Tema – Analiza algoritmilor

Alin Pisica, 324 CC

I. Clase de complexitate

1. Verificati valoarea de adevar a relatiilor, argumentand pentru fiecare raspunsul

- $0.01 \cdot n \cdot \log n - 2000 \cdot n + 6 = O(n \cdot \log n)$

$O(g(n)) = \{f : \mathbb{N}^* \to \mathbb{R}_+^* \mid \exists c \in \mathbb{R}_+^*, \exists n_0 \in \mathbb{N}^* a.i. f(n) \le c \cdot g(n), \forall\, n \ge n_0\}$

$\exists c \in \mathbb{R}_+^*\ a.i. 0.01 \cdot n \cdot \log n - 2000 \cdot n + 6 = c \cdot n \cdot \log n$
Fie $c = 1 \to 0.01 \cdot n \cdot \log n - 2000 \cdot n + 6 \le n \cdot \log n \Rightarrow -0.99 \cdot n \cdot \log n \le 2000 \cdot n - 6$
Pentru $n_0 = 1$, relatia devine adevarata, $n_0 = 1 \in \mathbb{N}^* \Rightarrow$
$0.01 \cdot n \cdot \log n\ - 2000 \cdot n + 6 = O(n \cdot \log n)$, adevarat pentru $\forall n \ge n_0$, unde $n_0 = 1 \in \mathbb{N}^*$.
Verificare pentru $n_0 = 1$ si $c = 1$:
$0.01 \cdot 1 \cdot \log 1 - 2000 \cdot 1 + 6 \le 1 \cdot 1 \cdot \log 1 \Rightarrow 0.01 \cdot 0 - 2000 + 6 < 0 \Rightarrow 6 < 2000, Adev$

- $\Theta(n \cdot \log n) \cup o(n \cdot \log n) \ne O(n \cdot \log n)$

$\Theta(g(n)) = \{f : \mathbb{N}^* \to \mathbb{R}_+^* \mid \exists c_1, c_2 \in \mathbb{R}_+^*, \exists n_0 \in \mathbb{N}^* a.i. c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n), \forall\, n \ge n_0\}$
$o(g(n)) = \{f : \mathbb{N}^* \to \mathbb{R}_+^* \mid \forall c \in \mathbb{R}_+^*, \exists n_0 \in \mathbb{N}^* a.i. f(n) < c \cdot g(n), \forall\, n \ge n_0\}$
$O(g(n)) = \{f : \mathbb{N}^* \to \mathbb{R}_+^* \mid \exists c \in \mathbb{R}_+^*, \exists n_0 \in \mathbb{N}^* a.i. f(n) \le c \cdot g(n), \forall\, n \ge n_0\}$

Din propozitia 1 din curs, stim ca $o(g(n)) \le O(g(n))$. (1)

$o(g(n)) = \{f : \mathbb{N}^* \to \mathbb{R}_+^* \mid \forall c \in \mathbb{R}_+^*, \exists n_0 \in \mathbb{N}^* a.i. f(n) < c \cdot g(n), \forall\, n \ge n_0\}$
$f(n) < c \cdot g(n)$
$O(g(n)) = \{f : \mathbb{N}^* \to \mathbb{R}_+^* \mid \exists c \in \mathbb{R}_+^*, \exists n_0 \in \mathbb{N}^* a.i. f(n) \le c \cdot g(n), \forall\, n \ge n_0\}$
$f(n) \le c \cdot g(n)$

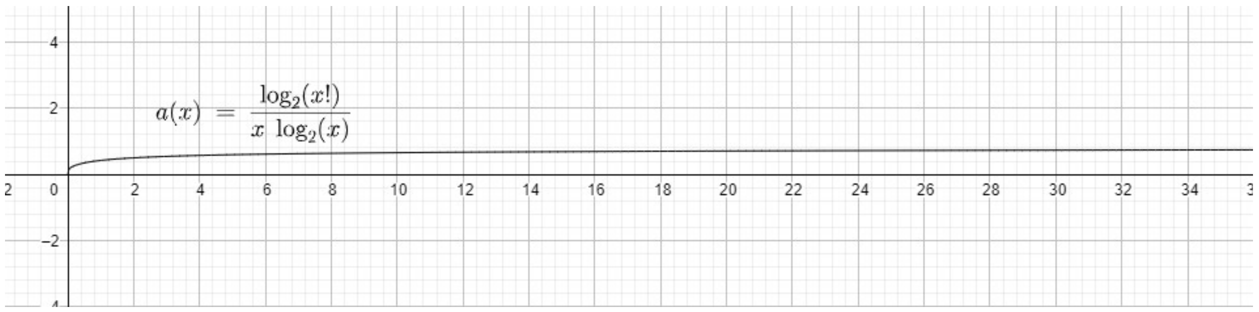
$\Theta(g(n)) = \{f : \mathbb{N}_+^* \to \mathbb{R}_+^* | \exists c_1, c_2 \in \mathbb{R}_+^*, \exists n_0 \in \mathbb{N}_+^*\ a.i. c_1 \cdot g(n) \le f(n) \le\ c_2 \cdot g(n), \forall n \ge n_0\}$

Fie $c_1, c_2 \in \mathbb{R}_+^*$, unde $c_2 > c_1$. Presupunand $f(n) = (n \cdot \log n) \to din\, \Theta(n \cdot logn)$ obtinem $\exists c_1, c_2 \in \mathbb{R}_+^*\ a.i. c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n) \to c_1 \cdot n \cdot \log n \le n \cdot \log n \le c_2 \cdot n \cdot \log n$. Pentru $c_1 < 1\ si\ c_2 > 1$, relatia anterioara ramane adevarata, insa vor aparea elemente in plus, fata de cele ce se obtin din $o(n \cdot \log n)$, din curs (1) stiind ca $o(g(n)) < O(g(n))$. Astfel, vom ajunge la valoarea de adevar a relatiei $\Theta(n \cdot \log n) \cup o(n \cdot \log n) \ne O(n \cdot \log n)$

- $\sqrt{n} \cdot \log n! = \Omega(n^{\frac{3}{2}} \cdot \log n)$

$Fie\ f(n) = \sqrt{n} \cdot log(n!)\ si\ g(n) =\ n^{\frac{3}{2}} \cdot \log\, n.\ Verificam\ prin\ trecerea\ la\ limita.$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sqrt{n} \cdot log(n!)}{n^{\frac{3}{2}} \cdot \log\, n} = \lim_{n\to\infty} \frac{\sqrt{n} \cdot \log(n!)}{n \cdot \sqrt{n} \cdot \log n} = \lim_{n\to\infty} \frac{\log 1 + \log 2 + \cdots + \log n}{n \cdot \log n} > 0, \forall n \in \mathbb{N}^*.$$



- $n! = \Omega(5^{\log n})$

$Fie\ f(n) = n!\ si\ g(n) = 5^{\log n}.\ Verificam\ prin\ trecerea\ la\ limita.$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{n!}{5^{\log n}} \xRightarrow{a^{\log_b c}=c^{\log_b a}} \lim_{n\to\infty}\frac{n!}{n^{\log 5}} \xRightarrow{pentru\ baza\ 2, \log 5>1} +\infty > 0. =>$$

$n! = \Omega\left(5^{\log n}\right)\ adevarat$

- $log^{2019}n = o(n)$

2. ***Calculati estimativ, oferind explicatii, complexitatea temporala pentru urmatorii algoritmi, pentru cele trei cazuri (favorabil, defavorabil si mediu):***

OBSERVATIE: Am facut abstractie de incadrarea fractiilor intre paranteze patrate in cazul in care este necesara trecerea la parte intreaga, pentru a face mai lizibil codul si a nu il incarca de paranteze.

