Примерни решения на задачите от семестриалното контролно по ДАА на КН2 & Инф., 23 април 2023 г.

Зад. 1 Реална функция $f(n)$ се нарича *асимптотично положителна*, ако съществува $n_1 > 0$, такова че $f(n) > 0$ за всяко $n \geq n_1$.

1 т. Напишете дефиницията на "$\Theta(g(n))$".

9 т. Нека $f(n)$ и $g(n)$ са асимптотично положителни функции. Докажете, че $\max(f(n), g(n)) = \Theta(f(n) + g(n))$. Вашето доказателство не може да използва наготово никакви резултати от лекции или упражнения, а трябва да се базира на дефиницията на "Θ" и на неща, които ще докажете тук.

Решение: Дефиницията на "$\Theta(g(n))$" е казана на лекции. Доказателството е в сборника с решени задачи, както и в лекционните записки.

Зад. 2 Разгледайте следните функции, написани на `C`.

```
1  int bar(int n) {
2    return (int)sqrt((double)n*n);
3  }
4
5  int foo(int x, int y) {
6    return (( x + y + bar(x-y) ) / 2);
7  }
8
9  int quux(int *array, int length) {
10   if (length == 1) {
11     return array[0];
12   }
13   else if (length == 2) {
14     return foo(array[0], array[1]);
15   }
16   else return foo(quux(array, length-1), array[length-1]);
17 }
```

2 т. Какво връща `quux(A, n)`, където `A` е непразен масив от цели числа, а `n` е броят на неговите елементи?

18 т. Докажете това формално и прецизно.

Решение: `quux(A, n)` връща максималния елемент на A. Сега ще докажем това прецизно.

Първо, забелязваме, че `bar(n)` връща $\sqrt{n^2}$, което е $|n|$:

- ако $n \geq 0$, то $\sqrt{n^2} = n$,
- ако $n < 0$, то $\sqrt{n^2} = -n$.

Разглеждаме `foo(x,y)`. Щом `bar(n)` $=$ $|n|$, вярно е, че `foo(x,y)` $= \frac{x+y+|x-y|}{2}$. Тогава `foo(x,y)` връща $\max\{x, y\}$, тъй като:

- ако $x \geq y$, то $|x - y| = x - y$, така че `foo(x,y)` $= \frac{x+y+x-y}{2} = x$.
- ако $x < y$, то $|x - y| = y - x$, така че `foo(x,y)` $= \frac{x+y+y-x}{2} = y$.

Ще докажем, че за всяко length ≥ 1, `quux(array, length)` връща максималния елемент на масива $\text{array}[1 \mathinner{..} \text{length} - 1]$. Доказателството е със силна индукция по големината на масива length.

Ако length $= 0$, условието на ред 11 е истина и на ред 12 функцията връща първия елемент, който тривиално е максималният елемент.

Ако length $= 1$, изпълнението отива на ред 13, където условието е истина и функцията връща `foo(x,y)`, където x и y са двата елемента на масива. Както вече се убедихме, `foo(x,y)` е максимумът на x и y.

Ако length > 1, изпълнението достига ред 16. Допускаме, че $\text{quux}(\text{array}, \text{length}-1)$ връща максимума на $\text{array}[1 \mathinner{..} \text{length} - 2]$. Тогава функцията на ред 16 връща

$$\max\{\max(\text{array}[1 \mathinner{..} \text{length} - 2]), \text{array}[\text{length} - 1]\}$$

Но очевидно

$$\max\{\max(\text{array}[1 \mathinner{..} \text{length} - 2]), \text{array}[\text{length} - 1]\} = \max(\text{array}[1 \mathinner{..} \text{length} - 1])$$

Следователно, на ред 16 функцията връща $\max(\text{array}[1 \mathinner{..} \text{length} - 1])$.

Зад. 3 Дадени са n работници $h_1, \ldots, h_n$, които работят в една и съща фирма. Фирмата им е изплатила някакви великденски бонуси. При изчисляването на бонусите са станали грешки и **всички** работници са получили неправилни бонуси и сега нещата трябва да се коригират. За $i \in \{1, \ldots, n\}$, паричната корекция за h_i е $c_i \in \mathbb{Z} \setminus \{0\}$, като, ако $c_i > 0$, фирмата трябва да доплати c_i лева на h_i, а ако $c_i < 0$, то h_i трябва да върне c_i лева на фирмата.

