COMPLEXITATEA ALGORITMILOR

Estimarea timpului de executare în funcție de complexitatea computațională

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

Exemplu:
f_P = 3GHz => 1 ciclu de procesor durează aproximativ $\frac{1}{3\text{GHz}} = \frac{1}{3\times10^9} = 0.3 \times 10^{-9}\text{secunde} = 3 \times 10^{-10}\text{secunde}$

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?

$t = (2 \times 10^4)^2 \times 5 \times 3 \times 10^{-10}$ secunde $= 4 \times 10^8 \times 15 \times 10^{-10}$ secunde $= 6 \times 10^{-1}$ secunde $= 0.6$ secunde

Clase uzuale de complexitate computațională (în ordine crescătoare)

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

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

2. Clasa $\mathcal{O}(\log_2 n)$ – *complexitate logaritmică*

Exemple: suma cifrelor unui număr - $\mathcal{O}(\textit{numărul de cifre ale lui } n \approx \log_{10} n)$, 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 - $\mathcal{O}(n)$
- operația de căutare binară - $\mathcal{O}(\log_2 n)$
- afișarea rezultatului - $\mathcal{O}(1)$
- COMPLEXITATEA ALGORITMULUI: $\mathcal{O}(n + \log_2 n + 1) \approx \mathcal{O}(n)$

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: algoritmul Roy-Floyd-Warshall-Tomescu pentru determinarea matricei drumurilor unui graf plecând de la matricea de adiacență - $\mathcal{O}(n^3)$, sortarea fiecărei linii dintr-o matrice pătratică de dimensiune n folosind sortarea prin interschimbare sau Bubblesort

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:

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

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

Exemple:

1. Putem sorta crescător $n = 1.000.000$ de numere în mai puţin de o secundă?

$f_P = 3$ GHz => 1 ciclu de procesor durează aproximativ $\frac{1}{3\text{GHz}} = \frac{1}{3\times10^9} = 0.3 \times 10^{-9}$secunde $= 3 \times 10^{-10}$secunde => 1 operaţie elementară durează maxim $5 \times 3 \times 10^{-10}$secunde $= 15 \times 10^{-10}$secunde

a) considerăm un algoritm de sortare cu complexitatea $\mathcal{O}(n^2)$ (de exemplu, sortarea prin interschimbare):

$t_1 = (10^6)^2 \times 5 \times 3 \times 10^{-10}$ secunde = $10^{12} \times 15 \times 10^{-10}$ secunde = 15×10^2 secunde = 1500 secunde = 25 minute

b) considerăm un algoritm de sortare cu complexitatea $\mathcal{O}(n \log_2 n)$ (de exemplu, Quicksort):

$t_2 = (10^6 \times \log_2 10^6) \times 5 \times 3 \times 10^{-10}$ secunde = $10^6 \times 6 \times \log_2 10 \times 15 \times 10^{-10}$ secunde = $10^6 \times 6 \times 3 \times 15 \times 10^{-10}$ secunde = 270×10^{-4} secunde = 0.027 secunde

2. Considerăm un algoritm cu complexitatea $\mathcal{O}(n^2)$. Care este valoarea maximă a lui n pentru care algoritmul rulează în mai puţin de o secundă?

$t = n^2 \times 5 \times 3 \times 10^{-10}$ secunde ≤ 1 secundă => $n^2 \times 5 \times 3 \times 10^{-10} \leq 1$ => $n^2 \leq \frac{1}{5\times3\times10^{-10}} = 0.07 \times 10^{10}$ => $n \leq \sqrt{7 \times 10^8} = 2.7 \times 10^4 = 27.000$ => $n \leq 27.000$