Algoritm1(A[0..n], n)

```
for i = 1..n                          $c_1$     n + 1
    if (A[i] > 0)                     $c_2$     $\sum_{i=1}^{n} 1$
        for j = 1..i                  $c_3$     $\sum_{i=1}^{n}(t_i + 1)$
            if  (A[j] mod 2 == 0)     $c_4$     $\sum_{i=1}^{n} t_i$
                for k = 1..j          $c_5$     $\sum_1^n \sum_1^{t_i}(t_j + 1)$
                    s = s + i + j + k $c_6$     $\sum_1^n \sum_1^{t_i} t_j$
return s
```

Nu vom lua in considerare ca si operatie *return s* deoarece se executa in O(1) si nu afecteaza complexitatea programului.

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot \sum_{i=1}^{n} 1 + c_3 \cdot \sum_{i=1}^{n}(t_i + 1) + c_4 \cdot \sum_{i=1}^{n} t_i + c_5 \cdot \sum_1^n \sum_1^{t_i}\left(t_j + 1\right) + c_6 \cdot \sum_1^n \sum_1^{t_i} t_j$$

a) In cazul cel mai favorabil, toate elementele vectorului A vor fi negative, astfel costurile c_{3-6} nu vor fi atinse niciodata.
   $t_i = 0, t_j = 0$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot \sum_{i=1}^{n} 1 = c_1 \cdot (n+1) + c_2 \cdot n = n \cdot (c_1 + c_2) + c_1 = O(n)$$

b) In cazul cel mai defavorabil, toate numerele sunt pozitive (pentru a trece de primul if-statement) si pare (pentru a trece de al doilea if-statement).

$t_i = i, t_j = j$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot \sum_{i=1}^{n} 1 + c_3 \cdot \sum_{i=1}^{n} (i+1) + c_4 \cdot \sum_{i=1}^{n} i + c_5 \cdot \sum_{1}^{n} \sum_{1}^{i} (j+1) + c_6 \cdot \sum_{1}^{n} \sum_{1}^{i} j$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot n + c_3 \cdot (\frac{n \cdot (n+1)}{2} + n) + c_4 \cdot n + c_5 \cdot \sum_{1}^{n} (\frac{i \cdot (i+1)}{2} + i) + c_6 \cdot \sum_{1}^{n} \frac{i \cdot (i+1)}{2}$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot n + c_3 \cdot \frac{n^2 + 3n}{2} + c_4 \cdot n + c_5 \cdot \sum_{1}^{n} \frac{i^2 + 3i}{2} + c_6 \cdot \sum_{1}^{n} \frac{i^2 + i}{2}$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot n + c_3 \cdot \frac{n^2 + 3n}{2} + c_4 \cdot n + c_5 \cdot \frac{\sum_1^n i^2 + 3\sum_1^n i}{2} + c_6 \cdot \frac{\sum_1^n i^2 + \sum_1^n i}{2}$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot n + c_3 \cdot \frac{n^2 + 3n}{2} + c_4 \cdot n + c_5 \cdot \frac{\frac{n(n+1)(2n+1)}{6} + 3\frac{n(n+1)}{2}}{2} + c_6 \cdot \frac{\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}}{2}$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot n + c_3 \cdot \frac{n^2 + 3n}{2} + c_4 \cdot n + c_5 \cdot \frac{n(n+1)(2n+10)}{12} + c_6 \cdot \frac{n(n+1)(2n+4)}{12}$$

$$T(n) = \frac{1}{12}[12c_1(n+1) + 12c_2 n + 6c_3(n^2 + 3n) + 12c_4 n + c_5 n(n+1)(2n+10) + c_6 n(n+1)(2n+4)]$$

$$T(n) = \frac{1}{12}[12c_1(n+1) + 12c_2 n + 6c_3(n^2 + 3n) + 12c_4 n + c_5 n(n+1)(2n+10) + c_6 n(n+1)(2n+4)]$$

$$T(n) = \frac{1}{12}(12c_1 n + 12c_1 + 12c_2 n + 6c_3 n^2 + 18c_3 n + 12c_4 n + 2c_5 n^3 + 12c_5 n^2 + 10c_5 n + 2n^3 c_6 + 6n^2 c_6 + 4nc_6)$$

$$T(n) = \frac{1}{12}[n^3(2c_5 + 2c_6) + n^2(6c_3 + 12c_5 + 6c_6) + n(12c_1 + 12c_2 + 18c_3 + 12c_4 + 10c_5 + 4c_6) + 12c_1]$$

$$T(n) = n^3 \frac{c_5 + c_6}{6} + n^2 \frac{c_3 + 2c_5 + c_6}{2} + n \frac{6c_1 + 6c_2 + 9c_3 + 6c_4 + 5c_5 + 2c_6}{6} + 12c_1$$

