

COMPLEXITATEA ALGORITMILOR

Complexitatea unui algoritm:

1. timp de executare
2. memoriei utilizate

Utilitate:

- a) 1 problemă -> cel puțin 2 algoritmi de rezolvare -> care este mai performant?
- b) 1 problemă -> 1 algoritm de rezolvare -> este suficient de performant pentru ceea ce am eu nevoie?

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

Notatie (Big O): O (numărul maxim de operații elementare estimat)

Exemplu: $O(n^2)$ => 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

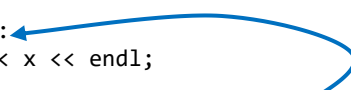
```
#include <iostream>
using namespace std;

int main()
{
    int x = 1;

    Eticheta_A:
        cout << x << endl;
        x++;
        if(x < 10) goto Eticheta_A;
    cout << endl;

    for(int x = 1; x < 10; x++)
        cout << x << endl;
    cout << endl;

    x = 1;
    while(x < 10)
    {
        cout << x << endl;
        x++;
    }
    return 0;
}
```



Estimarea complexității unui algoritm

Exemplu 1: Determinarea maximului dintr-un tablou unidimensional

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int i, n, maxv, v[1000];
```

```
    cout << "n = ";
    cin >> n;
```

1 operație elementară
1 operație elementară

```
    for(i = 0; i < n; i++)
    {
        cout << "v[" << i << "] = ";
        cin >> v[i];
    }
```

se execută de n ori:

1 operație elementară
1 operație elementară

Total: 2n operații elementare

```
    maxv = v[0];
```

1 operație elementară

```
    for(i = 1; i < n; i++)
        if(v[i] > maxv)
            maxv = v[i];
```

se execută de n-1 ori:
1 operație de decizie

Total: n-1 operații elementare

```
    cout << "Maximul: " << maxv << endl;
```

1 operație elementară

```
    return 0;
}
```

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

Reguli de reducere a expresiilor din complexitatea unui algoritm:

1. constantele (multiplicative sau aditive) nu contează

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

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

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

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

Exemplu 2: Sortarea prin interschimbare

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int i, n, maxv, v[1000];
```

```
    cout << "n = ";
    cin >> n;
```

1 operație elementară
1 operație elementară

```
    for(i = 0; i < n; i++)
    {
        cout << "v[" << i << "] = ";
        cin >> v[i];
    }
```

se execută de n ori:

1 operație elementară
1 operație elementară

Total: 2n operații elementare

```
    for(i = 0; i < n; i++)
        for(j = i+1; j < n; j++)
            if(v[i] > v[j])
            {
                aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
```

Total: $n(n-1)/2$ operații elementare

```
    cout << "Tabloul sortat: " << endl;
    for(i = 0; i < n; i++)
        cout << v[i] << " ";
```

1 operație elementară
se execută de n ori:
1 operație elementară
Total: n operații elementare

```
    return 0;
}
```

$$\text{TOTAL} = 3n + 3 + \frac{n(n-1)}{2} = 3n + 3 + \frac{n^2-n}{2} = \frac{n^2-n+6n+6}{2} = \frac{n^2+5n+6}{2} \text{ operații elementare} \Rightarrow$$

$$\text{complexitatea } \mathcal{O}\left(\frac{n^2+5n+6}{2}\right) \approx \mathcal{O}(n^2 + 5n + 6) \approx \mathcal{O}(n^2)$$

Pentru $i = 0 \Rightarrow$ se execută de $n-1$ ori operația de decizie

Pentru $i = 1 \Rightarrow$ se execută de $n-2$ ori operația de decizie

.....