Възниква въпросът дали има двама души, такива че корекцията, която трябва да бъде платена на единия е точно равна (като абсолютна стойност) на корекцията, която другият трябва да върне.

15 т. Предложете колкото е възможно по-ефикасен алгоритъм, който по дадени n и $c_1, \ldots, c_n$ дава отговор на този въпрос. Напишете подробен псевдокод. Допустимо е да използвате викания на алгоритми, които не са предназначени да дават отговор на тази задача, но са изучавани на лекции и помагат за конструиране на ефикасно решение, без да давате код или верифицирате или изследвате тези помощни алгоритми.

15 т. Докажете формално и прецизно коректността на Вашия алгоритъм и изследвайте сложността му по време и по памет.

Решение: На лекции разгледахме следния алгоритъм.

```
ALG2SUM(A[1..n]: масив от цели числа, такъв че A[1] ≤ A[2] ≤ ··· ≤ A[n]; m ∈ Z)
1  i ← 1, j ← n
2  while i < j do
3      if A[i] + A[j] = m
4          return 1
5      else if A[i] + A[j] < m
6          i++
7      else
8          j--
9  return 0
```

Този алгоритъм почти решава задачата. Трябва да се направят следните дребни модификации. Първо, аргумент m няма – сега $m = 0$. Второ, ако корекциите са представени с масив $c[1..n]$, не е казано, че масивът е сортиран, така че трябва ние да го сортираме. По условие е допустимо да се вика сортиращ алгоритъм, без да се описва кода му и без да се доказват свойствата му. Ето тази модифицирана версия решава задачата.

```
ALG2SUMMOD(c[1..n]: масив от цели числа)
1  HEAPSORT(c)
2  i ← 1, j ← n
3  while i < j do
4      if A[i] + A[j] = m
5          return 1
6      else if A[i] + A[j] < m
7          i++
8      else
9          j--
10 return 0
```

Алгоритъмът връща 1, ако има два различни елемента, сумиращи се до нула, или 0, ако няма такива. Коректността доказваме с инвариант като този на лекции "Всеки път, когато изпълнението е на ред 3, всички диади в $c[1..n]$ се намират в подмасива $c[i..j]$". Доказателството, включително и терминацията, практически повтаря доказателството от лекции.

Сложността по време е $\Theta(n \lg n)$ за предварителното сортиране и $\Theta(n)$ за изпълнението на **while**-цикъла. Общо е $\Theta(n \lg n)$. Сложността по памет е $\Theta(1)$, ако HEAPSORT ползва HEAPIFY, която има сложност по памет $\Theta(1)$.

Зад. 4 Дадено е множество $A = \{a_1, a_2, \dots, a_n\}$ от цели числа. Дадено е и $k \in \{2, \dots, n\}$. За удобство допуснете, че k дели $n - (k - 1)$. Нека $m = \frac{n-k+1}{k}$. Дефинираме, че k-*квантилите на* A са тези $k - 1$ елементи, за които множеството от **останалите** елементи се разбива на k множества, да ги наречем $S_1, \dots, S_k$, такива че $|S_1| = |S_2| = \dots = |S_k| = m$ и всеки елемент на S_1 е по-малък от всеки елемент на S_2, всеки елемент на S_2 е по-малък от всеки елемента на S_3, ..., всеки елемент на S_{k-1} е по-малък от всеки елемент на S_k.

1 т. Намерете 2-квантилите на $\{4, 14, -5, 0, 12, -333, 8, 44, 151, 6, 16\}$. За кое понятие, споменавано много пъти на лекции, става дума?

1 т. Намерете 4-квантилите, които още се наричат *квартили*, на $\{56, 1, -7, -44, 21, 31, -3, -15, 11, 12, 41, -99, 67, 99, 18\}$.

1 т. Предложете алгоритъм със сложност $O(n \lg n)$, който намира k-квантилите на A.

22 т. Предложете алгоритъм със сложност $O(n \lg k)$, който намира k-квантилите на A. Няма нужда от псевдокод. Достатъчно е да опишете идеята на алгоритъма, но ясно и недвусмислено. Няма нужда от доказателство за коректност. Съвсем накратко обосновете асимптотичната горна граница върху сложността му по време.