3. Considerăm un algoritm cu complexitatea $\mathcal{O}(2^n)$. Care este timpul de executare al său pentru $n = 100$?

$t = 2^{100} \times 5 \times 3 \times 10^{-10}$ secunde
Considerăm aproximarea $2^{10} \approx 10^3$ => $2^{100} = (2^{10})^{10} \approx 10^{30}$

$$t = 10^{30} \times 5 \times 3 \times 10^{-10}\text{secunde} = 15 \times 10^{20}\text{ secunde} = \frac{15 \times 10^{20}}{3600}\text{ ore}$$

$$= \frac{15 \times 10^{16} \times \cancel{10^4}}{\cancel{3600}}\text{ ore} = 15 \times 10^{16} \times 3\text{ ore} = 45 \times 10^{16}\text{ ore}$$

$$= \frac{45 \times 10^{16}}{24}\text{zile} = \frac{45 \times 10^{14} \times \cancel{10^2}}{\cancel{24}}\text{zile} = 45 \times 10^{14} \times 4\text{ zile}$$

$$= 180 \times 10^{14}\text{ zile} = \frac{180 \times 10^{14}}{365}\text{ani} = \frac{180 \times 10^{11} \times \cancel{10^3}}{\cancel{365}}\text{ani}$$

$$= 180 \times 10^{11} \times 3\text{ ani} = 54 \times 10^{12}\text{ani} = 54000 \times 10^{9}\text{ ani}$$

Presupunem că se inventează un procesor astfel încât pentru $n = 100$ să obținem $t = 1$ secundă.

$$2^{n+1} = 2 \times 2^n => t_{n+1} = 2 \times t_n$$

$n = 100 => t = 1$ secundă
$n = 101 => t = 2$ secunde
$n = 102 => t = 2^2$ secunde
.....
$n = 200 => t = 2^{100}$ secunde $\approx 4 \times 10^{22}$ ani

Compararea sortării prin interschimbare cu Quicksort din punctul de vedere al timpului de executare:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

//sortarea prin interschimbare -> O(n^2)
void sortare(int v[], int n)
{
    int i, j, aux;

    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;
            }
```

```
}


//functie pentru afisarea unui tablou -> O(n)
void afisare(int v[], int n)
{
    int i;

    printf("\nTabloul:\n");
    for(i = 0; i < n; i++)
        printf("%d ", v[i]);
    printf("\n");
}

int cmpNumere(const void* a, const void* b)
{
    int va = *(int*)a;
    int vb = *(int*)b;

    if(va < vb) return -1;
    if(va > vb) return +1;
    return 0;
}

int main()
{
    int i, n, *v, *w;
    double t;

    printf("Numarul de elemente din tablou: ");
    scanf("%d", &n);

    //aloc dinamic tablourile v si w
    v = (int*)malloc(n * sizeof(int));
    w = (int*)malloc(n * sizeof(int));

    //populez tablourile v si w cu n numere aleatorii

    //initializez generatorul de numere aleatorii
    //time(NULL) = numarul de secunde trecute de la 01.01.1970
    srand(time(NULL));

    //generez n valori aleatorii pe care le introduc in tabloul v
    for(i = 0; i < n; i++)
        //la fiecare apel al functiei rand() este generat un nou numar
aleatoriu
        //cuprins intre 0 si 32767
        v[i] = w[i] = rand() * rand();

//    afisare(v, n);

    //clock() = numarul de cicli de procesor utilizati de la inceputul
programului
    t = clock();
```

```
    //sortarea prin interschimbare
//    sortare(v, n);

    //calculam numarul de cicli utilizati pentru sortare
    t = clock() - t;

    //calculez timpul de executare
    t = t / CLOCKS_PER_SEC;

    printf("\nTimpul de executare pentru sortarea prin interschimbare: %.3lf
secunde\n", t);

    //eliberez zona de memorie alocata dinamic pentru tabloul v
    free(v);

    //clock() = numarul de cicli de procesor utilizati de la inceputul
programului
    t = clock();

    //Quicksort
    qsort(w, n, sizeof(int), cmpNumere);

//    afisare(w, n);

    //calculam numarul de cicli utilizati pentru sortare
    t = clock() - t;

    //calculez timpul de executare
    t = t / CLOCKS_PER_SEC;

    printf("\nTimpul de executare pentru Quicksort: %.3lf secunde\n", t);

    //eliberez zona de memorie alocata dinamic pentru tabloul w
    free(w);

    return 0;
}
```