# Complexități

1. [1]Se dă un algoritm care primește ca date de intrare trei tipuri de input de mărime n.

Pentru tipul 1 de date de intrare, complexitatea ca timp de execuţie este $\theta(n^4)$, iar probabilitatea de a avea acest input este de $\frac{1}{n^2}$

Pentru tipul 2 de date de intrare, complexitatea ca timp de execuţie este $\theta(n^3)$, iar probabilitatea de a avea acest input este $\frac{1}{n}$.

Pentru tipul 3 de date de intrare, complexitatea ca timp de execuţie este $\theta(n)$, iar probabilitatea de a avea acest input este $1 - \frac{1}{n} - \frac{1}{n^2}$. Calculaţi:

a) Complexitatea în cazul defavorabil: $\theta(n^4)$ - cazul cu tipul 1 de date de intrare

b) Complexitatea în cazul favorabil: $\theta(n)$ - cazul cu tipul 3 de date de intrare

c) Complexitatea în cazul mediu

- **caz mediu** - timp de execuție.
  - *average complexity* (AC): $AC(A) = \sum_{I \in D} P(I)E(I)$

A - algoritm; *E(I)* număr de operații; P*(I)* probabilitatea de a avea *I* ca și date de intrare
D – mulțimea tuturor datelor de intrare posibile pentru un n fixat

$\Rightarrow n^4 \cdot \frac{1}{n^2} + n^3 \cdot \frac{1}{n} + n \cdot (1 - \frac{1}{n} - \frac{1}{n^2}) =$

$= n^2 + n^2 + n - 1 - \frac{1}{n}$

$= 2n^2 - \frac{1}{n} + n - 1 \in \theta(n^2)$

d) Overall complexity: $O(n^4)$ (O vs. $\theta$ fiindcă at worst, we have $n^4$, dar pentru alte tipuri de date de intrare, poate merge şi mai bine ($n^3$, n) - O: „margine superioară”

2. Care este complexitatea funcției f1? *

```
def f1(n):
    p = 1
    for i in range(1, n+1):
        p = p*i
    return p
```

*Mark only one oval.*

- $O(1)$
- $O(n)$
- $\theta(n)$
- $\theta(1)$

Explicație: AC (average case) = BC (best case) = WC (worst case) - se execută tot timpul T(n) = n pași

[1] Cerințe preluate și adaptate din quiz-uri de la cursul Data Structures, MIE 2020/2021, prof. Zsuzsanna Oneț-Marian

$T(n) \in \theta(n)$

3. Care este complexitatea funcției f3? *

```python
import random


def f3(n, m):
    a = 0
    b = 0
    for i in range(n):
        a += random.randint(1, 100)
    for j in range(m):
        b += random.randint(1, 50)
```

*Mark only one oval.*

- $\theta(n \cdot m)$ time, $\theta(1)$ space
- $\theta(n + m)$ time, $\theta(n + m)$ space
- $\theta(n + m)$ time, $\theta(1)$ space
- $\theta(n \cdot m)$ time, $\theta(n + m)$ space

Read more:

Space complexity of range fn Python 2.x vs. 3.x

Complexity of other Python operations

4. Care este complexitatea ca timp de execuție a funcției f2? Calculați complexitatea în caz favorabil, defavorabil și caz mediu. *

```python
def f2(lst):
    """
    Verifica daca exista un numar par in lista
    :param lst: lista de numere naturale
    :type lst: list
    :return: True daca in lista exista un numar par, False altfel
    :rtype: bool
    """
    poz = 0
    n = len(lst)
    while poz < n and lst[poz] % 2 != 0:
        poz += 1

    return poz < n
```

A se vedea rezolvare în curs 9 (secțiunea Exemple Sume). Dacă avem informații despre tipul de listă (e.g. generare random elemente, etc), trebuie luată în considerare probabilitatea de generare în cazul mediu. Altfel, versiunea default: probabilitate 1/n ca elementul să fie pe o poziție dată.

