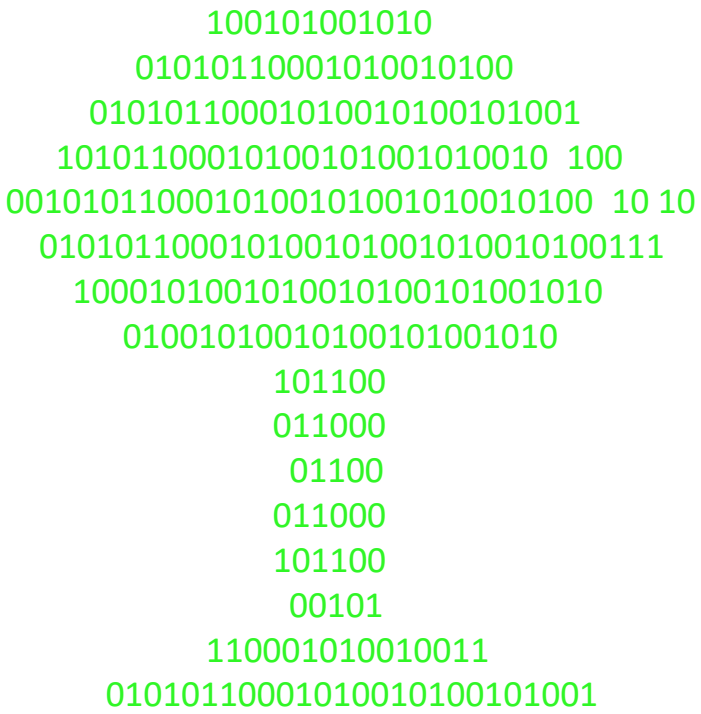


Structuri de date și algoritmi

Curs, IS – An II

Obiectivele primului curs

1. Sa intelegem **de ce** trebuie sa invatam despre structuri de date si algoritmi

2. Sa intelegem **cum** putem sa le invatam eficient
   - Ce competente vom evalua la sfarsit si cum?
   - Ce instrumente avem la dispozitie pentru invatare?

3. Sa facem cunostinta cu cateva **notiuni de baza**
   - Ce sunt structurile de date
   - Cum impacteaza ele eficienta unui program
   - Cum masuram eficienta unui program

Motivatie

- Reprezentarea informatiei este un aspect fundamental in stiinta calculatoarelor
- Studiem structurile de date pentru a invata sa dezvoltam **programe eficiente**
- Studierea efectului organizarii datelor si al algoritmilor asupra eficientei unui program

Law of the instrument (law of the hammer):

When all you have is a hammer, everything looks like a nail

(A. Maslow)

Motivatie

Am intrebat 5 AI chatbots:

- Avand in vedere cele mai recente rezultate in domeniul IA care ar putea sa le ameninte cariera, pe ce ar trebui sa se concentreze studentii mei?
- Care ar fi competentele si abilitatile esentiale/critice pe care ar trebui sa le dezvolte pentru a fi competitivi in piata muncii din domeniul lor?
- Cum ar trebui sa-si adapteze strategiile de invatare pentru a integra cat mai eficient tehnologiile IA?
- Cum ar trebui eu, ca profesor, sa adresez ingrijorarea mea ca ar putea folosi IA-ul generativ ca shortcut in procesul de invatare si in acest fel ar invata superficial si nu ar deveni experti?

Motivatie

Gemini Copilot Claude ChatGPT perplexity

Sinteza raspunsuri:

Critical thinking

Deep understanding of fundamentals and theory

Problem solving

Strong programming skills

Communication and collaboration skills

Continuous Learning and Adaptability

Ethical considerations

Motivatie

Gemini Claude ChatGPT Copilot perplexity

Selectie din raspunsuri:

Foundational Knowledge: Ensure they have a strong grasp of core computer science concepts, including data structures, algorithms, and computer architecture. These fundamentals are crucial for understanding more advanced topics. *(Copilot)*

"Deep Understanding of Fundamentals: AI thrives on the foundations. A superficial understanding of core CS concepts won't cut it. *This is where Data Structures is absolutely vital*." *(Gemini)*

Data Structures isn't just a course; it's a way of thinking about organizing information. *(Gemini)*

Coding proficiency: The AI field demands strong programming skills. Encourage students to become proficient in languages like **Python** (which is widely used for AI development), **C++** (for performance-critical applications), and **JavaScript** (for web-based AI tools). They should also be familiar with software engineering principles, such as **version control (Git)**, **unit testing**, and **debugging**.

