[i] + A[j] + A[k] = 0 then
11.                 Ans.PushBack(⟨i, j, k⟩);
12.                 j ← j + 1;
13.                 k ← k − 1;
14.             else if A[i] + A[j] + A[k] < 0 then
15.                 j ← j + 1;
16.             else
17.                 k ← k − 1;
18.     return Ans;
```

Алгоритъмът е със сложност по време $\Theta(n^2)$.

Задача 4.4. Даден е непразен масив $A[1..n] \in \mathbb{R}^n$. Да се сортира възходящо масивът, използвайки единствено функцията $reverse : \mathbb{R}^n \times \{1,\dots,n\} \times \{1,\dots,n\} \to \mathbb{R}^n$, дефинирана по следния начин:

$$reverse(A[1..n], i, j) \leftrightharpoons \begin{cases} [A[1], \dots, A[i-1], \underbrace{A[j], A[j-1], \dots, A[i+1], A[i]}_{\text{в обратен ред}}, A[j+1], \dots, A[n]], & i < j \\ A[1..n], \text{ иначе} \end{cases}$$

Решение. Идеята е да вземем най-голям елемент и да го сложим на първа позиция, веднага след което на последната свободна такава (с други думи - сортираме отзад напред). Нека да разиграем малък пример. Нека $A[1..3] = [1, 3, 2]$. Първо поставяме 3 на пъра позиция чрез $reverse([1, 3, 2], 1, 2)$, т.е. получаваме $[3, 1, 2]$. След това веднага го слагаме най-отзад чрез $reverse([3, 1, 2], 1, 3)$, т.е. получаваме $[2, 1, 3]$. Така вече най-големият (най-голям в общия случай) елемент си е на позицията. Сега намираме втория най-голям елемент и го слагаме на първа позиция, след което го слагаме на предпоследна позиция чрез последователно извикване на $reverse([2, 1, 3], 1, 1) \mapsto [2, 1, 3]$ и $reverse([2, 1, 3], 1, 2) \mapsto [1, 2, 3]$. Сега откриваме третият най-голям елемент и го слагаме на първа позиция след което на трета позиция отзад напред (в случая е безсмислено разбира, т.е. последователно извикваме $reverse([1, 2, 3], 1, 1) \mapsto [1, 2, 3]$ и $reverse([1, 2, 3], 1, 1) \mapsto [1, 2, 3]$. Така масивът е сортиран. Както сме свикнали от C/C++ ще си дефинираме и използваме спомагателни функции - за чистота на кода и за по-лесно доказателство на коректността. Нека първо си дефинираме функция, намираща индекса на най-левия максимален елемент от префикс на масива:

```
1. findIdxOfMax(A[1..n], k) : // A ∈ ℝ^n, k ∈ {1, ..., n}, n ∈ ℕ^+
2.     maxIdx ← 1;
3.     for i ← 2 to k
4.         if A[i] > A[maxIdx] then
5.             maxIdx ← i;
6.     return maxIdx;
```

Използвайки тази функция, вече можем чисто да напишем на код идеята от по-горе:

```
1. Task4(A[1..n]) : // A ∈ ℝ^n, n ∈ ℕ^+
2.     for k ← n downto 2
3.         maxIdx ← findIdxOfMax(A[1..n], k);
4.         reverse(A[1..n], 1, maxIdx);
5.         reverse(A[1..n], 1, k);
6.     return A[1..n];
```

Алгоритъмът е със сложност по време $\Theta(n^2)$.

Задача 4.5. Нека е дадена непразна булева матрица $A[1..m][1..n] \in (\{0,1\}^n)^m$, чиито колонки са сортирани низходящо, т.е. $\big(\forall r \in \{1,\dots,m-1\}\big)\big(\forall c \in \{1,\dots,n\}\big)\big(A[r+1][c] \leq A[r][c]\big)$. Да се намери височината на най-високата колонка от единици, т.е. $\max\{\sum_{r=1}^{m} A[r][c] \mid c \in \{1,\dots,n\}\}$. Предложете колкото можете по-бърз алгоритъм, решаващ изчислителната задача. Разгледайте следния пример:

$$\begin{cases} \textbf{Instance: } A[1..4][1..5] = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \textbf{Solution: } 3 \text{ //since that is the highest row of ones} \end{cases}$$

Решение. Идеята е много проста. Откриваме височината на първата колонка и я запазваме в променлива h. След това проверяваме дали втроата колонка би имала повече единици. Ако има, то преглеждаме нейната височина и обновяваме h. Наблюдението, което използваме е, че тъй като търсим максимална по височина колонка, то ако втората колонка има по-малка височина от първата, то директно я прескачаме. Така се движим "стъпаловидно" от горе-ляво към долу-дясно. За нагледно с червен цвят е отбелязан пътят, който ще "измине" алгоритъмът на примерния вход:

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Ясно е, че клетките над червената начупена линия няма нужда да бъдат разглеждани. Вече, въоражени с идея, сме готови да реализираме алгоритъм, решаващ изчислителната задача:

```
1. Task5(A[1..m][1..n]) : // A ∈ ({0,1}^n)^m, n, m ∈ ℕ⁺
2.     i ← 0;
3.     for j ← 1 to n
4.         while i < m and A[i + 1][j] = 1 do
5.             i ← i + 1;
6.     return i;
```

Алгоритъмът е със сложност по време $\Theta(n) + O(m) = O(n + m)$.

Задача 4.6. Нека е дадена непразна матрица $A[1..m][1..n] \in ((\mathbb{N}_0)^n)^m$ от уникални елементи, със свойството, че всеки елемент е по-голям от северните и западните си съседи. Казано формално $\forall r \forall c\big(A[i][j] < A[i+1][j] \wedge A[i][j] < A[i][j+1]\big)$, за r и c подходящи: да не излизаме извън дефиниционната област. Нека също така е дадено число $k \in \mathbb{N}_0$. Да се провери дали елемента k се среща в матрицата $A[1..m][1..n]$. Предложете колкото можете по-бърз алгоритъм, решаващ изчислителната задача. Разгледайте следния пример:

$$\begin{cases} \textbf{Instance: } A[1..4][1..5] = \begin{bmatrix} 1 & 2 & 7 & 12 & 14 \\ 3 & 6 & 8 & 16 & 17 \\ 5 & 10 & 11 & 20 & 21 \\ 13 & 15 & 19 & 22 & 25 \end{bmatrix}, k = 15 \\ \textbf{Solution: } \text{TRUE} \end{cases}$$

Решение. Идеята е да започнем или в най-североизточният или най-югозападният край и стъпаловидно да търсим елемента. Нека за определеност започваме в североизточният ъгъл. Движим се надолу, когато търсения елемент е по-голям от текущия и наляво, ако търсеният елемент е по-малък. Разбира се, ако текущия и търсения елемент съвпадат връщаме TRUE. Нека да оцветим в червено пътят, който изминава алгоритъмът за да намери 15 в примерната матрица:

$$\begin{bmatrix} 1 & 2 & 7 & 12 & 14 \\ 3 & 6 & 8 & 16 & 17 \\ 5 & 10 & 11 & 20 & 21 \\ 13 & 15 & 19 & 22 & 25 \end{bmatrix}$$

Защо тази идея работи се убедете за упражнение! Може да го докажете дори формално. Вече сме готови да реализираме алгоритъм, решаващ изчислителната задача:

```
1. Task6(A[1..m][1..n], k) : // A ∈ ((ℕ₀)^n)^m, n, m ∈ ℕ⁺
2.     i ← 1;
3.     j ← n;
4.     while i ≤ m and j ≥ 1 do
5.         if A[i][j] = k then
6.             return TRUE;
7.         else if A[i][j] > k then
8.             j ← j − 1;
9.         else
10.            i ← i + 1;
11.    return FALSE;
```

Алгоритъмът е със сложност по време $O(n + m)$.