Забележка: имайте предвид, че става дума за множество, така че числата в него са две по две различни.

Решение: "2-квантилите" са само един елемент, а именно медианата. Медианата на $\{4, 14, -5, 0, 12, -333, 8, 44, 151, 6, 16\}$ е 8.

Квартилите на $\{56, 1, -7, -44, 21, 31, -3, -15, 11, 12, 41, -99, 67, 99, 18\}$ са -7, 12 и 41:

$$-99, -44, -15, -7, -3, 1, 11, 12, 18, 21, 31, 41, 56, 67, 99$$

Да се намерят квантилите на сортиран масив е тривиално. Ето така:

Alg1($A[1, \dots, n]$: цели уникални числа, $k \in \{2, \dots, n\}$)
(∗ k дели $n - (k - 1)$ ∗)
Heapsort(A)
$m \leftarrow \frac{n-k+1}{k}$
quantiles $\leftarrow \emptyset$
for $i \leftarrow 1$ **to** $k - 1$
    quantiles $\leftarrow$ quantiles $\cup \{A[i(m + 1)]\}$
return quantiles

Сложността е $\Theta(n \lg n)$ заради сортирането.

Сега ще опишем алгоритъм, намиращ квантилите във време $O(n \lg k)$.

- Изчисляваме позицията на средния квантил. Дефинираме, че "средният квантил" е $\left\lfloor \frac{k}{2} \right\rfloor$-ия по големина квантил. Примерно, ако $k = 10$, то квантилите са 9 на брой и петият по големина от тях е средният квантил. Очевидно $\left\lfloor \frac{k}{2} \right\rfloor$-ия квантил би бил в клетка номер $\left\lfloor \frac{k}{2} \right\rfloor (m + 1)$, ако масивът беше сортиран.

  Нека $pp \leftarrow \left\lfloor \frac{k}{2} \right\rfloor (m + 1)$. "pp" идва от **p**ivot **p**osition, понеже $\left\lfloor \frac{k}{2} \right\rfloor$-ият квантил ще бъде pivot. Изпълняваме pivot $\leftarrow$ Select($A, pp - 1$), където Select е алгоритъмът, изучаван на лекции, за намиране на елемент по големина. Както знаем, Select($A, pp - 1$) връща елемента, който би бил на позиция pp, ако A беше сортиран (с други думи, преди който има точно $pp - 1$ елемента по големина), като алгоритъмът работи във време $O(n)$.

- Изпълняваме Partition(A, pivot), където Partition е във версията, в която е описан в подсекцията за алгоритъма Select. А именно, той първо намира къде се намира pivot в масива, разменя го с най-десния елемент и изпълнява Lomuto-Partition.

Ефектът от това е, че $\left\lfloor \frac{k}{2} \right\rfloor$-ия квантил се оказва на мястото си, а именно на позиция pp, вляво от него са точно елементите, по-малки от него, а вдясно са точно елементите, по-големи от него.

Функцията Partition връща индекс-позицията на pivot, който индекс сега игнорираме, понеже той вече е известен.

- Изпълняваме рекурсивно същото върху подмасивите $\mathsf{A}[1\mathinner{..}\mathsf{pp}-1]$ и $\mathsf{A}[\mathsf{pp}+1\mathinner{..}\mathsf{n}]$, откъдето получаваме списъци от квантили съответно L и R. Крайният отговор е списъкът $\mathsf{L}, \mathrm{pivot}, \mathsf{R}$.

Доказателството за коректност може да се направи със силна индукция по големината на входа. Изпускайки детайлите: при допускането, че двете рекурсивни викания връщат коректни списъци от квантили L и R, тривиално се показва, че $\mathsf{L}, \mathrm{pivot}, \mathsf{R}$ е списъкът от квантилите на $\mathsf{A}[1\mathinner{..}\mathsf{n}]$.