(*ChatGPT*)

Motivatie

Critical thinking and evaluation: Students need to develop strong skills in:

- Evaluating AI-generated solutions
- Understanding limitations and potential biases of AI tools
- Making informed decisions about when to use AI versus traditional approaches

(Claude)

Balance AI with Fundamentals: Don't rush into using AI tools extensively before mastering introductory programming and problem-solving skills.

(Perplexity)

Integrate AI Tools (Cautiously): Explore how AI tools (like GitHub Copilot, but with careful consideration of its limitations) can be used to *assist* with coding, but emphasize that students must understand the underlying concepts. AI should be a tool, not a replacement for understanding. *(Gemini)*

Creativity: AI is a tool, but human creativity still drives innovation.

(ChatGPT)

https://docs.google.com/document/d/170dXZFtzBAqHieBpNXkUiQAxDircKW67JQo3gVlg1xs/edit?usp=sharing

Aspecte organizatorice

- Materiale
  - Platforma Moodle: http://edu.tuiasi.ro

- Titulari
  - Curs:
    - Simona Caraiman, simona.caraiman@academic.tuiasi.ro (cabinet AC: A3-1)
    - Otilia Zvorișteanu, otilia.zvoristeanu@academic.tuiasi.ro (cabinet AC: A3-5)
  - Laborator:
    - Otilia Zvorișteanu
    - Elena Maftei, elena-claudia.maftei@student.tuiasi.ro
    - Adrian Prodan, adrian.prodan@student.tuiasi.ro
    - Daniel Ignat, daniel-andrei.ignat@academic.tuiasi.ro
    - Iuliana Rusu, iuliana-elena.rusu@academic.tuiasi.ro
    - Irina Rotaru, irina.rotaru@academic.tuiasi.ro

Aspecte organizatorice

Evaluare

Laborator 50% (minim 5)

- 30% activitatea continua
- 70% test final (proba practica, L14)

Colocviu 50% (minim 5)

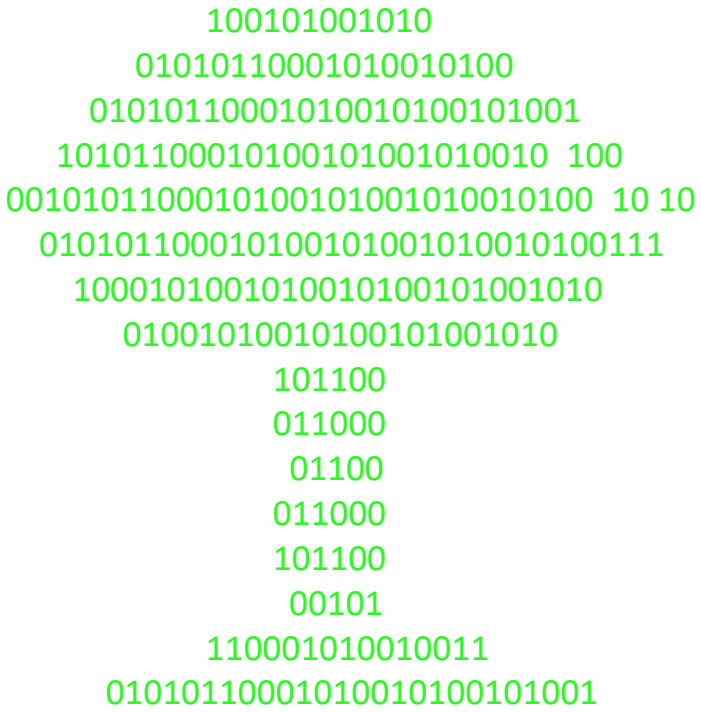
- 40% - test pe parcurs (Moodle) – *nu se reface in re-examinări*
- 60% - evaluare finala (proba scrisa, S14)

Bonusuri

- Participarea la programul Learn & Be Curious by Amazon
- Prezența la curs

Reguli

- Se vor utiliza doar adresele instituționale (@student.tuiasi.ro, @academic.tuiasi.ro) pentru conectare pe platformele digitale sau e-mailuri.
- Nu se admit absențe la laborator!
- Cunoștințele de la curs se acumulează înaintea fiecărei lucrări de laborator.
- Instrumentele bazate pe AI nu sunt interzise în procesul de învățare, dar nu vor fi accesibile la examen.
- Nimic din ce spuneți sau întrebați la curs nu va fi folosit împotriva voastră!
- Cine nu are nicio întrebare în timpul unui curs, înseamnă că a irosit 2 ore!!
- Nu contează ce gândiți, important e că gândiți!!!