4.1.1 Добавяне на фиктивни променливи

Доказателството за коректност на алгоритъм (зададен чрез псевдокод) не винаги е възможно. Оказва се, че не ни достигат променливите, дадени ни в псевдокода. В зависимост от това, къде се е провалило доказателството, добавяме нови, **фиктивни** променливи. Те не оказват никакво влияние на алгоритъма. Служат единствено за опорни точки в доказателството за коректност. Независимо дали алгоритъмът е итеративен или рекурсивен, такъв проблем може да настъпи. Сега ще се съсредоточим върху итеративни алгоритми.
Спорно е дали мястото на подсекцията е тук или като подсекция на ***Секция 2.1****.*

Задача 4.7. Даден е непразен масив $A[1..n] \in \mathbb{Z}^n$. Да се състави колкото се може по-бърз алгоритъм, намиращ мажорантата на $A[1..n]$, ако такава има. Съставете втори алгоритъм, който да установява дали мажоранта има.

Решение. Преди да изложим какъв да е псевдокод, ще докажем спомагателно твърдение. Въвеждаме следните означения:

Дефиниция (локална)

- $pair(p) \overset{\text{def}}{\leftrightarrow} \big(|p| = 2\big)$
- $pset(S) \overset{\text{def}}{\leftrightarrow} \big(\forall p \in S\big)\big(pair(p)\big)$
- $cpair_{A[1..n]}(i,j) \overset{\text{def}}{\leftrightarrow} \big(i \in \{1,\dots,n\}\big) \wedge \big(j \in \{1,\dots,n\}\big) \wedge \big(A[i] \neq A[j]\big)$ //i.e. contrary pair
- $comp_{A[1..n]}(\langle i,j\rangle, \langle i',j'\rangle) \overset{\text{def}}{\leftrightarrow} \big(i \neq i'\big) \wedge \big(i \neq j'\big) \wedge \big(j \neq i'\big) \wedge \big(j \neq j'\big)$ //i.e. compatible
- $cpset_{A[1..n]}(S) \overset{\text{def}}{\leftrightarrow} pset(S) \wedge \big(\forall \langle i,j\rangle \in S\big)\big(cpair_{A[1..n]}(i,j)\big) \wedge \big(\forall p_1 \in S\big)\big(\forall p_2 \in S\big)\big(comp_{A[1..n]}(p_1,p_2)\big)$
- $mcpset_{A[1..n]}(S) \overset{\text{def}}{\leftrightarrow} cpset_{A[1..n]}(S) \wedge \forall i \forall j \big(\langle i,j\rangle \notin S \Rightarrow \neg cpset_{A[1..n]}(S \cup \{\langle i,j\rangle\})\big)$
- $idx(S) \leftrightharpoons Dom(S) \cup Rng(S)$ //also known as field

Забележка. Казано неформално:

- ц-чифтове правим на индекси от $A[1..n]$, съответстващи на различни елементи
- ц-чифтове $\langle i,j\rangle$ и $\langle i',j'\rangle$ наричаме съвместими, ако $|\{i,i',j,j'\}| = 4$
- множество от съвместими ц-чифтове наричаме максимално, ако не можем да добавим ц-чифт, така че да остане съвместимо (т.е. всеки два елемента са съвместими)

Твърдение 4.1

Нека $A[1..n]$ има мажоранта m. Тогава:

- $\forall S\big(mcpset_{A[1..n]}(S) \Rightarrow idx(S) \neq \{1,\dots,n\}\big)$
- $\forall S\Big(mcpset_{A[1..n]}(S) \Rightarrow \big(\forall i \in \{1,\dots,n\} \setminus idx(S)\big)\big(A[i] = m\big)\Big)$

Забележка. Казано неформално:

- за всяко максимално множество от съвместими ц-чифтове, има нечифтован индекс
- за всяко максимално множество от съвместими ц-чифтове, всеки нечифтован индекс отговаря на елемент, равен на мажорантата

Доказателство. Нека $A[1..n]$ е произволен масив, съдържащ мажоранта m. Нека S е множество: $mcpset_{A[1..n]}(S)$ и нека означим броят на срещанията на мажорантата с k. Нека $i_1, \ldots, i_k$ са (точно) индексите на срещанията на мажорантата в $A[1..n]$. Допускаме, че индексите на всички срещания на мажорантата са ц-чифтовани, т.е. има k различни (от $i_1, \ldots, i_k$ и помежду си) индекси $j_1, \ldots, j_k$ такива, че $\{\langle i_1, j_1\rangle, \ldots, \langle i_k, j_k\rangle\} \subseteq S$. Тоест $A[1..n]$ се състои от поне $2k$ елементи, или казано с други думи $n \geq 2k$. От друга страна, знаем че $k \geq \lfloor \frac{n}{2} \rfloor + 1 > \frac{n}{2}$, откъдето $2k > n$, което е абсурд. Следователно има поне един не ц-чифтован индекс i, отговарящ на мажорантата. Нека сега допуснем, че има не ц-чифтован индекс j, отговарящ на елемент, различен от мажорантата. Тогава $cpset_{A[1..n]}(S \cup \{\langle i, j\rangle\})$, което противоречи с максималността на S. □

Пример. Нека $A[1..10] = [1, 2, 3, 1, 1, 1, 4, 1, 1, 5]$. Примери за максимални множества от съвместими ц-чифрове са $S_1 = \{\langle 1, 2\rangle, \langle 3, 4\rangle, \langle 5, 7\rangle, \langle 6, 10\rangle\}, S_2 = \{\langle 2, 3\rangle, \langle 7, 10\rangle\}$. В първия случай множеството от не ц-нечифтованите индекси е $\{8, 9\}$, а във втория е $\{1, 4, 5, 6, 8, 9\}$. И за двете множества S_1 и S_2 имаме, че:

- имат не ц-чифтован индекс, т.е. $idx(S_1) = \{1, 2, 3, 4, 5, 6, 7, 10\} \neq \{1, \ldots, 10\}$ и $idx(S_2) = \{2, 3, 7, 10\} \neq \{1, \ldots, 10\}$;
- всички не ц-чифтовани индекси отговарят на елемент, равен на мажорантата $m = 1$, т.е. за S_1 имаме $A[8] = A[9] = m$, а за S_2 имаме $A[1] = A[4] = A[5] = A[6] = A[8] = A[9] = m$.