$$T(n) = O(n^3)$$

c) Cazul mediu

$$t_i = \frac{i+0}{2} = \frac{i}{2}, t_j = \frac{j+0}{2} = \frac{j}{2}$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot \sum_{i=1}^{n} 1 + c_3 \cdot \sum_{i=1}^{n} \left(\frac{i}{2} + 1\right) + c_4 \cdot \sum_{i=1}^{n} \frac{i}{2} + c_5 \cdot \sum_{i=1}^{n} \sum_{j=1}^{\frac{i}{2}} \left(\frac{\frac{j}{2}}{2} + 1\right) + c_6 \cdot \sum_{i=1}^{n} \sum_{1}^{\frac{i}{2}} \frac{j}{2}$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot \sum_{i=1}^{n} 1 + c_3 \cdot \sum_{i=1}^{n} \left(\frac{i}{2} + 1\right) + c_4 \cdot \sum_{i=1}^{n} \frac{i}{2} + c_5 \cdot \sum_{i=1}^{n} \sum_{j=1}^{\frac{i}{2}} \left(\frac{j}{2} + 1\right) + c_6 \cdot \sum_{i=1}^{n} \sum_{j=1}^{\frac{i}{2}} \frac{j}{2}$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot \frac{n(n+1)}{2} + c_3 \cdot \left(\frac{1}{2} \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1\right) + \frac{1}{2} c_4 \cdot \sum_{i=1}^{n} i + c_5 \cdot \sum_{i=1}^{n} \left(\frac{1}{2} \sum_{j=1}^{\frac{i}{2}} j + \sum_{j=1}^{\frac{i}{2}} 1\right) + c_6 \cdot \sum_{i=1}^{n} \left(\frac{1}{2} \sum_{j=1}^{\frac{i}{2}} j\right)$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot \frac{n(n+1)}{2} + c_3 \cdot \left(\frac{n(n+1)}{4} + n\right) + \frac{1}{2} c_4 \cdot \frac{n(n+1)}{2} + c_5 \cdot \sum_{i=1}^{n} \left(\frac{1}{2} \frac{i(i+2)}{8} + \frac{i}{2}\right) + c_6 \cdot \sum_{i=1}^{n} \left(\frac{1}{2} \frac{i(i+2)}{8}\right)$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot \frac{n(n+1)}{2} + c_3 \cdot \left(\frac{n(n+1)}{4} + n\right) + \frac{1}{2} c_4 \cdot \frac{n(n+1)}{2} + c_5 \cdot \sum_{i=1}^{n} \left(\frac{1}{2} \frac{i(i+2)}{8} + \frac{i}{2}\right) + c_6 \cdot \sum_{i=1}^{n} \left(\frac{1}{2} \frac{i(i+2)}{8}\right)$$

$$T(n) = c_1 \cdot (n+1) + c_2 \cdot \frac{n(n+1)}{2} + c_3 \cdot \frac{n(n+5)}{4} + c_4 \cdot \frac{n(n+1)}{4} + c_5 \cdot \sum_{i=1}^{n} \frac{i(i+10)}{16} + c_6 \cdot \sum_{i=1}^{n} \frac{i(i+2)}{16}$$

$$T(n) = c_1 \cdot (n+1) + n(n+1) \cdot \frac{c_2}{2} + n(n+5) \cdot \frac{c_3}{4} + n(n+1) \cdot \frac{c_4}{4} + \frac{c_5}{16} \cdot \sum_{i=1}^{n} (i^2 + 10i) + \frac{c_6}{16} \cdot \sum_{i=1}^{n} (i^2 + 2i)$$

$$T(n) = c_1 \cdot (n+1) + n(n+1) \cdot \frac{c_2}{2} + n(n+5) \cdot \frac{c_3}{4} + n(n+1) \cdot \frac{c_4}{4} + \frac{c_5}{16} \cdot \left(\sum_{i=1}^{n} i^2 + 10\sum_{i=1}^{n} i\right) + \frac{c_6}{16} \cdot \left(\sum_{i=1}^{n} i^2 + 2\sum_{i=1}^{n} i\right)$$

$$T(n) = c_1 \cdot (n+1) + n(n+1) \cdot \frac{c_2}{2} + n(n+5) \cdot \frac{c_3}{4} + n(n+1) \cdot \frac{c_4}{4} + \frac{c_5}{16} \cdot \left(\frac{n(n+1)(2n+1)}{6} + 10\frac{n(n+1)}{2}\right) + \frac{c_6}{16} \cdot \left(\frac{n(n+1)(2n+1)}{6} + 2\frac{n(n+1)}{2}\right)$$

$$T(n) = c_1 \cdot (n+1) + n(n+1) \cdot \frac{c_2}{2} + n(n+5) \cdot \frac{c_3}{4} + n(n+1) \cdot \frac{c_4}{4} + \frac{c_5}{16} \cdot \frac{n(n+1)(2n+1) + 60n(n+1)}{6} + \frac{c_6}{16} \cdot \frac{n(n+1)(2n+1) + 6n(n+1)}{6}$$

$$T(n) = c_1 \cdot (n+1) + n(n+1) \cdot \frac{c_2}{2} + n(n+5) \cdot \frac{c_3}{4} + n(n+1) \cdot \frac{c_4}{4} + \frac{c_5}{16} \cdot \frac{n(n+1)(2n+61)}{6} + \frac{c_6}{16} \cdot \frac{n(n+1)(2n+7)}{6}$$

$$T(n) = c_1 \cdot (n+1) + n(n+1) \cdot \frac{c_2}{2} + n(n+5) \cdot \frac{c_3}{4} + n(n+1) \cdot \frac{c_4}{4} + \frac{c_5}{16} \cdot \frac{2n^3 + 63n^2 + 61n}{6} + \frac{c_6}{16} \cdot \frac{2n^3 + 9n^2 + 7n}{6}$$

$$T(n) = n^3 \frac{c_5 + c_6}{48} + n^2 \left(\frac{c_2}{2} + \frac{c_3}{2} + \frac{c_4}{4} + \frac{21c_5}{32} + \frac{3c_6}{32}\right) + n\left(c1 + \frac{c_2}{2} + \frac{5c_3}{4} + \frac{c_4}{4} + \frac{61c_5}{96} + \frac{7c_6}{96}\right) + c_1$$

$$T(n) = O(n_3)$$

Algoritm2(A[0..n], n)

```
  s = 0
  for i = 1..n                    $c_1$    n + 1
    for i = j..n                  $c_2$    $\sum_{i=1}^{n} t_i + 1$
      if  (A[i] > A[j])           $c_3$    $\sum_{i=1}^{n}\sum_{j=1}^{t_i} 1$
        for k = 1..i              $c_4$    $\sum_{i=1}^{n}\sum_{j=1}^{t_i}(t_j + 1)$
          s = s + A[i] * A[k]     $c_5$    $\sum_{i=1}^{n}\sum_{j=1}^{t_i} t_j$
  return s
```

Observatie: Nu vom lua in considerare ca si operatii *s=0* si *return s* deoarece se executa in O(1) si nu afecteaza complexitatea programului.

$$T(n) = c_1(n+1) + c_2\sum_{i=1}^{n}(t_i + 1) + c_3\sum_{i=1}^{n}\sum_{j=1}^{t_i} 1 + c_4\sum_{i=1}^{n}\sum_{j=1}^{t_i}(t_j + 1) + c_5\sum_{i=1}^{n}\sum_{j=1}^{t_i} t_j$$

a) Caz favorabil: vectorul este sortat crescator, astfel incat niciun numar nu este mai mic decat toate cele din stanga sa => A[i] > A[j] intotdeauna fals.