5. Care este complexitatea (ca timp de execuție) pentru funcția f4? Scrieți calculul complexității cu ajutorul sumelor. *

```
def f4(n):
    a = 0
    for i in range(n):
        for j in range(i, n):
            a = a + i + j
```

$AC = BC = WC$

$T(n) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} 1$

$T(n) = \sum_{i=1}^{n-1} n - i - 1 + 1$

$T(n) = \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i$

$T(n) = n \cdot \sum_{i=0}^{n-1} 1 - \sum_{i=0}^{n-1} i$

$T(n) = n \cdot n - (0 + 1 + 2 + \ldots + n - 1)$

$T(n) = n^2 - \frac{n \cdot (n-1)}{2}$

$T(n) = \frac{2n^2}{2} - \frac{n \cdot (n-1)}{2}$

$T(n) = \frac{n^2 - n}{2} \in \theta(n^2)$

6. Care este complexitatea (ca timp de execuție) pentru funcția f5? *

```
def f5(n):
    k = 0
    for i in range(n / 2, n + 1):
        j = 2
        while j <= n:
            k = k + n / 2
            j = j * 2
    return k
```

*Mark only one oval.*

- ( ) $\theta(n)$
- (yellow) $\theta(n \cdot \log n)$
- ( ) $\theta(n^2)$
- ( ) $\theta(n^2 \cdot \log n)$

Explicație: $T(n) = \sum_{i=n/2}^{n} \sum_{j=2}^{n} 1$*

*Step: $j = j * 2$ pentru suma a 2-a ⇒ al doilea while se execută de $\log_2 n$ ori.

$$T(n) = \sum_{i=n/2}^{n} \log n$$

$$T(n) = \log n \sum_{i=n/2}^{n} 1$$

$$T(n) = \log n \cdot (n - \frac{n}{2} + 1)$$

$$T(n) = \log n \left(\frac{n}{2} + 1\right) \in \theta(n \cdot \log n)$$

7. Care este complexitatea (ca timp de execuție, overall complexity) pentru funcția f7? *

```
def f7(n):
    a = 0
    i = n
    while i > 0:
        a += i
        i /= 2
    return a
```

*Mark only one oval.*

- O(n)
- $O(\sqrt{n})$
- $O(\frac{n}{2})$
- $O(\log n)$
- θ(n)
- $\theta(\log n)$
- $\theta(\sqrt{n})$
- $\theta(\frac{n}{2})$

8. Care este complexitatea pentru funcția f8?

```
def f8(n):
    a = 0
    i = n
    while i > 0:
        j = 1
        while j < n:
            for k in range(0, n, 2):
                a += i + j + k
            j = j * 2
        i = i / 2
    return a
```

$\mathrm{BC} = \mathrm{AC} = \mathrm{WC}$

Pentru *for:* n/2 pași (step=2)

Pentru cele 2 while-uri: executate de $\log n$ ori (i se injumatateste de logn ori, j se multiplica de logn ori)

Complexitate: $\theta(n\ (\log n)^2)$

9. Care este complexitatea pentru funcția recursive_f1? Scrieți relația de recurență.

```
def recursive_f1(n):
    if n <= 0:
        return 1
    else:
        return 1 + recursive_f1(n - 1)
```

$$T(n) = \begin{cases} 1 \text{ daca } n \leq 0 \\ 1 + T(n-1) \text{ altfel} \end{cases}$$

$T(n) = 1 + T(n-1)$
$T(n-1) = 1 + T(n-2)$
$T(n-2) = 1 + T(n-3)$
...
$T(1) = 1 + T(0)$

$$\Rightarrow T(n) = 1 + T(n-1) = 1 + 1 + T(n-2) = 1 + 1 + 1 + T(n-3) = \ldots = 1 \cdot n + T(0) = n + 1 \in \theta(n)$$

10. Care este complexitatea pentru funcția recursive_f2? Scrieți relația de recurență.

```
def recursive_f2(n):
    if n <= 1:
        return 1
    else:
        return 1 + recursive_f2(n - 5)
```

$$T(n) = \begin{cases} 1 \text{ daca } n \leq 1 \\ 1 + T(n-5) \text{ altfel} \end{cases}$$