Pentru $i = n-2 \Rightarrow$ se execută de 1 ori operația de decizie

Pentru $i = n-1 \Rightarrow$ se execută de 0 ori operația de decizie

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

Complexitățile unor structuri repetitive

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 $\mathcal{O}(n(m + p))$

Estimarea timpului de executare în funcție de complexitatea computațională

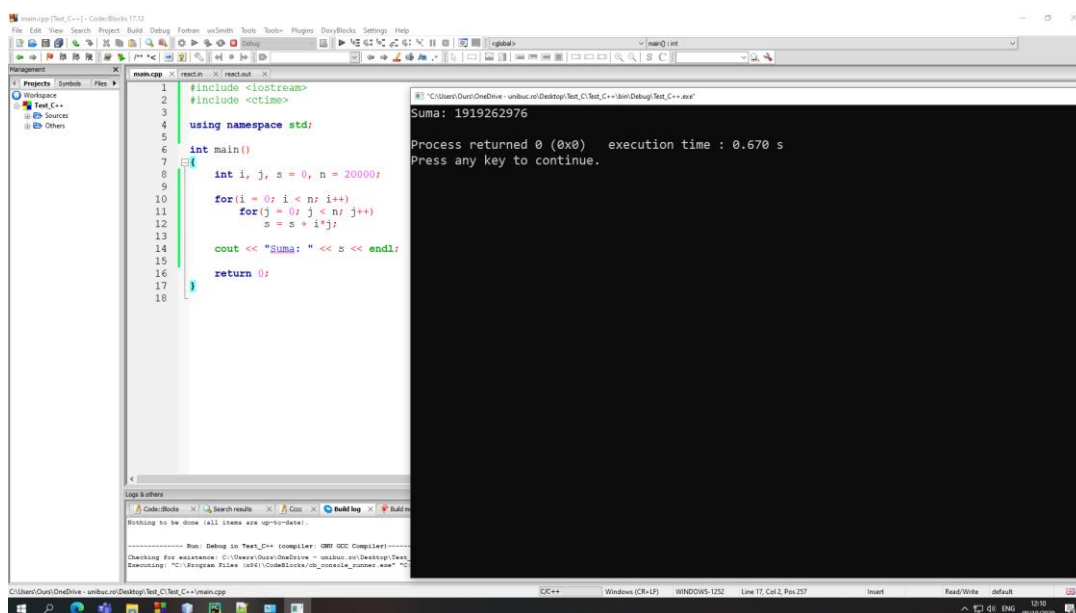
Observație: Orice operație elementară durează maxim 5 cicli de procesor!

Exemplu:

$f_p = 3\text{GHz} \Rightarrow 1 \text{ ciclu de procesor durează aproximativ } \frac{1}{3\text{GHz}} = \frac{1}{3 \times 10^9} = 0.3 \times 10^{-9} \text{ secunde} = 3 \times 10^{-10} \text{ secunde} \Rightarrow 1 \text{ operație elementară durează maxim } 5 \times 3 \times 10^{-10} \text{ secunde} = 15 \times 10^{-10} \text{ secunde!}$

Considerăm un algoritm cu complexitatea $\mathcal{O}(n^2)$ pe care vrem să-l rulăm pentru $n = 20000 = 2 \times 10^4$. Cât este, aproximativ, timpul de executare t ?

$$t = \underbrace{20000^2}_{\text{numărul de operații elementare efectuate}} \times \underbrace{5 \times 3 \times 10^{-10}}_{\text{durata maximă a unei operații elementare}} \text{ secunde} = (2 \times 10^4)^2 \times 15 \times 10^{-10} \text{ secunde} = 60 \times 10^{-2} \text{ secunde} = 0.6 \text{ secunde}$$



```
1 #include <iostream>
2 #include <ctime>
3 using namespace std;
4
5 int main()
6 {
7     int i, j, s = 0, n = 20000;
8     for(i = 0; i < n; i++)
9         for(j = 0; j < n; j++)
10             s = s + i*j;
11     cout << "Suma: " << s << endl;
12     return 0;
13 }
```

Suma: 1919262976

Process returned 0 (0x0) execution time : 0.670 s

Press any key to continue.