Facultatea de Matematică și Informatică

UNIVERSITATEA BABEȘ-BOLYAI
BABEȘ-BOLYAI TUDOMÁNYEGYETEM
BABEȘ-BOLYAI UNIVERSITÄT
BABEȘ-BOLYAI UNIVERSITY
TRADITIO ET EXCELLENTIA

Complexitatea Algoritmilor

Paul-Ioan Someșan

Importanța studiului complexității algoritmilor

- Având în vedere faptul că unitățile de calcul au acces limitat la resurse, este important ca, atunci când implementăm un algoritm, să încercăm să estimăm „câte resurse va utiliza”.
- Pentru calculator, există două resurse importante pe care le poate utiliza: **timpul** și **spațiul**.
- **Complexitatea timp** este relevantă pentru a estima ordinul de mărime al timpului de execuție, în funcție de dimensiunea datelor de intrare.
- **Complexitatea spațiu** este relevantă pentru a estima câtă memorie suplimentară este necesară pe parcursul execuției algoritmului.

Notația Big-O

- **Big-O este notația utilizată la nivel de liceu pentru analizarea complexității algoritmilor.**

  Aceasta urmărește determinarea complexității în **cel mai rău caz**.
- Pentru **complexitatea timp**, notația Big-O estimează **ordinul de mărime al numărului de pași** efectuați de algoritm în cel mai rău caz.
- Pentru **complexitatea spațiu**, notația Big-O estimează **ordinul de mărime al memoriei suplimentare** utilizate de algoritm în cel mai rău caz.

Cum arată complexitățile?

- Întrucât ne interesează doar **ordinul de mărime**, nu vom determina efectiv numărul exact de pași pentru complexitatea timp și nici cantitatea exactă de memorie utilizată pentru complexitatea spațiu.
- Ceea ce vom face este să estimăm, în funcție de datele de intrare, **ordinul de mărime al pașilor efectuați** (complexitate timp) și **ordinul de mărime al memoriei utilizate** (complexitate spațiu).
- În contextul examenelor de la finalul clasei a XII-a, complexitatea spațiu este, de regulă, mai puțin relevantă. De exemplu, int a[101]; nu depinde de datele citite, deci are complexitate spațiu **O(1)**.

Reguli importante în calculul complexităților

- Vom discuta de aici încolo doar despre **complexitățile de timp**, acestea fiind cele relevante pentru Bacalaureat și Admitere.
- Deoarece ne interesează doar **ordinul de mărime** al numărului de pași efectuați, **constantele pot fi neglijate**.
- De exemplu, $O(2 \cdot n) = O(n)$. Un program care parcurge de două ori intervalul de la 1 la n are un număr de pași proporțional cu n, deci complexitate **liniară**.
- De asemenea, $O(n + \sqrt{n}) = O(n)$, deoarece termenul dominant este n.

Complexitati consacrate

Nr. Crt.	Complexitate	Denumire
1.	$O(1)$	Complexitate constanta
2.	$O(\log_2 n)$	Complexitate logaritmica
3.	O(sqrt(n))	Complexitate radical
4.	$O(n)$	Complexitate liniara
5.	$O(n^2)$	Complexitate patratica
6.	$O(2^n)$	Complexitate exponentiala

Complexitati timp

```cpp
int n; cin >> n;
for(int i = 1; i <= n; ++i)
    cout << i << ' '; // O(n)
```

```cpp
int n; cin >> n; bool estePrim = true;
if(n < 2 || (n % 2 == 0 && n != 2))
    estePrim = false;
for(int d = 3; d * d <= n; d += 2)
    if(n % d == 0)
        estePrim = false;
cout << estePrim; // O(sqrt(n))
```

```cpp
int n; cin >> n;
for(int d = 1; d * d <= n; ++d)
    if(n % d == 0 && d * d < n)
        cout << d << ' ' << n / d << ' ';
    else cout << d << ' '; // O(sqrt(n))
```

```cpp
int n, m; cin >> n >> m; // O(n*m)
for(int i = 1; i <= n; ++i)
    for(int j = 1; j <= m; ++j)
        cout << i * j << ' ';
```

Toți algoritmii au complexități!

- Exemplele anterioare sunt foarte simple, însă lucrurile se pot complica rapid. Este important de reținut că orice algoritm poate fi caracterizat printr-o complexitate de timp și de spațiu; trebuie doar să le identificăm.
- Unele dintre cele mai dificile complexități de determinat sunt cele ale funcțiilor recursive. Vom vedea în partea aplicativă a lecției cum pot fi calculate astfel de complexități.
- Deși există mai multe teoreme și metode formale, veți vedea că există și o metodă simplă, care funcționează în toate cazurile întâlnite la nivel de liceu și admitere.

Exercitii

Sursa: Examen Admitere UBB Vara 2025

7. Se consideră algoritmul `afla(n, x)`, unde n este număr natural ($1 \leq n \leq 10^4$), iar x este un vector cu n elemente numere întregi ($x[1], x[2], ..., x[n]$, $-100 \leq x[i] \leq 100$, pentru $i = 1, 2, ..., n$):

```
Algorithm afla(n, x):
    M ← x[1]
    For i ← 1, n execute
        For j ← i, n execute
            For k ← j, n execute
                If M < x[i] + x[j] + x[k] then
                    M ← x[i] + x[j] + x[k]
                EndIf
            EndFor
        EndFor
    EndFor
    Return M
EndAlgorithm
```

Care este complexitatea timp a algoritmului?

