Сложност на алгоритми

доц. д-р. Нора Ангелова

Оценка на програма

- Времева сложност — оценка на време за изпълнение
- Пространствена (обемна) сложност — оценка на използвана памет

Асимптотична нотация

Как даден алгоритъм работи при достатъчно голям размер n на входните данни?

- **Дефиниция**: Формалното оценяване на сложността на алгоритъм при "достатъчно голямо" n,
  т.е. при $n \to \infty$.
- n – размер на входните данни.

Асимптотична нотация

- $\boldsymbol{O(F)}$ - определя множеството от всички функции f, които нарастват **не** по-бързо от F,
т.е. съществува константа $c > 0$ такава, че $f(n) <= cF(n)$.

- $\boldsymbol{\theta(F)}$ - определя множеството от всички функции f, които нарастват толкова бързо, колкото и F (с точност до константен множител),
т.е. съществуват константи $c_1 > 0$ и $c_2 > 0$ такава, че
$c_1F(n) <= f(n) <= c_2F(n)$.

- $\boldsymbol{\Omega(F)}$ - определя множеството от всички функции f, които нарастват **не** по-бавно от F,
т.е. съществува константа $c > 0$ такава, че $f(n) >= cF(n)$.

$n \to \infty$

Нотацията O(f)

- Използва се при оценка на сложност на алгоритми и програми.

- Определя времето за изпълнение на програмата като функция на обема *n* на входните данни.

Нотацията O(f)

** n е дължината на редицата, с която програмата работи.*

Определяне на ефективността О-голямо:

- Трябва да се определи колко пъти n-те елемента на редицата се „разглеждат“.
- Разглеждането може да бъде сравняване, създаване, изтриване на стойности и др.

Анализът О-голямо дава асимптотично време за изпълнение на алгоритъм – границата на времето за изпълнение, когато n расте неограничено.

Пример:

Време за изпълненение: $n + 5$ или $n * 5$
Анализът О определя времето за изпълнение като $O(n)$.
Казва се още, че времето за изпълнение е от порядъка на n.

Време за изпълнение: $n^2 + n \rightarrow$ Сложност $O(n^2)$

Нотацията O(f)

- Функции за оценка на сложност

c,

$\log n$,

n,

$n.\log n$,

n^2,

n^3,

2^n,

$n!$,

n^n

Свойства на O(f)

- Елементарна операция (не зависи от размера на входните данни) - $O(1)$;
- Рефлексивност: $f \in O(f)$;
- Транзитивност: ако $f \in O(g)$, $g \in O(h)$, то $f \in O(h)$;
- Транспонирана симетрия: ако $f \in \Omega(g)$, то $g \in O(f)$ и обратно;
- За всяко $k>0$, $k*f \in O(f)$;
- $n^r \in O(n^s)$, за $0 < r < s$;
- Нарастването на сума от функции:
  $f + g \in \max(O(f), O(g))$;
- Композиция на оператори - $f*g \in O(f*g)$;
- Условни оператори - определя се от асимптотично най-бавния между условието и различните случаи;
- Цикли, вложени цикли - $O(n)$, $O(n^p)$;

Пример

```
int n = 10;
int sum = 0;
for(int i=0; i<n; i++) {
    sum++;
}
```

- За инициализацията е необходимо константно време $a + b$
- За i = 0 времето е c
- За изпълнение на цикъла – $p * n$

Общо време = $a + b + c + p * n = p * n + q$,
където p, q са константи

Сложност: $O(n)$

Пример

```
int sum = 0;
for (int i=1; i<n; i*=2) {
    sum++;
}
```

i = 1, 2, 4, ..., 2^k, ... докато **i < n**.

$2^k = n$, $к = ?$

Цикълът се изпълнява [log *n*] пъти.

Сложност: $O(\log n)$

Двоично търсене (рекурсия)

- Броят на обръщенията към елементите на масива.
- Нека $T(n)$ е функцията, която задава броя на обръщенията.

Следователно:

$T(n) = T(n/2) + 1 = T(n/4) + 2 = T(n/2^k) + k$

При $2^k == n \rightarrow T(n) = T(1) + \log n$

Сложност: $O(\log n)$.

Недостатъци на Асимптотичната нотация

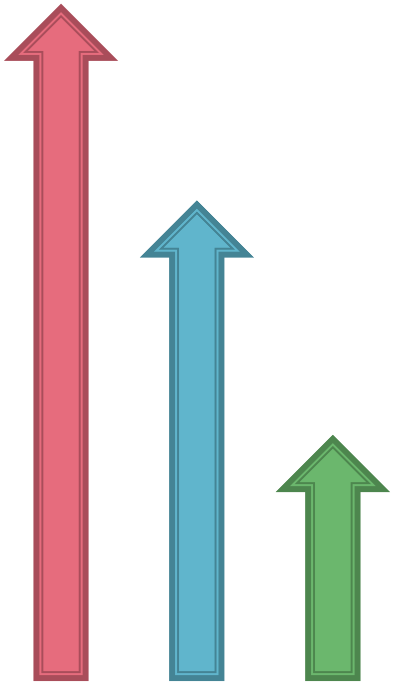
Нотацията се използва при $n \to \infty$.

Възможно е:

- Оценен алгоритъм да е по-бавен от алгоритъм с по-лоша оценка.
- Два алгоритъма с еднаква сложност да са различно бързи.

Асимптотичната нотация

- Оценки



Quick sort

Алгоритъм:

- Редицата се разделя на две подредици спрямо оста *x*.
- Елементите < x се записват в първата подредица.
- Елементите > x се записват във втората половина.
- Същото действие се повтаря за всяка от двете подредици.

Quick sort - сложност

- Нека S1 са елементите по-малки от оста.
- Нека S2 са елементите равни на оста.
- Нека S3 са елементите по-големи от оста.

В един по-реалистичен случай, в който повечето елементи са различни, изборът на произволна ос x ще генерира следните размери на трите части:
$S_1 \approx$ size / 2
$S_2 \approx 1$
$S_3 \approx$ size / 2

Всеки път се избира ос, елементите се разместват и размерът се намалява два пъти.

Сложност: O(*nlog(*n*))**

Quick sort - сложност

- Нека S1 са елементите по-малки от оста.
- Нека S2 са елементите равни на оста.
- Нека S3 са елементите по-големи от оста.

В най-лошия случай оста x е винаги или най-големият или най-малкият елемент.
Две от групите нямат елементи, а третата съдържа всички останали.

Сложност: O(n^2)

Следва продължение ...