Първоначално ще изложим алгоритъм, намиращ мажорантата, при условие, че входният масив съдържа мажоранта:

```
1.  MajorityElement(A[1..n]) : // A ∈ ℤ^n, n ∈ ℕ^+, A has majority element
2.      curr ← 1;
3.      cnt ← 1;
4.      for i ← 2 to n
5.          if A[curr] = A[i] then
6.              cnt ← cnt + 1;
7.          else
8.              cnt ← cnt − 1;
9.              if cnt = 0 then
10.                 curr ← i;
11.                 cnt ← 1;
12.     return A[curr];
```

Сега ще се опитаме да докажем коректността на алгоритъма (няма да успеем).

Инвариант

При всяко k-то достигане на ред 4 имаме, че

- $i = k + 1$
- $curr \in \{1, \ldots, i-1\}$
- $cnt > 0$
- $\exists S\big(mcpset_{A[1..i-1]}(S) \wedge \overbrace{(cnt > 0 \Rightarrow curr \notin idx(S))}^{\equiv\ curr \notin idx(S)} \wedge (\overbrace{cnt}^{\text{не ц-чифт.}} + \overbrace{|idx(S)|}^{\text{ц-чифт.}} = n)\big)$

Забележка. Казано неформално: има максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i-1]$, в което $curr$ не е ц-чифтован и броят не ц-чифтовани индекси на $A[1..i-1]$ е $cnt > 0$.

База. При $k = 1$-во достигане на ред 4 имаме, че $i = 2$, $curr = 1 \in \{1, \ldots, 2-1\} = \{1, \ldots, i-1\}$. Разглеждаме подмасивът $A[1..1]$. Той е съставен от един елемент. Тогава $S = \emptyset$ ни е свидетел. Наистина, $1 \notin \emptyset$ и има $cnt = 1 > 0$ не ц-чифтовани индекси.

Поддръжка. Нека е вярно за някое k-то непоследно достигане на ред 4 и нека S_{old} ни е свидетел, т.е. $mcpset_{A[1..i_{old}-1]}(S_{old}) \wedge (cnt_{old} > 0 \Rightarrow curr_{old} \notin idx(S_{old})) \wedge (cnt_{old} + |idx(S_{old})| = n)$. Сега се изпълнява за k-ти път тялото на цикъла. Имаме $i_{new} = i_{old} + 1 = k + 2$.

сл.1 $(A[curr_{old}] = A[i_{old}])$

Тогава $curr_{new} = curr_{old} \in \{1, \ldots, i_{old} - 1\} \subset \{1, \ldots, \overbrace{i_{new} - 1}^{i_{old}}\}$, $cnt_{new} = cnt_{old} + 1$. От ИП имаме, че S_{old} е максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i_{old} - 1]$, $curr_{old}$, не е ц-чифтован и броят не ц-чифтовани индекси на $A[1..i_{old} - 1]$ е $cnt_{old} > 0$. Лесно се проверява, че S_{old} е максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i_{new} - 1]$, $curr_{new}$ не е ц-чифтован и броят не ц-чифтовани инекси на $A[1..i_{new} - 1]$ е $cnt_{new} > 1 > 0$, т.е. S_{old} е свидетел: $mcpset_{A[1..i_{new}-1]}(S_{old}) \wedge (cnt_{new} > 0 \Rightarrow curr_{new} \notin idx(S_{old})) \wedge (cnt_{new} + |idx(S_{old})| = n)$.

сл.2 $(A[curr_{old}] \neq A[i_{old}])$

От ИП имаме, че S_{old} е максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i_{old} - 1]$, $curr_{old}$ не е ц-чифтован и броят не ц-чифтовани индекси на $A[1..i_{old}-1]$ е $cnt_{old} > 0$. Тъй като S_{old} е максимално, то всички не ц-чифтовани индекси отговарят на елементи с една и съща стойност $A[curr_{old}] \neq A[i_{old}]$.

сл.2.1 $(cnt_{old} > 1)$

Тогава $cnt_{new} = cnt_{old} - 1 > 0$, $curr_{new} = curr_{old} \in \{1, \ldots, i_{old} - 1\} \subset \{1, \ldots, \overbrace{i_{new} - 1}^{i_{old}}\}$. Нека $j \neq curr_{old}$ е индекс на $A[1..i_{old} - 1]$, който не е ц-чифтован. Такъв има поради горното разъждение и защото $cnt_{old} > 1$. Нещо повече, $A[j] \neq A[i_{old}]$. Дефинираме $S_{new} \leftrightharpoons S_{old} \cup \{\langle j, i_{old}\rangle\}$. Лесно се проверява, че S_{new} е максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i_{new} - 1]$, $curr_{new}$ не е ц-чифтован и броят не ц-чифтованите индекси на $A[1..i_{new} - 1]$ е cnt_{new}, т.е. S_{new} е търсеният свидетел.

сл.2.2 $(cnt_{old} = 1)$

Тогава $cnt_{new} = 1$, $curr_{new} = i_{old} \in \{1, \ldots, i_{old}\} = \{1, \ldots, i_{new} - 1\}$. Единствените два не ц-чифтовани индекса на $A[1..i_{old}]$ са $curr_{old}$ и i_{old} и имаме, че $A[curr_{old}] \neq A[i_{old}]$.

Дефинираме $S_{new} \leftrightharpoons S_{old} \cup \{\langle curr_{old}, i_{old} \rangle\}$. Лесно се проверява, че S_{new} е максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i_{new} - 1]$, но $cnt_{new} > 0$ и $curr_{new} = i_{old}$ е ц-чифтован (а ние искахме да докажем импликацията $cnt_{new} > 0 \Rightarrow curr_{new}$ е не ц-чифтован). **Не успяхме да намерим свидетел**. В общия случай, такъв може да има, но ние просто да не сме успели да го намерим (в конкретната задача може да се докаже, че наистина няма).