A. $O(3 * n)$
B. $O(n^3)$
C. $O(n / 3)$
D. $O(\log_3 n)$

Sursa: Examen Admitere UBB Vara 2021

8. Fie următorul subalgoritm, având ca parametru numărul natural nenul n și care returnează un număr natural.

```
Subalgoritm f(n):
    j ← n
    CâtTimp j > 1 execută
       i ← 1
       CâtTimp i ≤ n execută
           i ← 2 * i
       SfCâtTimp
       j ← j DIV 3
    SfCâtTimp
    returnează j
SfSubalgoritm
```

În care dintre următoarele clase de complexitate se încadrează complexitatea timp a algoritmului?

A. $O(\log_2 n)$
B. $O(\log_2^2 n)$
C. $O(\log_3^2 n)$
D. $O(\log_2 \log_3 n)$

Sursa: Simulare Examen Admitere UBB – Zece la Examene - 2025

11. Se considera algoritmul `f(n)`, unde n este număr natural ($1 \leq n \leq 10^6$).

```
Algorithm f(n):
        For i ← 1, n execute
                j ← 1
                While j < n execute
                        j ← j + i
                EndWhile
        EndFor
EndAlgorithm
```

Precizați care este complexitatea acestui algoritm.

A. $O(n)$
B. $O(n^2)$
C. $O(n * \log_2 n)$
D. $O(\log_2 n)$

Sursa: Examen Admitere UBB Toamna 2024

12. Se consideră algoritmul `F(n)`, unde n este număr natural nenul ($1 \leq n \leq 10^6$). Algoritmul `sqrt(n)` returnează radicalul lui n și are complexitatea $O(1)$. Notația [a] reprezintă partea întreagă a valorii lui a. Operatorul „/" reprezintă împărțirea reală, de exemplu: 3 / 2 = 1.5.

```
Algorithm F(n):
    If n = 1 then
        Return 1
    EndIf
    i ← [n / sqrt(n)]
    Return 1 + F(i)
EndAlgorithm
```

Care dintre următoarele afirmații sunt adevărate?

A. Algoritmul `F(n)` are complexitatea timp $O(\log \log n)$.
B. În urma apelului `F(200)` se obține valoarea 4.
C. În urma apelului `F(250)` se obține valoarea 5.
D. Algoritmul `F(n)` are complexitatea timp $O(1)$.

Sursa: Concurs Mate-Info UBB 2021

29. Se consideră următorul subalgoritm recursiv `fibonacci(n)`, unde **n** este un număr natural ($1 \leq n \leq 100$). Să se determine de câte ori se afișează mesajul "`Aici`" în cazul unui apel `fibonacci(n)` (considerând împreună apelul inițial și toate apelurile recursive generate).

```
Subalgoritm fibonacci(n):
  Dacă n ≤ 1 atunci
    returnează 1
  altfel
    Scrie "Aici"
    returnează fibonacci(n - 1) + fibonacci(n - 2)
  SfDacă
SfSubalgoritm
```

A. De `fibonacci(n)` ori.
B. De `fibonacci(n-1)` ori.
C. De `fibonacci(n)-1` ori.
D. De `fibonacci(n) - fibonacci(n-1)` ori.

Sursa: Examen Admitere UB Vara 2022

1. Considerăm următorul algoritm scris în pseudocod:

```
citește n (număr natural nenul)
i ← 0; p ← 1
cât timp i <= n execută
    j ← 1
    cât timp j <= p execută
        scrie j
        j ← j + 1
    i ← i + 1; p ← p * 2
```

Complexitatea algoritmului de mai sus este egală cu:

A) $O(n \cdot \log_2 n)$ B) $O(2^n)$ C) $O(n \cdot p)$ D) $O(n^2)$

Sursa: Examen Admitere UB Vara 2024

8. Considerăm următoarea secvență de cod, în care variabilele **i**, **j**, **n** și **s** sunt de tip întreg:

```
citește n (număr natural nenul)
s ← 0
i ← n
cât timp i >= 1 execută
    j ← 1
    cât timp j <= i execută
        s ← s + 1
        j ← j * 2
        i ← i div 2
scrie s
```

Care este complexitatea timp a acestei secvențe de cod?

A) $\mathcal{O}(2^n)$ B) $\mathcal{O}(n^2)$ C) $\mathcal{O}(n \log_2 n)$ D) $\mathcal{O}(\log_2 n)$

Sursa: Examen Admitere UB Vara 2025

13. Considerăm următorul algoritm descris în pseudocod, în care variabilele **i**, **j**, **n** și **cnt** sunt toate de tip număr natural:

```
citește n (număr natural)
cnt ← 0
i ← 1
cât timp i < n execută
    j ← 0
    cât timp j < n execută
        cnt ← cnt + 1
        j ← j + i
    i ← 2 * i
scrie cnt
```

Care este complexitatea timp a algoritmului dat?

A) $\mathcal{O}(n)$ B) $\mathcal{O}(\log_2 n)$ C) $\mathcal{O}(n \cdot \log_2 n)$ D) $\mathcal{O}(n^2)$

Facultatea de Matematică și Informatică

UNIVERSITATEA BABEȘ-BOLYAI
BABEȘ-BOLYAI TUDOMÁNYEGYETEM
BABEȘ-BOLYAI UNIVERSITÄT
BABEȘ-BOLYAI UNIVERSITY
TRADITIO ET EXCELLENTIA

Va multumesc!

Zecelagrile.ro – peste 4000 de grile de informatica