CURS 8
Complexitatea algoritmilor

Complexitatea unui algoritm:
1. timp de executare (complexitate computațională)
2. memoria utilizată

Se presupune faptul că algoritmii comparați au aceeași dimensiune a datelor de intrare!!!

Complexitate computațională = o estimare a numărului de operații elementare efectuate de către algoritm în funcție de dimensiunile datelor de intrare

Notație (Big O): $\mathcal{O}(\text{numărul maxim de operații elementare estimat})$

Exemplu: $\mathcal{O}(n^2)$ => dimensiunea datelor de intrare este n (variabila din expresie), iar algoritmul efectuează aproximativ n^2 operații elementare (expresia)

Operațiile elementare pe care le efectuează un algoritm sunt:
1. operația de atribuire și operațiile aritmetice
2. operația de decizie și operația de salt
3. operații de citire/scriere

Estimarea complexității unui algoritm

Exemplu 1: Determinarea maximului dintr-o listă

Instrucțiune	Operații elementare	
`n = int(input("Numar elemente: "))`	1 afișare + 1 citire	
`lista = []`	1 atribuire	
`for i in range(n):`	de n ori:	3n operații elementare
`    elem = int(input("Element:"))`	1 afișare + 1 citire	
`    lista.append(elem)`	1 atribuire	
`maxim = lista[0]`	1 atribuire	
`for i in range(1, n):`	de n-1 ori:	n-1 sau 2(n-1) operații elementare
`    if lista[i] > maxim:`	1 operație de decizie	
`        maxim = lista[i]`	1 atribuire?	
`print("Maximul:", maxim)`	1 afișare	
TOTAL:	5n-2+5 operații elementare	

TOTAL = 5n+3 => complexitatea $\mathcal{O}(5n+3) \approx \mathcal{O}(n)$

Reguli de reducere a expresiilor din complexitatea unui algoritm:

1. constantele (multiplicative sau aditive) nu contează

$$\mathcal{O}(5n+3) \approx \mathcal{O}(5n) \approx \mathcal{O}(n)$$

2. dintr-o expresie se păstrează doar termenul dominant ($n \to \infty$)

$$\mathcal{O}(3n^2+5n+7) \approx \mathcal{O}(3n^2) \approx \mathcal{O}(n^2)$$

$$\mathcal{O}(2^n+3n^2) \approx \mathcal{O}(2^n)$$

Exemplu 2: Sortarea prin selecție

<table>
<tr><th>Instrucțiune</th><th colspan="2">Operații elementare</th></tr>
<tr><td><code>n = int(input("Numar elemente: "))</code></td><td colspan="2">1 afișare + 1 citire</td></tr>
<tr><td><code>lst = []</code></td><td colspan="2">1 atribuire</td></tr>
<tr><td><code>for i in range(n):</code></td><td>de n ori:</td><td rowspan="3">3n operații elementare</td></tr>
<tr><td><code>   elem = int(input("Element: "))</code></td><td>1 afișare + 1 citire</td></tr>
<tr><td><code>   lst.append(elem)</code></td><td>1 atribuire</td></tr>
<tr><td><code>for i in range(n-1):</code></td><td colspan="2" rowspan="4">$\frac{n(n-1)}{2}$ operații de decizie
sau
$n(n-1)$ operații de decizie + atribuire</td></tr>
<tr><td><code>   for j in range(i+1, n):</code></td></tr>
<tr><td><code>   if lst[i] > lst[j]:</code></td></tr>
<tr><td><code>   lst[i], lst[j] = lst[j], lst[i]</code></td></tr>
<tr><td><code>print("Lista sortata:")</code></td><td colspan="2">1 afișare</td></tr>
<tr><td><code>for i in range(n):</code></td><td>de n ori:</td><td rowspan="2">n operații elementare</td></tr>
<tr><td><code>   print(lst[i], end=" ")</code></td><td>1 afișare</td></tr>
<tr><td>TOTAL:</td><td colspan="2">$n(n-1)+4n+4 = n^2+3n+4$</td></tr>
</table>

complexitatea $\mathcal{O}(n^2+3n+4) \approx \mathcal{O}(n^2)$

Pentru i = 0 => se execută de n-1 ori operația de decizie
Pentru i = 1 => se execută de n-2 ori operația de decizie
....
Pentru i = n-2 => se execută de 1 ori operația de decizie

$$\text{TOTAL} = (n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2}$$

for(i = 0; i < n; i++)
    operație cu complexitatea $\mathcal{O}(1)$;
for(j = 0; j < m; j++)
    operație cu complexitatea $\mathcal{O}(1)$;
	Complexitatea $\mathcal{O}(n+m)$

for(i = 0; i < n; i++) operație cu complexitatea $\mathcal{O}(m)$;	Complexitatea $\mathcal{O}(nm)$

for(i = 0; i < n; i++)
    for(j = 0; j < m; j++)
        operație cu complexitatea $\mathcal{O}(1)$;
	Complexitatea $\mathcal{O}(nm)$

for(i = 0; i < n; i++)
    for(j = 0; j < m; j++)
        operație cu complexitatea $\mathcal{O}(p)$;
	Complexitatea $\mathcal{O}(nmp)$

for(i = 0; i < n; i++)
{
    for(i = 0; i < m; i++)
        operație cu complexitatea $\mathcal{O}(1)$;
    for(j = 0; j < p; j++)
        operație cu complexitatea $\mathcal{O}(1)$;
}
	Complexitatea $\mathcal{O}(n(m+p))$

Complexitatea $\mathcal{O}(nm)$ poate fi considerată:

- $\mathcal{O}(n^2)$ dacă $m \approx n$
- $\mathcal{O}(n)$ dacă $m \ll n$

Exemplu: determinarea valorilor distincte dintr-o listă de numere

```
lista = [2, 1, 1, 7, 2, 3, 4, 1, 1, 1, 2, 4]

n = int(input("Numar elemente: "))
lst = []
for i in range(n):
    elem = int(input("Element: "))
    lst.append(elem)

distincte = []
for i in range(n):
    if lst[i] not in distincte: -> O(d = nr val distincte)
        distincte.append(lst[i])

print("Elementele distincte:")
for i in range(len(distincte)):
    print(distincte[i], end=" ")
```

Complexitatea este $\mathcal{O}(nd)$, unde d reprezintă numărul valorilor distincte din listă, deci poate fi:

- $\mathcal{O}(n^2)$ dacă $d \approx n$
- $\mathcal{O}(n)$ dacă $d \ll n$

Clase uzuale de complexitate computațională (în ordine crescătoare)

1. Clasa $\mathcal{O}(1)$ – *complexitate constantă*

Exemple: orice operație elementară, suma a două numere, formule simple (rezolvarea ecuației de gradul I sau II)

2. Clasa $\mathcal{O}(\log_b n)$ – *complexitate logaritmică*

Exemple: suma cifrelor unui număr - $\mathcal{O}(\log_{10} n)$, operația de căutare binară - $\mathcal{O}(\log_2 n)$

Presupunem că numărul n are x cifre => $10^{x-1} \leq n < 10^x$ => $\log_{10} 10^{x-1} \leq \log_{10} n < \log_{10} 10^x$ => $x - 1 \leq \log_{10} n < x$ => $[\log_{10} n] = x - 1$ => $x = [\log_{10} n] + 1$.

Operația de căutare binară: să se verifice dacă o valoare x apare sau nu într-un șir format din n numere sortate crescător.

Exemplu: $v = (2, 3, 3, 7, 10, 15, 15, 25, 100, 101)$ și $x = 3$

$$\log_2 8 = 3 \Leftrightarrow 2^3 = 8 \Leftrightarrow \frac{8}{2} = 4;\ \frac{4}{2} = 2;\ \frac{2}{2} = 1$$

Algoritmul de căutare binară:

- citirea tabloului sortat crescător și a valorii x căutate - $\mathcal{O}(n + 1)$
- operația de căutare binară - $\mathcal{O}(\log_2 n)$
- afișarea rezultatului - $\mathcal{O}(1)$
- COMPLEXITATEA ALGORITMULUI: $\mathcal{O}(n + \log_2 n) \approx \mathcal{O}(n)$

3. Clasa $\mathcal{O}(n)$ – *complexitate liniară*

Exemple: citirea/scrierea/o singură parcurgere a unui tablou unidimensional cu n elemente, suma primelor n numere naturale (fără formulă) etc.

4. Clasa $\mathcal{O}(n \log_2 n)$

Exemple: Quicksort (sortarea rapidă), Mergesort (sortarea prin interclasare), Heapsort (sortarea cu ansamble)

5. Clasa $\mathcal{O}(n^2)$ – *complexitate pătratică*

Exemple: sortarea prin interschimbare, Bubblesort, compararea fiecărui element al unui tablou unidimensional cu n elemente cu toate celelalte elemente din tablou, citirea/scrierea/o singură parcurgere a unui tablou bidimensional cu n linii și n coloane

6. Clasa $\mathcal{O}(n^k), k \geq 3$ – *complexitate polinomială*

Exemple: sortarea fiecărei linii dintr-o matrice pătratică de dimensiune n folosind sortarea prin interschimbare sau Bubblesort - $\mathcal{O}(n^3)$

7. Clasa $\mathcal{O}(a^n), a \geq 2$ – *complexitate exponențială*

Exemple: generarea tuturor submulțimilor unei mulțimi cu n elemente - $\mathcal{O}(2^n)$

Teoremă: O mulțime cu n elemente are 2^n submulțimi.

Exemplu:

$$A = \{1,2,3\} \Longrightarrow \mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$$
$$\Longrightarrow |\mathcal{P}(A)| = 8 = 2^3$$

$$\mathcal{P}(A) = \text{mulțimea părților lui } A = \text{mulțimea tuturor submulțimilor lui } A$$

Observație: Complexitatea unui algoritm NU poate fi mai mică decât complexitatea citirii datelor de intrare și/sau scrierii datelor de ieșire!!!

Exemplu 1:

```
n = int(input("n = "))
i = 0
p = 1
while i <= n:
    j = 1
    while j <= p:
        print(j, end= " ")
        j = j + 1
    print()
    i = i + 1
    p = p * 2
```

```
n = 3
1                  2^0
1 2                2^1
1 2 3 4            2^2
1 2 3 4 5 6 7 8    2^3
```

$$2^0 + 2^1 + \cdots + 2^n = 2^{n+1} - 1$$

$\mathcal{O}(2^n)$

Exemplu 2:

```
a = int(input("a = "))
b = int(input("b = "))

p = 1
while p < a:
    p = p * 2
while p <= b:
    print(p, end=" ")
    p = p * 2
```

Afișează puterile lui 2 cuprinse între a și b

$$2^k \leq b => k \leq \log_2 b$$

$\mathcal{O}(\log_2 b)$

Exemplu 3:

```
v = [int(x) for x in input("Valorile: ").split()]
i = 0
j = len(v) - 1
while i < j:
    while i < len(v) and v[i] < 0:
        i = i + 1
    while j >= 0  and v[j] >= 0:
        j = j - 1
    if i < j:
        v[i], v[j] = v[j], v[i]

print("\nValorile:\n", v, sep="")
```

$\mathcal{O}(n)$

Sortare parțială: **numerele negative înaintea celor pozitive**

				j	**i**				
-10	**-15**	**-21**	**-30**	**-1**	**8**	**19**	**20**	**10**	**7**