$t_i = i, t_j = 0$

$$T(n) = c_1(n+1) + c_2\sum_{i=1}^{n}(i+1) + c_3\sum_{i=1}^{n}\sum_{j=1}^{t_i}1 = c_1(n+1) + c_2\sum_{i=1}^{n}(i+1) + c_3\sum_{i=1}^{n}\sum_{j=1}^{n}1$$

$$T(n) = c_1(n+1) + c_2\frac{n^2+3n}{2} + c_3n^2 = n^2\left(\frac{c_2}{2}+c_3\right) + n\left(c_1+\frac{3c_2}{2}\right) + c_1 = O(n^2)$$

b) Caz defavorabil: vectorul este sortat strict crescator

$t_i = i, t_j = j$

$$T(n) = c_1(n+1) + c_2\sum_{i=1}^{n}(i+1) + c_3\sum_{i=1}^{n}\sum_{j=1}^{i}1 + c_4\sum_{i=1}^{n}\sum_{j=1}^{i}(j+1) + c_5\sum_{i=1}^{n}\sum_{j=1}^{i}j$$

$$T(n) = c_1(n+1) + c_2\frac{n^2+3n}{2} + c_3n^2 + c_4\sum_{i=1}^{n}\frac{i^2+3i}{2} + c_5n^2$$

$$T(n) = c_1(n+1) + c_2\frac{n^2+3n}{2} + c_3n^2 + c_4\frac{\sum_{i=1}^{n}i^2 + 3\sum_{i=1}^{n}i}{2} + c_5n^2$$

$$T(n) = c_1(n+1) + c_2\frac{n^2+3n}{2} + c_3n^2 + c_4\frac{\sum_{i=1}^{n}i^2 + 3\sum_{i=1}^{n}i}{2} + c_5n^2$$

$$T(n) = c_1(n+1) + c_2\frac{n^2+3n}{2} + c_3n^2 + c_4\frac{\frac{n(n+1)(2n+1)}{6} + 3\frac{n(n+1)}{2}}{2} + c_5n^2$$

$$T(n) = c_1(n+1) + c_2\frac{n^2+3n}{2} + c_3n^2 + c_4\frac{2n^3+12n^2+10n}{12} + c_5n^2$$

$$T(n) = n^3\frac{c_4}{6} + n^2\left(\frac{c_2}{2}+c_3+c_4+c_5\right) + n\left(c_1+\frac{3c_2}{2}+\frac{5c_4}{6}\right) + c_1 = O(n_3)$$

c) Caz mediu

$t_i = i, t_j = \frac{j}{2}$

$$T(n) = c_1(n+1) + c_2\sum_{i=1}^{n}(i+1) + c_3\sum_{i=1}^{n}\sum_{j=1}^{i}1 + c_4\sum_{i=1}^{n}\sum_{j=1}^{i}(\frac{j}{2}+1) + c_5\sum_{i=1}^{n}\sum_{j=1}^{i}\frac{j}{2}$$

$$T(n) = c_1(n+1) + c_2\frac{n^2+3n}{2} + c_3n^2 + c_4\sum_{i=1}^{n}\frac{i(i+5)}{4} + c_5\sum_{i=1}^{n}\frac{i(i+1)}{4}$$

$$T(n) = c_1(n+1) + c_2\frac{n^2+3n}{2} + c_3n^2 + c_4\sum_{i=1}^{n}\frac{i^2+5i}{4} + c_5\sum_{i=1}^{n}\frac{i^2+i}{4}$$

$$T(n) = c_1(n+1) + c_2\frac{n^2+3n}{2} + c_3n^2 + c_4\frac{2n^3+18n^2+16n}{24} + c_5\frac{n^3+3n^2+2n}{3}$$

$$T(n) = n^3\left(\frac{c_4}{12}+\frac{c_5}{3}\right) + n^2\left(\frac{c_2}{2}+c_3+\frac{3c_4}{4}+c_5\right) + n\left(c_1+\frac{3c_2}{2}+\frac{2c_4}{3}+\frac{2c_5}{3}\right) + c_1 = O(n^3)$$

Algoritm3(n)

```
  s = 0
  for i = 1..n                  $c_1$        n + 1
    for i = j..i*i              $c_2$        $\sum_{i=1}^{n} i^2 + 1 = \frac{2n^3+3n^3+n+6}{6}$
        s = s + j               $c_3$        $\sum_{i=1}^{n} i^2 = \frac{2n^3+3n^2+n}{6}$
  return s
```

Observatie: Nu vom lua in considerare ca si operatii *s=0* si *return s* deoarece se executa in O(1) si nu afecteaza complexitatea programului.

Observatie: Deoarece nu avem instructiuni care sa poata avea un numar de repetitii variabil in functie de cazuri, complexitatea va fi aceeasi in toate cele 3 cazuri (favorabil, defavorabil si mediu), asadar calculele vor fi efectuate o singura data.

$$T(n) = c_1(n+1) + c_2 * \sum_{i=1}^{n} i^2 + 1 + c_3\sum_{i=1}^{n} i^2$$

$$= c_1(n+1) + c_2\frac{2n^3+3n^3+n+6}{6} + c_3\frac{2n^3+3n^2+n}{6}$$

$$T(n) = n^3\left(\frac{c_3}{3}+\frac{c_4}{3}\right) + n^2\left(\frac{c_3}{2}+\frac{c_4}{2}\right) + n\left(c_2+\frac{c_3}{6}+\frac{c_4}{6}\right) + c_1 + c_2 + c_3 = O(n^3)$$

Algoritm4(n)

```
  s = 0                         $c_1$        1
  i = n / 2                     $c_2$        1
  while (i < n)                 $c_3$        $\frac{n}{2} + 2$
    j = i                       $c_4$        $\frac{n}{2} + 1$
    while (j <= n)              $c_5$        $(\frac{n}{2} + 1)(\log n + 1)$
      k = 1                     $c_6$        $\left(\frac{n}{2} + 1\right)\log n$
        while (k <= n)          $c_7$        $(\frac{n}{2} + 1)(\log n)(\log n + 1)$
          s += 1                $c_8$        $(\frac{n}{2} + 1)(\log^2 n)$
          k = k * 2             $c_9$        $(\frac{n}{2} + 1)(\log^2 n)$
      j = j * 2                 $c_{10}$     $\left(\frac{n}{2} + 1\right)\log n$
```

```
        i = i + 1                        $c_{11}$            $\frac{n}{2} + 1$
    return s
```

Observatie: Nu vom lua in considerare ca si operatie si *return s* deoarece se executa in O(1) si nu afecteaza complexitatea programului.