Structuri de date și algoritmi

Principii de bază

- Fiecare structura de date si fiecare algoritm prezinta **costuri** si **beneficii**
- Notiunea de **compromis**: cost vs. beneficii
  - Ex.: o abordare uzuala este de a reduce timpul de executie cu pretul cresterii necesitatilor de stocare, sau vice-versa
- Programatorii trebuie sa stapaneasca notiunile fundamentale de structuri de date si algoritmi pentru a evita “reinventarea rotii”
- Structurile de date trebuie sa raspunda unor **necesitati**
  - Programatorii trebuie sa analizeze necesitatile unei aplicatii si apoi sa decida asupra structurilor de date utilizate

Obiectivele disciplinei

1. Prezentarea structurilor de date comune
2. Introducerea conceptului de compromis
   - fiecare structura de date are asociate costuri si beneficii
   - pentru fiecare structura de date vom descrie spatiul ocupat si timpul necesar efectuarii operatiilor tipice
3. Vom invata cum sa masuram eficienta unei structuri de date sau a unui algoritm
   - astfel veti putea determina care structura de date este cea mai potrivita pentru rezolvarea unei anumite probleme

Structuri de date

Definitie

- entitate destinată să stocheze și să organizeze/structureze o colecție de date

+

- metode de a crea, accesa, căuta și modifica datele

Structuri de date

Componente

<table>
<tr><td>1. Multimea de operatii pentru a manipula tipuri de date ce caracterizeaza obiecte abstracte</td><td>tip abstract de date (TAD)</td></tr>
<tr><td>2. Structura de stocare in care sunt memorate obiectele abstracte</td><td rowspan="2">implementarea structurii de date</td></tr>
<tr><td>3. Implementarea fiecarei operatii de la pct. 1, folosind structurile de stocare specificate la pct. 2</td></tr>
</table>

Structuri de date

Clasificare

- Statice vs. Dinamice
  - modificarea in timp a dimensiunii colectiei de date

- Omogene vs. Neomogene
  - Tipul de date

- Ordonate vs. Inlantuite
  - *Ordonata* (secventiala/implicita): exista o relatie de ordine exacta a elem.; zona fixa de locatii successive de memorie
  - *Inlantuita* (explicita): informatii de inlantuire continute de fiecare elem. (relatii explicite intre elem.)

Eficienta

- O solutie este eficienta daca rezolva problema data respectand constrangerile impuse resurselor consumate
  - Ex. de constrangeri pe resurse: spatiul de stocare disponibil (memorie, disk), timpul permis efectuarii fiecarui task
- Costul unei solutii = resursele consumate (spatiu + timp)

Eficienta

Pasi in alegerea unei structuri de date

1. Analiza problemei pt. a determina operatiile de baza ce trebuie suportate (ex. inserarea unui nou element, cautarea, stergerea unui element)

2. Cuantificarea constrangerilor de resurse asociate fiecarei operatii

3. Selectarea structurii de date care indeplineste cel mai bine aceste constrangeri

Costuri vs. Beneficii

- Fiecare structura de date necesita
  - Un spatiu de memorie pt. fiecare element stocat
  - Un anumit timp pt. efectuarea unei operatii de baza
  - Un anumit efort de programare

- Fiecare problema
  - Prezinta constrangeri legate de spatiul si timpul disponibile
  - Necesita utilizarea operatiilor de baza intr-o anumita proportie

Costuri vs. Beneficii

Exemplu

- Stocarea conturilor si realizarea de tranzactii intr-un sistem bancar
  - Perspectiva clientilor:
    - Deschiderea si inchiderea conturilor mai rare decat accesarea
    - Clientii sunt dispusi sa astepte mai multe minute pt deschidere/inchidere cont decat pt depozite/retragere numerar
  - Perspectiva bazei de date:
    - Operatii: inserare, stergere, cautare, actualizare
    - Tranzactiile la ATM nu modifica major baza de date (se modifica doar valoarea stocata intr-o inregistrare de tip cont bancar)
    - Adaugarea unui cont poate dura cateva minute
    - Inchiderea (stergerea) unui cont – fara constrangeri de timp

Costuri vs. Beneficii

Exemplu (cont.)