Това всъщност беше очакван резултат, тъй като при вход $A[1..4] = [1, 2, 1, 1]$, още след първата итерация на цикъла ц-чифтоваме индексите 1 и 2 и оставаме без не ц-чифтовани индекси, където индексите са на масива $A[1..2]$.

Нека сега разгледаме следната преработка на превдокода:

```
1.  MajorityElementExpanded(A[1..n]) : // A ∈ ℤ^n, n ∈ ℕ^+, A has majority element
2.      curr ← 1;
3.      cnt ← 1;
4.      t ← 0;
5.      for i ← 2 to n
6.          if A[curr] = A[i] then
7.              cnt ← cnt + 1;
8.          else
9.              cnt ← cnt − 1;
10.             if cnt = 0 then
11.                 curr ← i;
12.                 cnt ← 1;
13.                 t ← 1 − t;
14.     return A[curr];
```

Ясно е, че променливата t не оказва влияние на останалите променливи и на изпълнението на кода изобщо. С други думи, ясно е, че t е фиктивна променлива. В някои ситуации, където не е толкова тривиално, е необходимо да се направи доказателство с инвариант, за еквивалентността на променливите за всяко k-то достигане на ред 4 и ред 5 съответно за първоначалния и модифицирания алгоритъм. Вече сме готови да формулираме инвариант.

Инвариант

При всяко k-то достигане на ред 5 имаме, че

- $i = k + 1$
- $curr \in \{1, \ldots, i - 1\}$
- $cnt > 0$
- $t \in \{0, 1\}$
- $\exists S \big(mcpset_{A[1..i-1]}(S) \wedge \overbrace{(cnt - t > 0 \Rightarrow curr \notin idx(S))}^{\text{ако има не ц-чифт., то } curr \text{ е такъв}} \wedge (\overbrace{cnt - t}^{\text{не ц-чифт.}} + \overbrace{|idx(S)|}^{\text{ц-чифт.}} = n) \big)$

Забележка. За разлика от първоначалния инватирант, тук антецедента $cnt - t > 0$ на импликацията $cnt - t > 0 \Rightarrow curr \notin idx(S)$ не е винаги $true$.

База. При $k=1$-во достигане на ред 5 имаме, че $i=2$, $curr=1\in\{1\}$, $cnt=1>0$, $t=0\in\{0,1\}$. Разглеждаме подмасивът $A[1..1]$. Той е съставен от един елемент. Тогава $S=\emptyset$ ни е свидетел. Наистина, $1\notin\emptyset$ и има $cnt-t=1-0=1>0$ не ц-чифтовани индекси.

Поддръжка. Нека е вярно за някое k-то непоследно достигане на ред 5 и нека S_{old} е такова, че $mcpset_{A[1..i_{old}-1]}(S_{old})\wedge(cnt_{old}-t_{old}>0\Rightarrow curr_{old}\notin idx(S_{old}))\wedge(cnt_{old}-t_{old}+|idx(S_{old})|=n)$. Сега се изпълнява за k-ти път тялото на цикъла. Имаме $i_{new}=i_{old}+1=k+2$. Колкото до t, имаме $t_{new}=t_{old}$ или $t_{new}=1-t_{old}$, т.е. $t_{new}\in\{0,1\}$.

сл.1 $(A[curr_{old}]=A[i_{old}])$

Тогава $curr_{new}=curr_{old}\in\{1,\ldots,i_{old}-1\}\subset\{1,\ldots,\overbrace{i_{new}-1}^{i_{old}}\}$, $cnt_{new}=cnt_{old}+1$ и $t_{new}=t_{old}$. От ИП имаме, че S_{old} е максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i_{old}-1]$, $curr_{old}$ не е ц-чифтован (при условие, че има не ц-чифтовани индекси) и броят не ц-чифтовани индекси на $A[1..i_{old}-1]$ е $cnt_{old}-t_{old}$.

сл.1.1 $(cnt_{old}-t_{old}=0)$

Тогава $curr_{old}$ е чифтован спрямо S_{old}. Нека $p\in S_{old}$ е ц-чифта, който съдържа $curr_{old}$. Без ограничение на общността, нека $curr_{old}$ се среща в дясната координата на p, тоест $p=\langle j,curr_{old}\rangle$, където $j\in\{1,\ldots,i_{old}-1\}:\ A[j]\neq A[curr_{old}]$. Дефинираме $S_{new}\leftrightharpoons(S_{old}\setminus\{p\})\cup\{\langle j,i_{old}\rangle\}$. Лесно се проверява, че S_{new} е максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i_{new}-1]$, $curr_{new}=curr_{old}$ не е ц-чифтован спрямо S_{new} (тук условието да имаме поне един нечифтован елемент е изпълнено) и броят не ц-чифтовани елементи на $A[1..i_{new}-1]$ е $cnt_{new}-t_{new}=1$. Тоест, S_{new} е свидетел: $mcpset_{A[1..i_{new}-1]}(S_{new})\wedge(cnt_{new}>t_{new}\Rightarrow curr_{new}\notin idx(S_{new}))\wedge(cnt_{new}-t_{new}+|idx(S_{new})|=n)$.

сл.1.2 $(cnt_{old}-t_{old}>0)$

Тогава $curr_{new}=curr_{old}$ не е ц-чифтован (спрямо S_{old}). Сега, лесно се проверява, че S_{old} е максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i_{new}-1]$, $curr_{new}$ не е ц-чифтован (тук условието да имаме поне един нечифтован елемент е изпълнено) и броят не ц-чифтовани елементи на $A[1..i_{new}-1]$ е $cnt_{new}-t_{new}$. С други думи, S_{old} е свидетел: $mcpset_{A[1..i_{new}-1]}(S_{old})\wedge(cnt_{new}>t_{new}\Rightarrow curr_{new}\notin idx(S_{old}))\wedge(cnt_{new}-t_{new}+|idx(S_{old})|=n)$.

сл.2 $(A[curr_{old}]\neq A[i_{old}])$

От ИП имаме, че S_{old} е максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i_{old}-1]$, $curr_{old}$ не е ц-чифтован спрямо S_{old} (при условие, че има не ц-чифтовани индекси) и броят не ц-чифтовани индекси на $A[1..i_{old}-1]$ е $cnt_{old}-t_{old}$. Тъй като S_{old} е максимално, то всички не ц-чифтовани индекси отговарят на елементи с една и съща стойност $A[curr_{old}]\neq A[i_{old}]$.