Observatie: Deoarece nu avem instructiuni care sa poata avea un numar de repetitii variabil in functie de cazuri, complexitatea va fi aceeasi in toate cele 3 cazuri (favorabil, defavorabil si mediu), asadar calculele vor fi efectuate o singura data.

$$T(n) = c_1 + c_2 + c_3\left(\frac{n}{2} + 2\right) + c_4\left(\frac{n}{2} + 1\right) + c_5\left(\frac{n}{2} + 1\right)(\log n + 1) + c_6\left(\frac{n}{2} + 1\right)\log n + c_7\left(\frac{n}{2} + 1\right)\log n\,(\log n + 1) + c_8\left(\frac{n}{2} + 1\right)(\log^2 n) + c_9\left(\frac{n}{2} + 1\right)(\log^2 n) + c_{10}\left(\frac{n}{2} + 1\right)\log n + c_{11}(\frac{n}{2} + 1)$$

$$T(n) = nlog^2 n\left(\frac{c_7}{2} + c_8\right) + n\log n\left(\frac{c_5}{2} + c_6 + \frac{c_7}{2}\right) + \log^2 n\,(c_7 + 2c_8) + \log n\,(c_5 + 2c_6 + c_7) + n\left(\frac{c_3}{2} + c_4 + \frac{c_5}{2}\right) + c_1 + c_2 + 2c_3 + 2c_4 + c_5$$

$$T(n) = O(n\log^2 n)$$

3. Cea mai lunga subsecventa de suma k

O prima incercare in abordarea problemei se poate prin calcularea sumei a tuturor sub-secventelor din secventa initiala si pastrarea+actualizarea celei mai lungi lungimi la fiecare pas. Implementarea este una naiva, realizata prin parcurgerea intr-o bucla dubla a vectorului, insa complexitatea obtinuta astfel va fi $O(n^2)$. In schimb, prin folosirea unui hash-table (**hm** in cazul nostru), desi spatiul auxiliar necesar creste, complexitatea scade dramatic, reusind astfel sa obtinem O(n).

Definim urmatoarele metode/functii atribuite lui **hm**:

- find(x) = returneaza true daca elementul x se gaseste in hashtable
- end() = returneaza true daca este sfarsitul hashtable-ului

Cerinta, solicitand complexitatea temproala a algoritmului in cazul cel mai defavorabil, forteaza parcurgerea tuturor instructiunilor din program in numarul maxim de or ice poate fi atins. Astfel, presupunem ca toate instructiunile conditionale sunt indeplinite iar accesul se va face pe fiecare ramura.

subsequenceOfSumK(a[0..n], n, k)

```
    sum = 0                        $c_1$          1
    maxLen = 0                     $c_2$          2
    for i = 1..n                   $c_3$          $n + 1$
        sum += a[i]                $c_4$          $n$
        if sum == k                $c_5$          $n$
        maxLen = i + 1                      $c_6$        $n$
      if hm.find(sum) == hm.end()           $c_7$        $n$
        hm[sum] = i                         $c_8$        $n$
      if hm.find(sum - k) != hm.end()       $c_9$        $n$
        if maxLen < i – hm[sum – k]         $c_{10}$     $n$
          maxLen = i – hm[sum - k]          $c_{11}$     $n$
    return maxLen                           $c_{12}$     1
```

Cazul cel mai defavorabil:

$T(n) = c_1 + c_2 + c_3(n+1) + (c_4 + c_5 + c_6 + c_7 + c_8 + c_9 + c_{10} + c_{11}) \cdot n + c_{12}$

$T(n) = n(c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_9 + c_{10} + c_{11}) + c_1 + c_2 + c_3 + c_{12}$

$T(n) = O(n)$

II. Rezolvarea unei recurente

4. ***Gasiti clasa de complexitate, folosind una dintre cele trei metode predate la curs (iterative, arbore de recurenta sau Master), pentru***

$$\boldsymbol{T(n)} = \begin{cases} \boldsymbol{3T\left(\frac{n-2}{2}\right) + k_2 n^2, daca\ n > 1, k_2} \in \mathbb{R}_+^* \\ \boldsymbol{k_1, daca\ n = 1, k_1} \in \mathbb{R}_+^* \end{cases}$$

$S(n) = T(4n-2) \leftrightarrow S\left(\frac{n}{2}\right) = T(2n-2) \qquad (1)$

$T(4n-2) = 3T\left(\frac{4n-2-2}{2}\right) + k_2(4n-2)^2$

$T(4n-2) = 3T(2n-2) + k_2(4n-2)^2$

$Din\ 1\ obtinem\ ca\ \ S(n) = 3S\left(\frac{n}{2}\right) + k_2(4n-2)^2$

$T(1) = O(1)$

$S(n) = 3S\left(\frac{n}{2}\right) + \Theta(n^2) \mid \cdot 3^0$

$S\left(\frac{n}{2}\right) = 3S\left(\frac{n}{2^2}\right) + \Theta\left(\frac{n^2}{2}\right) \mid \cdot 3^1$

…

$S\left(\frac{n}{2^k}\right) = 3S\left(\frac{n}{2^{k+1}}\right) + \Theta\left(\frac{n^2}{2^k}\right) \mid \cdot 3^k$

$Prin\ insumare => S(n) = 3S\left(\frac{n}{2^{k+1}}\right) + \sum_{i=0}^{k} \Theta\left(\frac{n^2}{2^k}\right) \mid \cdot 3^k$

$S(n) = 3^{k+1}\Theta(1) + \sum_{i=0}^{k} \Theta\left(\frac{n^2}{2^{2i}}\right) \quad \cdot 3^i$

Notam $\frac{n}{2^{k+1}} = 1 => n = 2^{k+1} => \log n = k + 1$

$S(n) = 3^{\log}\ \ \Theta(1) + \Theta\left(n^2\left(4 - 3^{k+1} \cdot 4^{-\log n+1}\right)\right) =>$

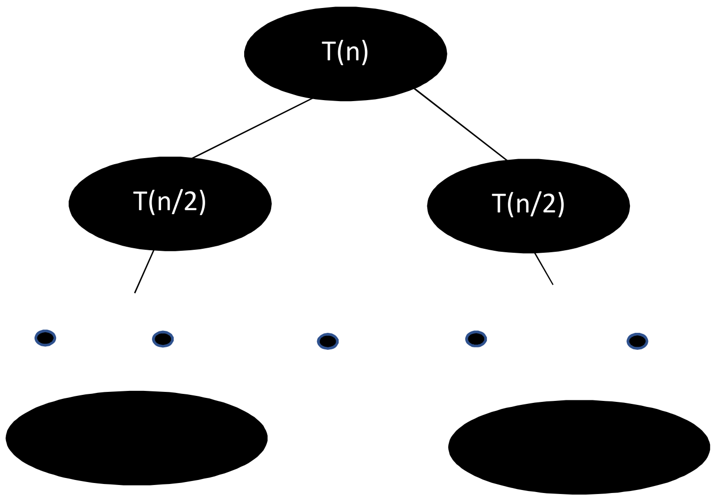
$S(n) = \Theta\left(3^{\log n}\right) + \Theta\left(4n^2 - 3^{k+1} \cdot 4^{-\log n+1} \cdot n^2\right) \quad | \quad 3^{\log n} < 3n^{\log 3}$

$S(n) = \Theta\left(n^{log3} + 4n^2 - \frac{n^{\log 3}}{4}\right) => S(n) = \Theta(n^2) => T(n) = \Theta((4n-2)^2) =>$

