Complexități

1. *Căror clase de complexități aparține funcția următoare? *

$$T(n) = 4n^2 + \frac{6}{7}n + 240$$

Check all that apply.

- [x] $\Omega(1)$
- [ ] $\theta(1)$
- [x] $\Omega(n)$
- [ ] $O(1)$
- [ ] $\Omega(n^3)$
- [ ] $O(n)$
- [ ] $\theta(n)$
- [ ] $\theta(2^n)$
- [x] $O(n^2)$
- [x] $O(n^3)$
- [ ] $\Omega(2^n)$
- [x] $O(2^n)$
- [ ] $\theta(n^3)$
- [x] $\theta(n^2)$
- [x] $\Omega(n^2)$

$T(n) = 4n^2 + \frac{6}{7}n + 240$

NOTAŢIA O („BIG-OH")

<u>Definiţia 1</u> (Notaţia O, "Big-oh"). Spunem că $T(n) \in O(f(n))$ dacă există **c** şi **n₀** constante pozitive (care nu depind de *n*) astfel încât $0 \le T(n) \le c \cdot f(n), \quad \forall n \ge n_0$.

$T(n) = 4n^2 + \frac{6}{7}n + 240$

$\in O(n^2), O(n^3), O(n^4), \dots.$

Demonstrație:

$n \geq 1$

$4n^2 \leq 4n^2$

$\frac{6}{7}n \leq \frac{6}{7}n^2$

$240 \leq 240n^2$
---------------------- +

$4n^2 + \frac{6}{7}n + 240 \leq 4n^2 + \frac{6}{7}n^2 + 240n^2$

$T(n) \leq n^2 (4 + \frac{6}{7} + 240)$

$\mathbf{n \geq 1}$

$\mathbf{c = (4 + \frac{6}{7} + 240)}$

$\mathbf{f(n) = n^2}$

Dar de asemenea $T(n) \leq n^3 (4 + \frac{6}{7} + 240)$, $T(n) \leq n^4 (4 + \frac{6}{7} + 240), \dots.$

Iar pentru $\forall$ n ≥ 4, $n^2 \leq 2^n \Rightarrow T(n) \in O(2^n)$

NOTAȚIA Ω („BIG-OMEGA")

<u>Definiția 2</u> (Notația Ω, "Big-omega"). Spunem că $T(n) \in \Omega(f(n))$ dacă există **c** și $\mathbf{n_0}$ constante pozitive (care nu depind de *n*) astfel încât $0 \leq c \cdot f(n) \leq T(n), \quad \forall n \geq n_0$.

$T(n) = 4n^2 + \frac{6}{7}n + 240$

$\in \Omega(n^2), \Omega(n), \Omega(1)$

Demonstrație:

$n \geq 1$

$4n^2 \leq 4n^2 + \frac{6}{7}n + 240 \;\; \forall n \geq 1$

$\mathbf{n \geq 1}$

$\mathbf{c = 4}$

$\mathbf{f(n) = n^2}$

Dar de asemenea $4n \leq T(n)$, $4n \leq 4n^2 \; \forall n \geq 1$, $1 \leq T(n)$, deci putem spune și că $T(n) \in \Omega(n), \Omega(1)$

NOTAȚIA θ („BIG-THETA")

Definiţia 3 **(Notaţia θ , "Big-theta")**. Spunem că $T(n) \in \theta(f(n))$ dacă $T(n) \in O(f(n))$ şi dacă $T(n) \in \Omega(f(n))$, altfel spus dacă există **c1, c2** şi **n_0** constante pozitive (care nu depind de n) astfel încât $c1 \cdot f(n) \le T(n) \le c2 \cdot f(n), \quad \forall n \ge n_0$.

$\mathrm{T(n)} \in \theta(\mathrm{n}^2)$

Demonstrație:

$4\mathrm{n}^2 \le \mathrm{T(n)} \le \mathrm{n}^2 \left(4 + \frac{6}{7} + 240\right)$

$\mathbf{n \ge 1}$

$\mathbf{c_1 = 4}$

$\mathbf{f(n) = n^2}$

$\mathbf{c_2} = 4 + \frac{6}{7} + 240$

Read more:

https://cs.stackexchange.com/questions/23068/how-do-o-and-%CE%A9-relate-to-worst-and-best-case

https://stackoverflow.com/questions/12138212/difference-between-big-theta-and-big-o-notation-in-simple-language

2. *[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 execuție 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 execuție 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 execuție este $\theta(n)$, iar probabilitatea de a avea acest input este $1 - \frac{1}{n} - \frac{1}{n^3}$. Calculați:

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

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

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

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

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

3. Care este complexitatea funcției f1? *

```python
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

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

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

4. 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]

5. 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).

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

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

$AC = BC = WC$

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

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

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

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

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

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

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

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

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

```python
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.

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

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.

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

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

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

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

8. Care este complexitatea pentru funcția f6? Calculați complexitatea overall.

```
def f6(n):
    for i in range(1, 2 * n - 5):
        j = i + 1
        ok = True
        a = 0
        while j <= 2 * n and ok:
            a = random.randint(1, 10)
            if a % 5 == 0:
                ok = False
            j = j + 1
```

CAZ DEFAVORABIL: ok nu devine fals, pentru fiecare i while se execută de 2n-i ori

$$T(n) = \sum_{i=1}^{2n-5} \sum_{j=i+1}^{2n} 1$$

$$T(n) = \sum_{i=1}^{2n-5} 2n - i$$

$$T(n) = \sum_{i=1}^{2n-5} 2n - \sum_{i=1}^{2n-5} i$$

$$T(n) = 2n \cdot \sum_{i=1}^{2n-5} 1 - \sum_{i=1}^{2n-5} i$$

$$T(n) = 2n \cdot (2n-5) - (1+2+\ldots+2n-5)$$

$$T(n) = 4n^2 - 10n - \frac{(2n-5)(2n-4)}{2}$$

$$T(n) = 4n^2 - 10n - (2n-5)(n-2)$$

$$T(n) = 4n^2 - 10n - (2n^2 - 4n - 5n + 10)$$

$$T(n) = 2n^2 - n - 10 \in \theta(n^2)$$

CAZ FAVORABIL: ok devine fals de fiecare dată din prima, i.e. while se execută doar 1 dată

$$T(n) = \sum_{i=1}^{2n-5} 1 = 2n - 5 \in \theta(n)$$

CAZ 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

- *for* se execută întotdeauna de 2n-5 ori
- nr. de pași pentru *while* depinde de momentul schimbării valorii variabilei *ok* ⇒ am avea nevoie de probabilitatea de a genera un număr divizibil cu 5 din 1,2, 2n-i încercări

OVERALL COMPLEXITY: $O(n^2)$

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

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

Mark only one oval.

- [ ] $O(n)$
- [ ] $O(\sqrt{n})$
- [ ] $O(\frac{n}{2})$
- [ ] $O(\log n)$
- [ ] $\theta(n)$
- [x] $\theta(\log n)$
- [ ] $\theta(\sqrt{n})$
- [ ] $\theta(\frac{n}{2})$

10. Care este complexitatea pentru funcția f8?

```python
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: $\theta(n\ (\log n\ )^2)$

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

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

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

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

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

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

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

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

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

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

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

```python
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

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

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

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

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

```python
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)
```

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

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

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

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

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

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

```python
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)
```

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

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

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

**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;