$T(n) = 1 + T(n-5)$
$T(n-5) = 1 + T(n-10)$
$T(n-10) = 1 + T(n-15)$
...
$T(k), k \leq 1 = 1$

$$\Rightarrow T(n) = 1 + T(n-5) = 1 + 1 + T(n-10) = 1 + 1 + 1 + T(n-15) = \ldots = 1 \cdot \frac{n}{5} = \frac{n}{5} \in \theta(n)$$

11. Care este complexitatea pentru funcția recursive_f3? Scrieți relația de recurență.

```
def recursive_f3(n):
    if n <= 0:
        return 1
    else:
        return 1 + recursive_f3(n / 2)
```

*corectare: in cod conditia de la if ar trebui sa fie n<=1

$$T(n) = \begin{cases} 1 \text{ daca } n \leq 0 \\ 1 + T(n/2) \text{ altfel} \end{cases}$$

$T(n) = 1 + T(n/2)$
$T(n/2) = 1 + T(n/2^2)$
...
$T(k) = 1, k \leq 1$

$\Rightarrow T(n) = 1 + T(n/2) = 1 + 1 + T(n/2^2) = \ldots = 1 \cdot \log_2 n + 1 \in \theta(\log n)$

12. Care este complexitatea pentru funcția recursive_f4? Scrieți relația de recurență.

```
def recursive_f4(n, m, o):
    if n <= 0:
        print('m is', m, '& n is', n)
    else:
        recursive_f4(n - 1, m + 1, o)
        recursive_f4(n - 1, m, o + 1)
```

$$T(n) = \begin{cases} 1 \text{ daca } n \leq 0 \\ 1 + 2T(n-1) \text{ altfel} \end{cases}$$

$T(n) = 1 + 2T(n-1)$
$T(n-1) = 1 + 2T(n-2)$
...
$T(1) = 1 + 2T(0)$
$T(0) = 1$

$T(n) = 1 + 2T(n-1)$
$T(n-1) = 1 + 2T(n-2) \mid \cdot 2 \Rightarrow 2T(n-1) = 2 + 2^2T(n-2)$
$T(n-2) = 1 + 2T(n-3) \mid \cdot 2^2 \Rightarrow 2^2T(n-2) = 2^2 + 2^3T(n-3)$
$T(n-3) = 1 + 2T(n-4) \mid \cdot 2^3 \Rightarrow 2^3\ T(n-3) = 2^3 + 2^4T(n-4)$
...

$T(2) = 1 + 2T(1) \mid \cdot 2^{n-2} \Rightarrow 2^{n-2}T(2) = 2^{n-2} + 2^{n-1}T(1)$
$T(1) = 1 + 2T(0) \mid \cdot 2^{n-1} \Rightarrow 2^{n-1}T(1) = 2^{n-1} + 2^nT(0) = 2^{n-1} + 2^n$

$\Rightarrow T(n) = 1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1} - 1 \in \theta(2^n)$

13. Care este complexitatea pentru funcția recursive_f5? Scrieți relația de recurență.

```
def recursive_f5(n):
    print(n)
    for i in range(n):
        print('*' * n)
    if n < 1:
        return 1
    else:
        return 1 + recursive_f5(n - 1)
```

**Considering print('*' *n) replaced with print(i) - pentru a evita complicarea problemei prin considerarea complexitatii multiplicarii string*int.

$$T(n) = \begin{cases} 1 \text{ daca } n = 0 \\ n + T(n-1) \text{ altfel} \end{cases}$$

$T(n) = n + T(n-1)$
$T(n-1) = n - 1 + T(n-2)$
$T(n-2) = n - 2 + T(n-3)$
...
$T(1) = 1 + T(0)$
$T(0) = 1$

$T(n) = n + T(n-1) = n + n - 1 + T(n-2) = n + n - 1 + n - 2 + T(n-3) = \ldots = n + (n-1) + (n-2) + \ldots + 3 + 2 + 1 + T(0) = \frac{n(n+1)}{2} + 1 \in \theta(n^2)$

***Pentru funcțiile *recursive_f1,...,recursive_f4*, în loc de 1 putem scrie *a* - constantă - numărul de pași nu este neapărat 1 (pas: 1 adunare, 1 atribuire, 1 comparatie/verificare condiție etc), dar este constant;