$T(n) = \Theta(16n^2 - 16n + 4) => T(n) = \Theta(n^2)$

$$\boldsymbol{T(n) = \begin{cases} 3T\left(\frac{n}{4}\right) + k_2 n^2, daca\ n > 1, k_2 \in \mathbb{R}_+^* \\ \qquad k_1, daca\ n = 1, k_1 \in \mathbb{R}_+^* \end{cases}, arbore\ de\ recurenta}$$

$T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n^2)$



Inaltime h.

$$T(n) = \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i k_2 n^2 + \Theta\left(n^{\log_4 n}\right) = \frac{\left(\frac{3}{16}\right)^{\log_4 n} - 1}{\frac{3}{16} - 1} k_2 n^2 + \Theta\left(n^{\log_4 n}\right)$$

$$\Theta\left(n^{\log_4 n}\right) \sim O(n^2) => T(n) = \frac{\left(\frac{3}{16}\right)^{\log_4 n} - 1}{\frac{3}{16} - 1} k_2 n^2 + \mathrm{O}(\mathrm{n}^2)$$

$$T(n) = \frac{16(1 - \left(\frac{3}{16}\right)^{\log_4 n})}{13} k_2 n^2 + \mathrm{O}(\mathrm{n}^2) = O(n^2) + O(n^2) = O(n^2)$$

$$\boldsymbol{T(n)} = \begin{cases} \boldsymbol{3T\left(\frac{n}{4}\right) + nlogn, daca\ n > 1, k_2} \in \mathbb{R}_+^* \\ \boldsymbol{k_1, daca\ n = 1, k_1} \in \mathbb{R}_+^* \end{cases}$$

$\begin{cases} a = 3 \\ b = 4 \end{cases} => n^{\log_b a} = n^{\log_4 3}$; $\log_4 3 < 1$

$f(n) = n \log n$

Caz 2: $f(n) = \Theta\left(n^{\log_4 3}\right)$, $\quad n \log n = \Theta\left(n^{\log_4 3}\right) - contradictie$

Caz 1: $f(n) = \Theta\left(n^{\log_b a - \varepsilon}\right)$, $\quad n \log n = \Theta\left(n^{\log_4 3 - \varepsilon}\right) - contradictie$

Caz 3:

a) $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$
$\exists c' \in \mathbb{R}_+^*, \exists n_0' \in \mathbb{N}^*\ a.i.\ n \lg n \geq c' n^{\log_4 3 + \varepsilon} \quad | : n, \forall n \geq n_0'$
$\lg n \geq c' n^{\log_4 3 - 1 + \varepsilon}, \forall n \geq n_0' \quad | \quad \log 1 \geq c' => 0 \geq c', c' \in \mathbb{R}_+^* \ \rightarrow contradictie$
$\log_4 3 - 1 + \varepsilon \rightarrow n\ creste\ mai\ repede\ ca \log n$
$=> \varepsilon = 1 - \log_4 3 > 0$
$\lg n \geq c', \forall n \geq n_0'$
$\begin{cases} n_0' = 2 \\ c' = 1 \end{cases} => \lg n \geq 1, \forall n \geq 2$

b) $\exists c \in (0,1), \exists n_0 \in \mathbb{N}^*\ a.i.\ af\left(\frac{n}{b}\right) \leq cf(n), \forall n \geq 0$
$\begin{cases} a = 3 \\ b = 4 \\ f(n) = n \lg n \end{cases} => 3 \frac{n}{4} \lg \frac{n}{4} \leq c\, n \lg n, \forall n \geq n_0$
$\frac{3}{4}(\lg n - \lg 4) \leq c \lg n \quad | : n, \quad \forall n \geq n_0$
$\frac{3}{4}(\lg n - 2) \leq c \lg n \quad | : n, \quad \forall n \geq n_0$
$c \geq \frac{3}{4} \frac{\lg n - 2}{\lg n} => c = \frac{3}{4} \in (0,1), \ n_0 = 2, \qquad \forall n \geq n_0$
$\begin{cases} a)\ adevarat \\ b)\ adevarat \end{cases} \xRightarrow{caz\ 3,\ \ Teorema\ Master} T(n) = \Theta(n \lg n)$

$$\boldsymbol{T(n)} = \begin{cases} \boldsymbol{3T\left(\frac{n}{3}\right) + \frac{n}{\log n}, daca\ n > 1, k_2} \in \mathbb{R}_+^* \\ \boldsymbol{k_1, daca\ n = 1, k_1} \in \mathbb{R}_+^* \end{cases}$$

$$T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{n\log n}$$

$$\begin{cases} a = 3 \\ b = 3 \end{cases} => n^{\log_b a} = n^{\log_3 3} = n$$

$$f(n) = \frac{n}{\log n}$$

Caz 2: $f(n) = \Theta\left(n^{\log_b a}\right) = \Theta(n), contradictie$

Caz 1: $f(n) = \Theta\left(n^{\log_b a - \varepsilon}\right), \qquad \frac{n}{\log} = \Theta(n^{1-\varepsilon}) - contradictie$

5. ***Gasiti clasa de complexitate, folosind metoda substitutiei si notatia Θ, pentru:***

$$\boldsymbol{T(n)} = \begin{cases} \boldsymbol{10T\left(\frac{n}{3}\right) + k_2 n^2, daca\ n > 1, k_2} \in \mathbb{R}_+^* \\ \boldsymbol{k_1, daca\ n = 1, k_1} \in \mathbb{R}_+^* \end{cases}$$

$$T(n) = 10T\left(\frac{n}{3}\right) + k_2 n^2, T(1) = \Theta(1)$$

$$T(n) = \Theta\left(n^{\log_3 10} - n^2\right) => \exists c_1, c_2 \in \mathbb{R}_+^*, n_0 \in \mathbb{N}^*\ a.i.\ c_1\left(n^{\log_3 10} - n^2\right) \le T(n) \le c_2\left(n^{\log_3 10} - n^2\right), \forall n \ge n_0$$

Caz de baza:

$$n = 1 => 0 \le T(n) \le 0, contradictie$$

$$n = 2 => c_1\left(2^{\log_3 10} - 4\right) \le T(2) \le c_2\left(2^{\log_3 10} - 4\right), dar\ T(2) = 10T\left(\frac{2}{3}\right) + \Theta(n) \rightarrow$$
$nu\ putem\ afla$ T$(\frac{2}{3})$

$$n = 3 => c_1\left(3^{\log_3 10} - 9\right) \le T(3) \le c_2\left(3^{\log_3 10} - 9\right) \rightarrow c_1 \le T(3) \le c_2 => n_0 = 3$$