сл.2.1 $(cnt_{old}-t_{old}>1)$ $//cnt_{old}>t_{old}+1\geq 1$, т.е. $cnt_{old}>1$

Тогава $cnt_{new}=cnt_{old}-1>0$, $curr_{new}=curr_{old}\in\{1,\ldots,i_{old}-1\}\subset\{1,\ldots,\overbrace{i_{new}-1}^{i_{old}}\}$ и $t_{new}=t_{old}$. Нека $j\neq curr_{old}+t_{old}$ е индекс, който не е ц-чифтован. Такъв има поради горното разъждение и защото $cnt_{old}-t_{old}>1$. Нещо повече, $A[j]\neq A[i_{old}]$. Дефинираме $S_{new}\leftrightharpoons S_{old}\cup\{\langle j,i_{old}\rangle\}$. Лесно се проверява, че S_{new} е максимално множество от съвместими ц-чифтове, където индексите са на $A[1..i_{new}-1]$, $curr_{new}$ не е ц-чифтован спрямо S_{new} (тук условието да имаме поне един нечифтован елемент е изпълнено) и броят на ц-чифтованите елементи на $A[1..i_{new}-1]$ е $cnt_{new}-t_{new}=cnt_{old}-t_{old}-1$, т.е. S_{new} е търсеният свидетел.

сл.2.2 $(cnt_{old} - t_{old} = 1)$ $//cnt_{old} = t_{old} + 1 \leq 2$, но $cnt_{old} > 0$ т.е. $cnt_{old} \in \{1, 2\}$

Тогава $cnt_{new} = t_{new} = 1$ и $curr_{new} \in \{1, \ldots, i_{new} - 1\}$. Наистина, от ИП имаме, че $i_{old} \in \{1, 2\}$ и $t_{old} \in \{0, 1\}$ и директно проверяваме следните два подслучая:

сл.2.2.1 $(cnt_{old} = 1 \wedge t_{old} = 0)$

сл.2.2.2 $(cnt_{old} = 2 \wedge t_{old} = 1)$

Дефинираме $S_{new} \leftrightharpoons S_{old} \cup \{\langle curr_{old}, i_{old}\rangle\}$. Лесно се проверява, че S_{new} е максимално множество от съвместими ц-чифтове, където индексите са от $A[1..i_{new} - 1]$ и има $cnt_{new} - t_{new} = 0$ не ц-чифтовани индекси спрямо S_{new} от $A[1..i_{new} - 1]$.

сл.2.3 $(cnt_{old} - t_{old} = 0)$ $//cnt_{old} = t_{old} = 1$

Тогава $cnt_{new} = 1$, $curr_{new} = i_{old} \in \{1, \ldots, i_{old}\} = \{1, \ldots, i_{new} - 1\}$ и $t_{new} = 1 - t_{old} = 0$. Лесно се проверява, че S_{old} е максимално множество от съвместими ц-чифтове, където индексите са от $A[1..i_{new} - 1]$ и има $cnt_{new} - t_{new} = 1$ не ц-чифтован индекс на $A[1..i_{new} - 1]$ и при това той е $curr_{new}$.

Терминация. При $k = n$-то достигане на ред 5 за първи път не влизаме в тялото на цикъла. От инварианта имаме, че:

- $i = n + 1$
- $curr \in \{1, \ldots, n\}$
- $cnt > 0$
- $t \in \{0, 1\}$
- $\exists S\big(mcpset_{A[1..n]}(S) \wedge (cnt - t > 0 \Rightarrow curr \notin idx(S)) \wedge (cnt - t + |idx(S)| = n)\big)$

Допускаме, че $cnt - t \not> 0$, т.е. $cnt - t \leq 0$, но $cnt > 0$ и $t \in \{0, 1\}$ и значи $cnt = t = 1$. Оттук $cnt - t = 0$. Следователно $|idx(S)| = n$. Сега, тъй като $mcpset_{A[1..n]}(S)$, в частност $cpset_{A[1..n]}(S)$, то $idx(S) = \{1, \ldots, n\}$. От друга страна имаме по условие, че $A[1..n]$ има мажоранта. Тогава от точка едно на **Твърдение 4.1** заключаваме, че $idx(S) \neq \{1, \ldots, n\}$, което е абсурд. Следователно $cnt - t > 0$. Тогава $curr \notin idx(S)$, но $curr \in \{1, \ldots, n\}$, т.е. $curr \in \{1, \ldots, n\} \setminus idx(S)$. Отново се възползваме от максималността на S и от точка две на **Твърдение 4.1** заключаваме, че $A[curr]$ е мажорантата, което е точно връщаната стойност, директно след цикъла.

Сега остана да съставим алгоритъм, който да установява дали $A[1..n]$ има мажоранта:

```
1. ContainsMajorityElement(A[1..n]) : // A ∈ ℤ^n, n ∈ ℕ^+
2.     m ← MajorityElementExpanded(A[1..n]);
3.     cnt ← 0;
4.     for i ← 1 to n
5.         if A[i] = m then
6.             cnt ← cnt + 1;
7.     return cnt ≥ ⌊n/2⌋ + 1;
```

Докажете коректността за упражнение. Сложността по време и на трите алгоритъма е $\Theta(n)$.

4.1.2 Амортизирана сложност (по време)

Няма да даваме формална дефиниция на понятието, тъй като не е по конспекта на курса. Въпреки това, всеки себеуважаващ се програмист, трябва да знае какво представлява амортизираната сложност, поне на интуитивно ниво. Амортизираната сложност (по време) се среща предимно при алгоритми от тип заявка. Цялата идея е да получим средната (в смисъл на математическо очакване) сложност по време на заявката. Нека да разгледаме пример.

Пример. Разглеждаме $std :: vector :: push\_back$. Нека имаме вектор $v[1..4] = [1, 2, 3, 4]$ и $v.capacity() = 4$, където $vector :: capacity$ ни връща заделената памет. Сега ще разгледаме какво се случва, когато добавяме елементи.

- $v.push\_back(5)$
  - няма алокирана памет, така че алокираме 8 единици памет: $[] \to [\_, \_, \_, \_, \_, \_, \_, \_]$
  - копираме елементите: $[\_, \_, \_, \_, \_, \_, \_, \_] \to [1, 2, 3, 4, \_, \_, \_, \_]$
  - добавяме новия елемент: $[1, 2, 3, 4, \_, \_, \_, \_] \to [1, 2, 3, 4, 5, \_, \_, \_]$
  - трием старата памет: $[1, 2, 3, 4] \to []$

  Общо време: $8 + 4 + 1 + 4 = 17$ единици работа.
- $v.push\_back(6)$
  - добавяме новия елемент: $[1, 2, 3, 4, 5, \_, \_, \_] \to [1, 2, 3, 4, 5, 6, \_, \_]$

  Общо време: 1 единица работа.
- $v.push\_back(7)$
  - добавяме новия елемент: $[1, 2, 3, 4, 5, 6, \_, \_] \to [1, 2, 3, 4, 5, 6, 7, \_]$

  Общо време: 1 единица работа.