Асимптотична горна граница за сложността по време даваме със следните прости съображения: дървото на рекурсията има височина $\lceil \lg k \rceil$, понеже и двата подмасива, върху които викаме, съдържат не повече от $\left\lfloor \frac{k}{2} \right\rfloor$ квантила на входа. На всяко ниво на рекурсията се върши само линейна работа. Ерго, нивáта са $\Theta(\lg k)$ и във всяко от тях се върши общо $\mathsf{O}(\mathsf{n})$ работа, откъдето е и горната граница $\mathsf{O}(\mathsf{n} \lg k)$ за сложността по време на алгоритъма.

Зад. 5 Обяснете какво означава “топологическо сортиране”, напишете псевдокода на някой (един е достатъчен) от алгоритмите за топологическо сортиране, които знаете, съвсем накратко обосновете коректността му (няма нужда от формално доказателство) и изследвайте сложността му по време.

Решение: Това е правено на лекции.

Зад. 6 Решете следните шест рекурентни уравнения.

$$A(n) = 5A\left(\frac{n}{7}\right) + 12n \qquad B(n) = \sqrt{5}B\left(\frac{n}{\sqrt{7}}\right) + \sqrt{13n}$$

$$C(n) = 5C\left(\frac{n}{5}\right) + n(\lg n)^5 \qquad D(n) = 2D\left(\frac{n}{2}\right) + n2^n$$

$$S(n) = S(n-1) + 2n^2 + 2^{3n} + n8^{n+2} + S(n-1) \qquad T(n) = 41T(n-2) + 72T(n-3) - 112T(n-4) + n^6 6^n$$

Решение: Разглеждаме $A(n) = 5A\left(\frac{n}{7}\right) + 12n$. Ще го решим с МТ, понеже то е от подходящия вид. В терминологията на МТ, $a = 5$, $b = 7$ и $f(n) = 12n$. Тогава $n^{\log_b a} = n^{\log_7 5}$. $\log_7 5 \approx 0.8270874751$, но това няма значение. Важното е, че $0 < \log_7 5 < 1$. Тогава е вярно, че $12n = \Omega\left(n^{\log_7 5} \cdot n^{\epsilon}\right)$ за някое положително ϵ. Ако и условието за регулярност е в сила, третият случай на МТ е приложим.

Да разгледаме условието за регулярност. Дали съществува константа c, такава че $0 < c < 1$ и $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ за всички достатъчно големи n? Неравенството става

$$5 \cdot 12\left(\frac{n}{7}\right) \leq c12n \;\leftrightarrow\; \frac{5}{7} \leq c$$

Тогава всяко c, такова че $\frac{5}{7} \leq c < 1$, "върши работа" за условието за регулярност. Условието за регулярност е приложимо. Съгласно третият случай на МТ, $A(n) \asymp n$.

Разглеждаме $B(n) = \sqrt{5}B\left(\frac{n}{\sqrt{7}}\right) + \sqrt{13n}$. Уравнението е решимо с МТ. Имаме $a = \sqrt{5}$, $b = \sqrt{7}$ и $f(n) = (13n)^{\frac{1}{2}}$. Тогава $\log_b a = \log_{\sqrt{7}} \sqrt{5}$. Но $\log_{\sqrt{7}} \sqrt{5} = \log_7 5$, така че сравняваме $n^{\log_7 5}$ с $(13n)^{\frac{1}{2}}$. Ще докажем, че $\log_7 5 > \frac{1}{2}$. Наистина,

$$\log_7 5 > \frac{1}{2} \;\leftrightarrow\; \frac{\lg 5}{\lg 7} > \frac{1}{2} \;\leftrightarrow\; 2\lg 5 > \lg 7 \;\leftrightarrow\; \lg 25 > \lg 7, \text{ което е очевидно}$$

Щом $\log_7 5 > \frac{1}{2}$, съществува положително ϵ, такова че

$$\sqrt{13n} = O\left(\frac{n^{\log_7 5}}{n^{\epsilon}}\right)$$

Съгласно първия случай на МТ, $B(n) \asymp n^{\log_7 5}$.

Разглеждаме $C(n) = 5C\left(\frac{n}{5}\right) + n(\lg n)^5$. Уравнението е решимо с разширението на МТ, разгледано на лекции. Имаме $a = 5$, $b = 5$ и $f(n) = n(\lg n)^5$. Тогава $\log_b a = \log_5 5 = 1$, така че $n^{\log_b a} = n^1$. Съгласно разширението на МТ, $C(n) = n(\lg n)^6$.