Pas de inductie: $\frac{n}{3} \rightarrow n$

Ipoteza de inductie

$$c_1\left(\frac{n^{\log_3 10}}{3^{\log_3 10}} - \frac{n^2}{9}\right) \le T(n) \le c_2\left(\frac{n^{\log_3 10}}{3^{\log_3 10}} - \frac{n^2}{9}\right) => \ c_1 n^{\log_3 10} - \frac{10}{9}c_1 n^2 + k_2 n^2 \le T(n)$$
$$\le c_2\left(n^{\log_3 10} - n^2\right) + n^2\left(k_2 - \frac{c^2}{9}\right)$$

$k_2 - \frac{c_1}{9} \geq 0 \ \ si \ \ k_2 - \frac{c_2}{9} \leq 0 => c_1 \leq 9k_2 \leq c_2$

$=> \begin{cases} c_1 \leq T(3) \leq c_2 \\ c_1 \leq 9k_2 \leq c_2 \end{cases} => \begin{cases} c_1 \leq 10k_1 + 9k_2 \leq c_2 \\ \quad c_1 \leq 9k_2 \leq c_2 \end{cases}, k_1, k_2 \in \mathbb{R}_+^*$

$=> \begin{cases} \quad c_1 \leq 9k_2 \\ c_2 \geq 10k_1 + c_1 \end{cases} => \exists\ c_1, c_2 \in \mathbb{R}_+^*, n_0 = 3\ a.i. \quad T(n) = \theta\left(n^{\log_3 10} - n^2\right), \forall n \geq n_0$

$$\boldsymbol{T(n)} = \begin{cases} \boldsymbol{2T\left(\frac{n}{3}\right) + k_2 n^4, daca\ n > 1, k_2} \in \mathbb{R}_+^* \\ \qquad \boldsymbol{k_1, daca\ n = 1, k_1} \in \mathbb{R}_+^* \end{cases}$$

Folosim metoda iterative pentru a afla clasa de complexitate

$T(n) = 2T\left(\frac{n}{3}\right) + k_2 n^4 \quad | \cdot 2^0$

$T\left(\frac{n}{3}\right) = 2T\left(\frac{n}{3^2}\right) + \Theta(\left(\frac{n}{3}\right)^4) \quad | \cdot 2^1$

…

$T\left(\frac{n}{3^k}\right) = 2T\left(\frac{n}{3^{k+1}}\right) + \Theta\left(\left(\frac{n}{3^k}\right)^4\right) \quad | \cdot 2^k$

$3^{k+1} = n \to k + 1 = \log_3 n => T(n) = 2^{k+1}T(1) + \sum_{i=0}^{k} \Theta(n^4)\frac{2^i}{3^{4i}} => T(n) = \Theta(n^4)$

$\exists c_1, c_2 \in \mathbb{R}_+^*, n_0 \in \mathbb{N}^*\ a.i. c_1 n^4 \leq T(n) \leq c_2 n^4, \qquad \forall n \geq n_0$

Caz de baza: $n = 1 \to c_1 \leq T(1) \leq c_2, valid$

Pas de inductie: n/3

Ipoteza de inductie

$c_1 \left(\frac{n}{3}\right)^4 \leq T\left(\frac{n}{3}\right) \leq c_2 \left(\frac{n}{3}\right)^4$

$$2c_1\left(\frac{n}{3}\right)^4 + k_2n^4 \le T\left(\frac{n}{3}\right) \le 2c_2\left(\frac{n}{3}\right)^4 + k_2n^4$$

$$\frac{2}{81}c_1\,n^4 + k_2n^4 \le T\left(\frac{n}{3}\right) \le \frac{2}{81}c_2n^4 + k_2n^4$$

III. Gasirea si rezolvarea unei recurente

6. ***Fie urmatorii doi algoritmi care calculeaza produsul a doua numere naturale, avand n cifre. Gasiti recurenta de complexitate pentru fiecare algoritm si aplicati una dintre cele patru metode predate la curs (iterative, arbore de recurenta, substitutie sau Master) pentru a determina clasa de complexitate.***

Algoritm 1 | Inmultire 1

```
Procedure ALGORITHM1(x, y, n)
  if n = 1 then                          $c_1$      1
    return x * y                         $c_2$      1
  else
    m = [n / 2]                          $c_3$      1
    p = 10^m                             $c_4$      1
    a = x div p                          $c_5$      1
    b = x mod p                          $c_6$      1
    c = y div p                          $c_7$      1
    d = y mod p                          $c_8$      1
    e = algorithm1(a, c, m)              $c_9$      $T(\frac{n}{2})$
    f = algorithm1(b, d, m)              $c_{10}$   $T(\frac{n}{2})$
    g = algorithm1(b, c, m)              $c_{11}$   $T(\frac{n}{2})$
    h = algorithm1(a, d, m)              $c_{12}$   $T(\frac{n}{2})$
    return p^2 * e + p * (g + h) + f     $c_{13}$   $T(\frac{n}{2})$
```

Algoritmul imparte la fiecare intrare pe ramura else numerele in jumatati (prima jumatate de cifre din numar, respective cea de-a doua jumatate de cifre din numar). Astfel, algoritmul functioneaza dupa tehnica de divide et impera. Astfel, fiecare apel recursive va reduce lungimea numerelor existente la n/2 cifre.

Caz de baza: $k_1 = c_1 + c_2$
Recursivitate: $k_2 = c_1 + c_3 + c_4 + c_5 + c_6 + c_5 + c_8 + c_{13}$

$$T(n) = \begin{cases} k1, & n = 1 \\ 4 \cdot T\left(\frac{n}{2}\right) + k_2n, & n > 1 \end{cases}$$

$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n) \quad | \cdot 4^0$
$T\left(\frac{n}{2}\right) = 4T\left(\frac{n}{2^2}\right) + \Theta\left(\frac{n}{2}\right) \quad | \cdot 4^1$
…
$T\left(\frac{n}{2^k}\right) = 4T\left(\frac{n}{2^{k+1}}\right) + \Theta\left(\frac{n}{2^k}\right) \quad | \cdot 4^k$

$=> T(n) = 4^{k+1}T\left(\frac{n}{2^{k+1}}\right) + \Theta\left(n\sum_{i=0}^{k}\left(\frac{4}{2}\right)^i\right), \qquad 2^{k+1} = n => T(n) = n^2\Theta(1) + \Theta\left(n\ \frac{n-1}{1}\right)$
$T(n) = \Theta(n^2)$

Algoritm 2 | Inmultire 2

```
Procedure ALGORITHM2(x, y, n)
  if n = 1 then                            $c_1$    1
    return x * y                           $c_2$    1
  else                                     $c_3$
    m = [n / 2]
    p = 10^m
    a = x div p
    b = x mod p
    c = y div p
    d = y mod p
    e = algorithm2(a, c, m)                     $T\left(\frac{n}{2}\right)$
    f = algorithm2(b, d, m)                     $T\left(\frac{n}{2}\right)$
    g = algorithm2(a - b, c - d, m)             $T\left(\frac{n}{2}\right)$
    return p^2 * e + p * (e + f - g) + f
```

Conceptul de functionare al algoritmului se bazeaza ca si la exemplul anterior pe metoda divide-et-impera.