- $v.push\_back(8)$
  - добавяме новия елемент: $[1, 2, 3, 4, 5, 6, 7, \_] \to [1, 2, 3, 4, 5, 6, 7, 8]$

  Общо време: 1 единица работа.

Общо време за добавяне на $5, 6, 7$ и 8: $17 + 1 + 1 + 1 = 20$ едциници работа.

- $v.push\_back(9)$
  - няма алокирана памет, така че алокираме 16 единици памет: $[] \to [\_, \ldots, \_]$ //16
  - копираме елементите: $[\_, \ldots, \_] \to [1, \ldots, 8, \_, \_, \_, \_, \_, \_, \_, \_]$
  - добавяме новия елемент: $[1, \ldots, 8, \_, \_, \_, \_, \_, \_, \_, \_] \to [1, \ldots, 8, 9, \_, \_, \_, \_, \_, \_, \_]$
  - трием старата памет: $[1, 2, 3, 4, 5, 6, 7, 8] \to []$

  Общо време: $16 + 8 + 1 + 8 = 33$ единици работа.

- $v.push\_back(10)$
  - добавяме новия елемент: $[1, \ldots, 8, 9, \_, \_, \_, \_, \_, \_, \_] \to [1, \ldots, 8, 9, 10, \_, \_, \_, \_, \_, \_]$

  Общо време: 1 единица работа.

...

- $v.push\_back(16)$
  - добавяме новия елемент: $[1, \ldots, 15, \_] \to [1, \ldots, 15, 16]$

  Общо време: 1 единица работа.

Общо време за добавяне на $9, 10, \ldots, 16$: $33 + \underbrace{1 + 1 + \cdots + 1}_{7} = 40$ едциници работа.

Аналогично, общото време за добавяне на $17, \ldots, 32$ отнема 80 единици работа и т.н. Може да забележим, че някои *push*-вания са бавни (линейни спрямо n), но те се срещат рядко. Повечето *push*-вания са константи. Сега взизмаме средното им време (по групички):

- $[5, 8]$

  Има общо 4 елемента и отнема общо 20 единици работа. Значи средно на *push*-ване е $\frac{20}{4} = 5$ единици работа.

- $[9, 16]$

  Има общо 8 елемента и отнема общо 40 единици работа. Значи средно на *push*-ване е $\frac{40}{8} = 5$ единици работа.

- $[17, 32]$

  Има общо 16 елемента и отнема общо 80 единици работа. Значи средно на *push*-ване е $\frac{80}{16} = 5$ единици работа.

...

Това осреднено време на заявка, наричаме амортизирана сложност (по време). Тоест амортизираната сложност на $std :: vector :: push\_back$ е $\Theta_a(1)$, където индексът a идва от *amortized* (локално означение). Разбира се, амортизираната сложност би била безсмислена, ако направим малък брой заявки, тъй като можем да попаднем на "бавна" заявка. Има смисъл да се разглежда **само** в контекста на много заявки. Забележете също така, че бавна заявка ни гарантира много на брой бързи заявки. Това ни гарантира, че не можем да попадаме много пъти (или всички пъти) в "лошия" случай. В рандомизираните алгоритми (каквото и да е това) нямаме такава гаранция!

Глава 5

Алгоритмична неподатливост

5.1 Компютърът - един голям автомат

Въпреки, че компютрите обикновено се отъждествяват с детерминирани машини на Тюринг (полезно и от теоретична и от практична гледни точки), те в действителност са детерминирани крайни автомати (тъй като имаме ограничена памет). Нека разгледаме пример.

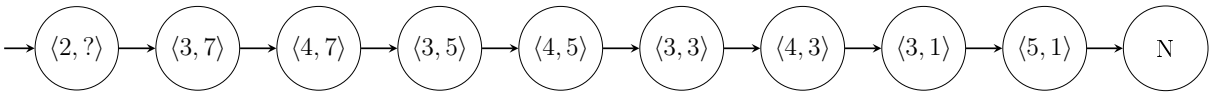
Пример. Дадени са следните програми (без вход от потребителя):

```
1. Func7() :
2.     a ← 7;
3.     while a > 1 do
4.         a ← a − 2;
5.     return a = 0;
```

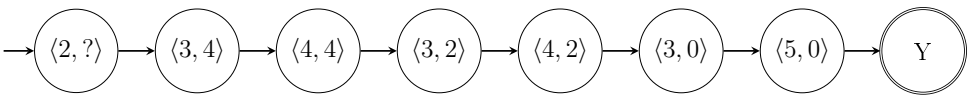
```
1. Func4() :
2.     a ← 4;
3.     while a > 1 do
4.         a ← a − 2;
5.     return a = 0;
```

Ясно е, че програмите връщат истина т.с.т.к. a е четно и лъжа иначе. Състоянията на автоматите ще бъдат наредени двойки от ред (на програмата) и стойността на променливите (в случая само a):

- $a = 7$



- $a = 4$



Макар по дефиниция, ДКА нямат ε-преходи, може да разглеждаме преходите като ε, тъй като от всяко състояние има най-много едно изходно ребро. Разбира се, изложеният метод работи само за предварително фиксирани променливи, т.е. без вход от потребителя. Помислете как бихме могли да обобщим да работи и с вход от потребитля.

5.2 Недетерминирани алгоритми

Както вече показахме в предходната секция, всяка програма **за компютър** може да се представи чрез краен детерминиран автомат. Нека сега да си мислим, че имаме магически компютър, който може да прави много изпълнения едновременно. Лесно се вижда, че такъв магически компютър би се представял чрез краен недетерминиран автомат. Програмите за стандартните компютри описвахме чрез псевдокод. Пргорамите за магическите компютри ще описаваме отново чрез псевдокод, но ще имаме бонус операция - недетерминиран избор. Разбираме се недетерминираните програми/алгоритми да връщат стойности единствено от {TRUE, FALSE}. Аналогично на ДКА, където дума се разпознава, ако има поне едно успешно изпълнение, тук ще връщаме TRUE, ако има поне едно успешно изпълнение.

Дефиниция 5.1: Изход на недетерминиран алгоритъм

Ще казваме, че недетерминиран алгоритъм връща TRUE, т.с.т.к. има поне едно успешно изпълнение.