Разглеждаме $D(n) = 2D\left(\frac{n}{2}\right) + n2^n$. Уравнението е решимо с МТ. Имаме $a = 2$ и $b = 2$, откъдето $n^{\log_b a} = n^1$. А $f(n) = n2^n$. Ако и условието за регулярност е в сила, третият случай на МТ е приложим.

Да разгледаме условието за регулярност. Дали съществува константа c, такава че $0 < c < 1$ и $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ за всички достатъчно големи n? Неравенството става

$$2 \cdot \left(\frac{n}{2} 2^{\frac{n}{2}}\right) \leq cn2^n \;\leftrightarrow\; n\sqrt{2}^n \leq cn2^n \;\leftrightarrow\; \sqrt{2}^n \leq c2^n \;\leftrightarrow\; c \geq \left(\frac{1}{\sqrt{2}}\right)^n$$

Всяка положителна константа c "върши работа". Съгласно третия случай на МТ, $D(n) \asymp n2^n$.

Разглеждаме $S(n) = S(n-1) + 2n^2 + 2^{3n} + n8^{n+2} + S(n-1)$. Преписваме рекурентното уравнението във вид, подходящ за решаване чрез метода с характеристичното уравнение:

$$S(n) = 2S(n-1) + 2n^2 \cdot 1^n + (64n + 1) \cdot 8^n$$

Съответното хомогенно уравнение е

$$S(n) = 2S(n-1)$$

с характеристично уравнение $x - 2 = 0$ и мултимножество от корените $\{2\}_M$. Нехомогенната част е $2n^2 \cdot 1^n + (64n + 1) \cdot 8^n$. Тя дава мултимножество $\{1, 1, 1, 8, 8\}_M$. Тогава $\{2\}_M \cup \{1, 1, 1, 8, 8\}_M = \{1, 1, 1, 2, 8, 8\}_M$. Най-големият корен е 8 с кратност две. Оттук решението е $S(n) \asymp n8^n$.

Разглеждаме $T(n) = 41T(n-2) + 72T(n-3) - 112T(n-4) + n^6 6^n$. Съответното хомогенно уравнение е

$$T(n) = 41T(n-2) + 72T(n-3) - 112T(n-4)$$

Съответното характеристично уравнение е $x^4 - 41x^2 - 72x + 112 = 0$. Разглеждаме полинома $x^4 - 41x^2 - 72x + 112$. С метода на Хорнер установяваме, че единицата е корен:

$$\begin{array}{c|ccccc}
 & 1 & 0 & -41 & -72 & 112 \\
1 & \downarrow & 1 & 1 & -40 & -112 \\
\hline
 & 1 & 1 & -40 & -112 & \multicolumn{1}{|r}{0}
\end{array}$$

Разглеждаме полинома $x^3 + x^2 - 40x - 112$. С метода на Хорнер установяваме, че седмицата е корен:

$$\begin{array}{c|cccc}
 & 1 & 1 & -40 & -112 \\
7 & \downarrow & 7 & 56 & 112 \\
\hline
 & 1 & 8 & 16 & \multicolumn{1}{|r}{0}
\end{array}$$

Тогава $x^4 - 41x^2 - 72x + 112 = (x-1)(x-7)(x^2 + 8x + 16)$. Но $x^2 + 8x + 16 = (x+4)^2$. Тогава $x^4 - 41x^2 - 72x + 112 = (x-1)(x-7)(x+4)^2$. Тогава мултимножеството от корените на характеристичното уравнение е $\{1, -4, -4, 7\}_M$. Нехомогенната част е $n^6 6^n$. Тя дава мултимножество $\{6, 6, 6, 6, 6, 6, 6\}_M$. Тогава

$$\{1, -4, -4, 7\}_M \cup \{6, 6, 6, 6, 6, 6, 6\}_M = \{1, -4, -4, 6, 6, 6, 6, 6, 6, 6, 7\}_M$$

Най-големият корен е 7 с кратност едно. Решението е $T(n) \asymp 7^n$. Забележете, че отрицателните корени са по-малки от 7 по абсолютна стойност и наличието им не променя факта, че $T(n)$ е положителна функция при подходящи начални условия.