Caz de baza: $k_1 = c_1 + c_2$
Recursivitate: $k_2 = c_1 + c_3$

$$T(n) = \begin{cases} k1, \ n = 1 \\ 3T\left(\frac{n}{2}\right) + k_2 n, n > 1 \end{cases}$$

$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) \quad | 3^0$
$T\left(\frac{n}{2}\right) = 3T\left(\frac{n}{2^2}\right) + \Theta\left(\frac{n}{2}\right) \quad | 3^0$
…
$T\left(\frac{n}{2^k}\right) = 3T\left(\frac{n}{2^{k+1}}\right) + \Theta\left(\frac{n}{2^k}\right) \quad | 3^0$

$$=> T(n) = 3^{k+1}T(1) + \Theta(n)\sum_{i=0}^{k}\left(\frac{3}{2}\right)^k, \quad 2^{k+1} = n => T(n) = 3^{\log n}\Theta(1) + \Theta(n)\frac{\left(\frac{3}{2}\right)^{\log n} - 1}{\frac{3}{2} - 1}$$

$$T(n) = \Theta\left(n^{\log 3}\right) + \Theta\left(\frac{n^{\log 3} - n}{\frac{3}{2} - 1}\right) \quad => \quad T(n) = \Theta\left(n^{\log 3}\right)$$

II. Problema 2

Structura de date solicitata in cadrul celei de-a doua probleme necesita o complexitate cat mai buna, avand ca referinta O(log n) pentru toate operatiile ce se efectueaza asupra acestei structuri. Astfel, pornim urmatoarea discutie in dezbaterea structurii alese:

a) **Vectori simpli** – Au lookup-ul in O(1), dar simplul fapt ca inserarea poate ajunge la O(n) ne trimite cu gandul ca nu este structura potrivita pentru aceasta problema.
b) **Stiva, coada, lista simplu inlantuita, lista dublu inlantuita** – Desi inserarea si stergerea se fac in O(1), celelalte operatii vor urca pana la O(n).
c) **Skip list** – Operatiile in medie se vor executa in O(log N), ceea ce este si necesar in problema noastra, insa pe worst-case scenario vor ajunge pana la O(n).
d) **Arbori (B-Tree, AVL Tree, Red Black Tree) –** Au toate operatiile in O(log n) in cel mai rau caz.

Tinand cont ca nu avem precizata o limita de memorie, ci este necesara doar o complexitate optima din punct de vedere al timpului, alegerea facuta va fi strict influentata de complexitatea in timp. Principala problema ce apare in rezolvarea cerintei este data de existenta operatiilor indexate in structura. Pana acum, in nicio carte (incluzand Introduction to Algorithms de Thomas Cormen, The Art of Computer Programming de Donald E. Knuth) si pe niciun site de specialitate (GeeksForGeeks, Dev.To, StackOverflow, Infoarena) nu am reusit sa intampin existenta unei structuri ce va pastra elementele indexat si va putea folosi toate operatiile mentionate in cerinta.
Cazurile tipice in Big-O-Notation sunt O(1) < O(log n) < O(n) < O(n log n) < O(n^2).

Astfel, optez pentru implementarea unui AVL Tree, in care voi pastra si index-ul unui element. Conventia facuta este ca index-urile elementelor vor fi unice (nu va fi tratat cazul inserarii la un index deja existent).

Am facut o conventie in cadrul functiei split(), care va returna doar arborele ramas avand ca radacina indexul respectiv.

Analiza complexitatilor – va fi facuta prin explicatii si deduceri (din cauza complexitatii si dimensiunii codului), ideile de complexitate in cazul atribuirilor directe, apeluri recursive de cautare etc. fiind deja analizate si aprofundate in cadrul cursului / laboratorului.

a) Functii auxiliare

i. maxComparison => O(1) – caz constant, returneaza maximul a doua numere

ii. rotateRight => O(1) – timp constant. Functia executa o serii de verificari if-else (care se executa in timp constant) impreuna cu o serie de atribuiri de complexitate O(1). Apelul functiei maxComparison face trimiterea la o functie ce se executa in O(1), deci nu va afecta complexitatea programului

iii. rotateLeft => O(1), asemanator cazului ii)

iv. rebalanceTree => nu este o functie recursiva, iar in cadrul ei se vor executa doar operatii ce necesita un timp constant in rezolvare

b) Functii necesare rezolvarii cerintei

i. init => O(1), se executa doar atribuiri si alocari de memorie

ii. insert => O(log n). In vederea inserarii unui element se cauta pozitia neceseara inserarii acestuia in mod recursive, urmata de rebalansarea arborelui, ceea ce duce la analiza complexitatii prin apropierea acesteia de O(log n). Complexitatea este data de

iii. removeItem (functia a fost redenumita din remove in removeItem pentru a nu exista conflicte de denumiri cu functiile din bibliotecile standard C) => O(log n). Pentru eliminarea unui element din arbore, acesta trebuie cautat (in mod recursiv) ceea ce duce nivelul complexitatii in O(log n), urmata de stergerea acestuia si rebalansarea arborelui. Apelul recursive duce la injumatatirea (de fiecare data) a dimensiunii problemei, mergandu-se doar pe una dintre cele doua ramuri ale unui nod.

iv. lookup => O(log n). Dimensiunea problemei este injumatatita la fiecare apel recursive, mergandu-se doar pe una din cele doua ramuri ale nodului.

v. size => O(n). Functia este self-explanatory, printr-un apel recursive se numara cate noduri exista in arbore. Totodata, poate fi improved prin atribuirea arborelui unui numar ce va reprezenta numarul de noduri si va fi actualizat la fiecare inserare / stergere, astfel putand fi trimisa in O(1) cu acces constant si memorie aditionala constanta (int size).

vi. split => O(log n). Functia (conform conventiei facute la inceput) va cauta elementul necesar prin injumatatirea problemei la fiecare pas, returnand astfel arborele obtinut avand ca radacina nodul respectiv. Functia poate fi upgrad-ata prin conectarea nodului parinte al elementului cautat si trimiterea lui catre NULL, fiind astfel obtinuti 2 arbori (unul avand ca parinte root-ul initial, unul avand ca parinte nodul dupa care facem split). In acest caz, va fi necesara o noua rebalansare a fiecarui arbore astfel obtinut.

vii. concat => O(log n). Concatenam cei 2 arbori prin parcurgerea pana la intalanirea unui element NULL (de aici si complexitatea mentionata anterior, injumatatirea dimensiunii problemei se face prin trecerea la un copil al nodului), la care este atasat cel de-al doilea arbore, urmata de rebalansarea intregului copac.

viii. displaySequence => O(n). Este necesara parcurgerea tuturor elementelor in cazul printarii lor.