Нека да разледаме слената задача:

Задача за разпознаване 1: COMPOSITES

$$\begin{cases} \textbf{Instance: } n \in \mathbb{N}^+ \\ \textbf{Question: } n \text{ съставно ли е?} \end{cases}$$

Решение. Ще покажем детерминирано и недетерминирано решение. Да започнем от детерминираното:

```
1. isCompositeDet(n) : // n ∈ ℕ⁺
2.     for q ← 2 to ⌊√n⌋
3.         if n ≡ 0 (mod q) then
4.             return TRUE;
5.     return FALSE;
```

Коректност няма да разглеждаме. Ясно как става - чрез инвариант. Времевата сложност на тази програма е $O(\sqrt{n})$. Големината на входа е $m = \log_2 n$, т.е. времевата сложност спрямо големината на входа е $O(\sqrt{2^m}) = O\big((\sqrt{2})^m\big)$ - експоненциална. А дали има полиномиален (спрямо големината на входа) детерминиран алгоритъм?

Нека сега разгледаме недетерминиран алгоритъм, решаващ задачата за полиномиално време:

```
1. isCompositeNonDet(n) : // n ∈ ℕ⁺
2.     q ← choose({2, 3, ..., ⌊√n⌋});          // недетерминиран избор
3.     if n ≡ 0 (mod q) then
4.         return SUCCESS;
5.     return FAIL;
```

Забележка. Можем да правим недетерминиран избор до предварително избрана константа за константно време. В случая, изборът **НЕ** е ограничен. Това което се случва отдолу е, че всяко битче на числото се избира да е нула или единица.. тоест недетерминираният избор отнема $\Theta(\log_2 \sqrt{n})$ време.

За да докажем коректността на недетерминирания алгоритъм трябва да покажем, че:

- при съставно n: има поне едно успешно изпълнение
- при просто n: всички изпълнения са неуспешни

Нека n е съставно. Тогава $\big(\exists q_0 \in \{2, 3, \ldots, \lfloor\sqrt{n}\rfloor\}\big)\big(n \equiv 0 \pmod{q_0}\big)$. Ясно е, че клона на изпълнение, в който choose е избрал q_0, е успешен. Нека сега n е просто. Тогава знаем, че $\big(\forall q \in \{2, 3, \ldots, \lfloor\sqrt{n}\rfloor\}\big)\big(n \not\equiv 0 \pmod{q_0}\big)$. Оттук е ясно, че всички изпълнения са неуспешни. Времевата сложност на тази програма е $\Theta(\log_2 \sqrt{n})$. Големината на входа е $m = \log_2 n$, т.е. времевата сложност спрямо големината на входа е $\Theta(\log_2 \sqrt{2^m}) = \Theta(\frac{1}{2}\log_2(2^m)) = \Theta(\frac{m}{2}) = \Theta(m)$ - полиномиална.

Нека сега разгледаме допълнението на горната задача и по-конкретно:

Задача за разпознаване 2: PRIMES

$$\begin{cases} \textbf{Instance: } n \in \mathbb{N}^+ \\ \textbf{Question: } n \text{ просто ли е?} \end{cases}$$

Решение. На пръв поглед, наивното разсъждение *просто ще върнем отрицанието на резултата от COMPOSITES*, ни върши работа. Нека обаче разгледа по-детайлно:

```
1. isPrimeDetNaive(n) : // n ∈ ℕ⁺
2. |  return ¬isCompositeDet(n);
```

Ясно е, че това решение работи. Също така е ясно, че сложността по време е експоненциална.

Нека сега разгледаме наивното за недетерминирания алгоритъм:

```
1. isPrimeNonDetNaive(n) : // n ∈ ℕ⁺
2. |  return ¬isCompositeNonDet(n);
```

За да докажем коректността на недетерминирания алгоритъм трябва да покажем, че:

- при просто n: има поне едно успешно изпълнение
- при съставно n: всички изпълнения са неуспешни

Ще покажем, че второто изискване не е изпълнено. Наистина, нека $n = 16$. Клона на изпълнение, в който $choose(\{2, 3, 4\})$ ни е избрало 3 е успешен, а ние искахме да е неуспешен. Тоест алгоритъмът не е коректен. Не е тривиално да се измисли полиномиален недетерминиран алгоритъм за тази задача.

Забележка

Нека $\mathscr{A}$ е задача за разпознаване. Тогава

- Ако имаме детерминиран алгоритъм за $\mathscr{A}$, то тривиално получаваме детерминиран алгоритъм за $\overline{\mathscr{A}}$ със същата времева сложност
- Ако имаме недетерминиран алгоритъм за $\mathscr{A}$, то нямаме обща схема по която да получим недетерминиран алгоритъм за $\overline{\mathscr{A}}$

5.3 Сводимост и NP-пълнота

Ще започнем с няколко прости дефиниции. Горе вече споменахме задачи за разпознаване, но не бяхме дали формална дефиниция. За целта нека припомним дефиниция от лекции:

Дефиниция 5.2: Изчислителна задача

Изчислителна задача $\mathcal{P}$ ще наричаме двуместна релация $\mathcal{P} \subseteq \mathcal{I} \times \mathcal{S}$, където $\mathcal{I}$ е безкрайно, изброимо множество от екземпляри (instances), а $\mathcal{S}$ е множество от решения (solutions).

Дефиниция 5.3: Задача за разпознаване

Задача за разпознване $\mathcal{P}$ ще наричаме изчислителна задача $\mathcal{P} \subseteq \mathcal{I} \times \{\text{TRUE, FALSE}\}$.

Нека въведем за улеснение следната локална дефиниция:

Дефиниция (локална)

Ще означаваме с $\mathcal{DP}$ класът от всички задачи за разпознаване.

Вече пристъпваме към по-съществените дефиниции от текущата глава:

Дефиниция 5.4: Класът P

$\text{P} \leftrightharpoons \{\mathcal{P} \in \mathcal{DP} | \big(\exists \mathscr{A}$ - дет. алгоритъм$\big)\big((\mathscr{A}$ *решава положителната част на* $\mathcal{P}$*)* & $(\mathscr{A}$ *има полиномиална времева сложност при изход "да"*$)\big)\}$.

Дефиниция 5.5: Класът NP

$\text{NP} \leftrightharpoons \{\mathcal{P} \in \mathcal{DP} | \big(\exists \mathscr{A}$ - недет. алгоритъм$\big)\big((\mathscr{A}$ *решава положителната част на* $\mathcal{P}$*)* & $(\mathscr{A}$ *има полиномиална времева сложност при изход "да"*$)\big)\}$.

Забележка. Какво значи алгоритъм да решава изчислителна задача няма да дефинираме формално. В горните две дефиниции се интересуваме само от положителната част на задачата за разпознаване. За отрицателната част нямаме никакви изисквания - алгоритъмът може да връща некоректен резултат, а може даже да зацикля!

Следните две дефиниции са еквивалентни, но често непрактични:

Дефиниция (алтернативна): Класът P

$\text{P} \leftrightharpoons \{\mathcal{P} \in \mathcal{DP} | \big(\exists \mathscr{A}$ - дет. алг.$\big)\big((\mathscr{A}$ *решава* $\mathcal{P}$*)* & $(\mathscr{A}$ *има полином. вр. сложност*$)\big)\}$.