- Constrangeri asupra structurii de date
  - Nu conteaza costul stergerii unei inregistrari
  - Eficienta mare la cautarea si modificarea unei inregistrari
  - Eficienta moderata la inserarea unei inregistrari
  - Inregistrarile pot fi accesate printr-un numar de cont unic (*exact match query*)

- Structura de date potrivita: tabela de dispersie (hash table)
  - Permite cautarea exacta foarte rapida
  - O inregistrare poate fi modificata rapid
  - Permite inserarea eficienta de noi inregistrari
  - Si stergerile se pot efectua eficient, dar prea multe stergeri pot duce la degradarea performantei celorlalte operatii
    - Tabela poate fi reorganizata periodic (offline)

Masurarea eficientei unei solutii

Analiza asimptotica a algoritmilor

- Permite estimarea resurselor consumate de un algoritm
- Permite compararea costurilor relative a doi sau mai multi algoritmi care rezolva aceeasi problema

Concepte de baza:

- *Ordin de complexitate*
- *Rata de crestere* (rata cu care creste costul unui algoritm odata cu dimensiunea problemei)
- *Limite superioare si inferioare* pt. o rata de crestere

Masurarea eficientei unei solutii

Concepte de baza:

- *Modelul de calcul* specifica
  - *Operatii fundamentale* (elementare) pe care le utilizeaza algoritmul
  - *Costul* - in unitati abstracte de timp, asociat fiecarei operatii

  - *Ex:*
    - *Alg. numerici* – numarul de operatii aritmetice: cost (*, /) >> cost (+,-)
    - Alg. de sortare – nr. de comparatii
    - Alg. geometrie computationala – nr. de varfuri/muchii prelucrate

- Executie pe masini abstracte (modele de sist. de calcul)

Masurarea eficientei unei solutii

Concepte de baza:

- *Modelul de calcul*
  - *Costul* unei operatii – exprimat in unitati abstracte de timp = pasi
  - *Timpul de executie* – nr. total de pasi
    - Functie de marimea setului de date de intrare

Determinarea valorii maxime dintr-un vector

Algorithm ***arrayMax***(A, n)	# operatii
$currentMax \leftarrow A[0]$	2
for $(i = 1; i<n; i++)$	$2n$
	(i=1 once, i<n n times, i++ (n-1) times)
if $A[i] > currentMax$ **then**	$2(n-1)$
$currentMax \leftarrow A[i]$	$2(n-1)$
return $currentMax$	1
Total	$6n-1$

Masurarea eficientei unei solutii

Concepte de baza:

- *Masuri de complexitate*
  - Descriu aspectul de performanta

  - Timpul (spatiul) de executie in cazul cel mai defavorabil
  - Timpul (spatiul) de executie in cazul mediu
  - Timpul (spatiul) de executie in cazul cel mai favorabil

  Pt. ex. cautarii secventiale a unei valori intr-un vector de dimensiune n:
  - Cazul cel mai defavorabil: **T(n) = n**
  - Cazul mediu: **Tmed(n) ~= (n+1)/2**

Masurarea eficientei unei solutii

Concepte de baza:

- *Complexitatea algoritmilor*

  - Pt un alg. ce depinde de un nr. natural ***n***
    – conteaza doar tipul de dependenta matematica dintre nr. de operatii si ***n***
    – coeficientii conteaza mai putin

  Ordinul dependentei (ordinul algoritmului)

  Notatia “O(n)”

  ex. Nr. op. = $4n^3 + 2n^2 + n + 8$ → **$O(n^3)$**

Masurarea eficientei unei solutii

Notatia “O(n)”

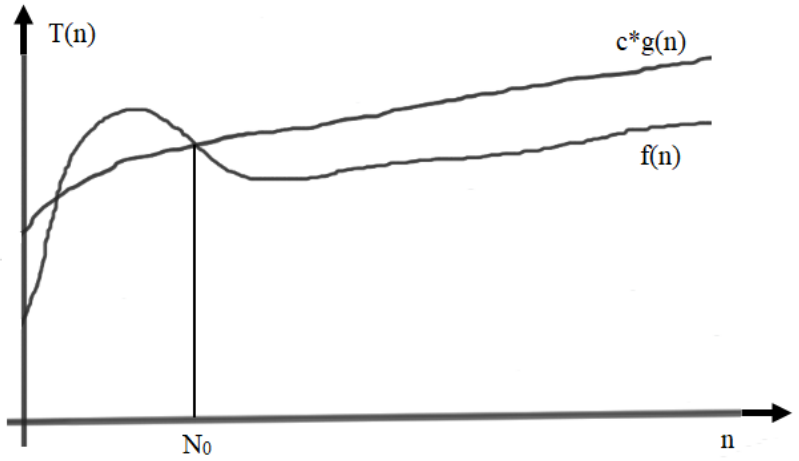
Definitie: Se considera doua functii f(n) si g(n) definite pe multimea numerelor naturale. Se spune ca f(n) este de ordin g(n) si se noteaza

$$f(n)=O(g(n))$$

daca exista doi intregi pozitivi c si N_0, a.î. f(n) <=c*g(n) pt. orice n>=N_0.

f(n) este asimptotic marginita de g(n).