Дефиниция (алтернативна): Класът NP

$\text{NP} \leftrightharpoons \{\mathcal{P} \in \mathcal{DP} | \big(\exists \mathscr{A}$ - недет. алг.$\big)\big((\mathscr{A}$ *решава* $\mathcal{P}$*)* & $(\mathscr{A}$ *има полином. вр. сложност*$)\big)\}$.

Забележка. Употребата на символът $\exists$ буквално се замества със "съществува". Скобите и логическото изложение са с цел по-лесно възприемане. Аналогично символът $\forall$ буквално се замества със "за всяко".

Целта на текущата секция е да даде основи, с които читателят да доказва, че съставянето на полиномиален алгоритъм за дадена ЗЗР е напълно нетривиална задача. За целта, въвеждаме следните две дефиниции:

Дефиниция 5.6: m-сводимост

Нека $\mathcal{A}, \mathcal{B} \in \mathcal{DP}$. Ще казваме, че $\mathcal{A}$ е m-сводима към $\mathcal{B}$, т.с.т.к. има **тотална** функция f: за всеки екземпляр x на $\mathcal{A}$ е изпълнено:

- $f(x)$ е екземпляр за $\mathcal{B}$
- решенията на $\mathcal{A}$ и $\mathcal{B}$ за екземпляри съответно x и $f(x)$ съвпадат
- f се изчислява чрез (детерминиран) алгоритъм

Ще пишем $\mathcal{A} \leq_m \mathcal{B}$.

Дефиниция 5.7: Karp - сводимост (полиномиална m-сводимост)

Нека $\mathcal{A}, \mathcal{B} \in \mathcal{DP}$. Ще казваме, че $\mathcal{A}$ е Karp - сводима към $\mathcal{B}$, т.с.т.к. има **тотална** функция f: за всеки екземпляр x на $\mathcal{A}$ е изпълнено:

- $f(x)$ е екземпляр за $\mathcal{B}$
- решенията на $\mathcal{A}$ и $\mathcal{B}$ за екземпляри съответно x и $f(x)$ съвпадат
- f се изчислява чрез детерминиран алгоритъм за полиномиално време

Ще пишем $\mathcal{A} \leq_m^p \mathcal{B}$ или $\mathcal{A} \leq_p \mathcal{B}$.

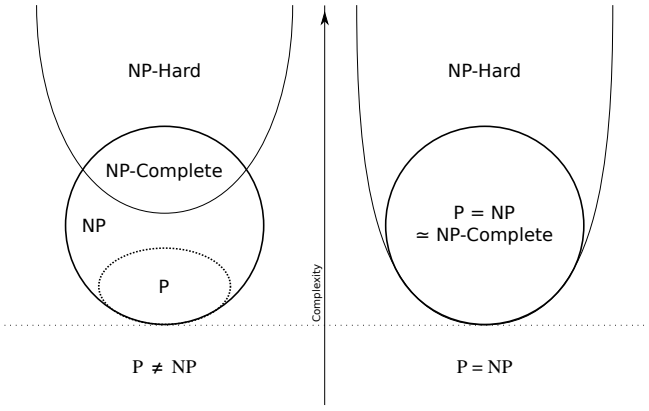
Вече сме готови да въведем основните (по-интересни) дефиниции:

Дефиниция 5.8: Класът NP-hard

$\text{NP-h} \leftrightharpoons \{\mathcal{A} \in \mathcal{DP} | \big(\forall \mathcal{B} \in \text{NP}\big)\big(\mathcal{B} \leq_p \mathcal{A}\big)\}$.

Дефиниция 5.9: Класът NP-complete

$\text{NP-c} \leftrightharpoons \text{NP} \cap \text{NP-h}$.



Фигура 5.1: Ойлерова диаграма за класовете P, NP, NP-h и NP-c

Изобщо не е тривиално да се докаже, че ВСИЧКИ задачи за разпознаване от NP са Karp-сводими до дадена задача за разпознаване. Ако имаме такава задача и при това намерим детерминирано полиномиално нейно решение, то ще имаме детерминирано полиномиално решение на ВСИЧКИ задачи за разпознаване от NP, откъдето ще следва, че P=NP. Първата задача за разпознаване, за която е доказано, че е NP-hard е така наречената SAT (от satisfiability):

Задача за разпознаване 3: SAT

$$\begin{cases} \textbf{Instance: } \varphi \text{ - булева формула в КНФ} \\ \textbf{Question: } \text{Удволетворима ли е } \varphi? \end{cases}$$

Пример. $\begin{cases} \varphi = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_5} \vee x_8) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge x_6 \wedge x_7 \\ \text{да } //\text{сертификат: } \langle x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \rangle = \langle T, F, F, T, F, T, T, F \rangle \end{cases}$

Теорема 5.1: Cook-Levin

SAT е NP-трудна.

Доказателство. Накратко, само идеята. Всеки недетерминиран алгоритъм може да се представи чрез недетерминирана машина на тюринг (всъщност "истинската" дефиницията на NP използва недетерминирани МТ, вместо недетерминирани алгоритми). Тази машина бива представяна чрез булеви формули, т.е. състоянията ѝ, преходите ѝ, началните ѝ състояния, финалните ѝ състояния, лентата ѝ и т.н., като "връзките" между тях биват правени отново чрез булеви формули. Съответно тези формули са в КНФ. □

Следното твърдение ще ни е основния инструмент, с помощта на който ще доказваме, че дадена задача за разпознаване е NP-трудна. Съответно е инструмент, с който да доказваме, че не сме ние глупавите, а че самите задачи са с трудна природа (тоест никой човек досега не е успял да ги реши полиномиално).

Твърдение 5.1

Нека $\mathcal{A}, \mathcal{B} \in \mathcal{DP}$. Тогава ако $\mathcal{B} \in$ NP-h и $\mathcal{B} \leq_p \mathcal{A}$, то $\mathcal{A} \in$ NP-h.

Доказателство. Нека $\mathcal{C}$ е произволна от NP. Тъй като $\mathcal{B} \in$ NP-h, то $\mathcal{C} \leq_p \mathcal{B}$, но по условие имаме $\mathcal{B} \leq_p \mathcal{A}$. Сега от транзитивността имаме $\mathcal{C} \leq_p \mathcal{A}$. Но $\mathcal{C} \in$ NP беше произолно, следователно $\big(\forall \mathcal{C} \in \text{NP}\big)\big(\mathcal{C} \leq_p \mathcal{A}\big)$, т.е. $\mathcal{A} \in$ NP-h (по дефиниция). □