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c$$

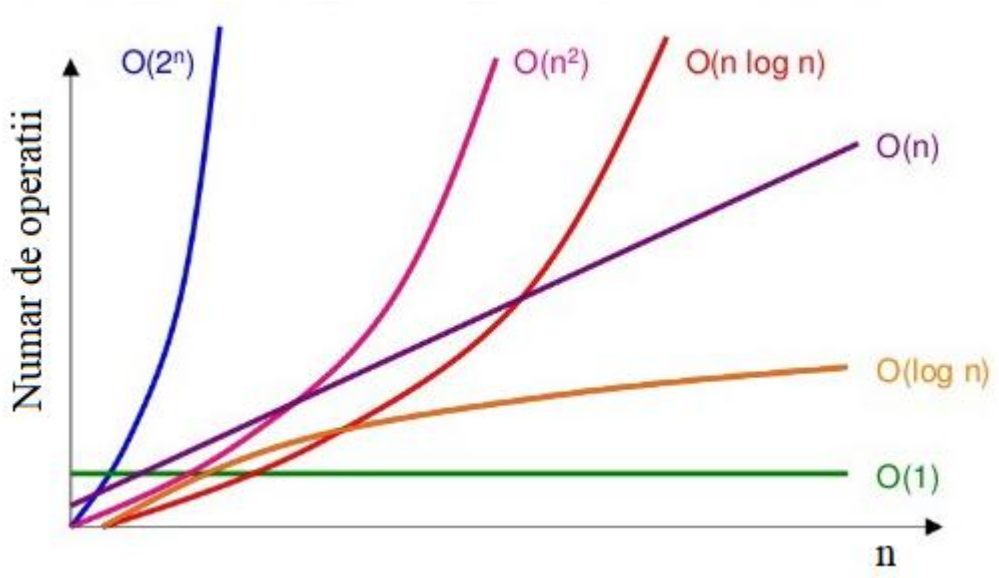


Masurarea eficientei unei solutii

Concepte de baza:

- *Complexitatea algoritmilor*

  - O(1) – clasa algoritmilor constanti
  - O(log *n*) – clasa alg. logaritmici
  - O(log^k *n*) – clasa alg. polilogaritmici
  - O(*n*) – clasa alg. liniari
  - O(n^2) – clasa alg. patratici
  - O(n^k) – clasa alg. polinomiali
  - O(2^n) – clasa alg. exponentiali



(preluat Thomas J. Cortina, Carnegie Mellon University)

Timpul de execuție (nr. pasi) vs. dimensiunea problemei pentru diverse ordine de complexitate

n	O(log n)	O(n)	O(n*log n)	O(n^2)	O(n^3)	O(2^n)
10	2.30	10	23.03	100	1000	1024
20	3.00	20	59.91	400	8000	1048576
30	3.40	30	102.04	900	27000	1073741824
40	3.69	40	147.56	1600	64000	1.0995E+12
50	3.91	50	195.60	2500	125000	1.1259E+15

Masurarea eficientei unei solutii

Proprietăți ale notației O:

- Dacă f(n) = c*g(n), atunci O(f(n)) = O(g(n)) (c – constantă nenulă)

  Coeficienții nenuli se ignoră
- Dacă f(n) = $a_0 + a_1*n + a_2*n^2 + \ldots + a_m*n^m$, atunci O(f(n)) = $O(n^m)$

  Ordinul de complexitate este dat de gradul polinomului
- Dacă f(n) = $f_1(n) + f_2(n) + \ldots + f_m(n)$ și $f_i(n) \leq f_{i+1}(n)$ $\forall$ i=1, 2, —, m, atunci O(f(n)) = $O(max(f_1(n), f_2(n), \ldots, f_m(n)))$

  Secvențe succesive
- Dacă f(n) = $f_1(n) * f_2(n) * \ldots * f_m(n)$, atunci O(f(n)) = $O(f_1(n)* f_2(n)* \ldots * f_m(n))$

  Secvențe imbricate
- Dacă f(n) = $\log_a n$ si g(n)=$\log_b n$, atunci O(f(n))=O(g(n))

  Toate funcțiile logaritmice au aceeasi rată de creștere (nu conteaza baza logaritmului: $\log_a b = \log_c b / \log_c a$)

Masurarea eficientei unei solutii

Exemple

Atentie la sintaxa! Nu întotdeauna un for este de complexitate O(n)

1. O buclă *for*

```
for (i = 0; i < n; i++) {
        S; //secventa de ordin O(1)
}
```

$\Rightarrow n*O(1) = O(n)$

2. Doua bucle *for*

```
for (i = 0; i < n; i++)
   for (j = 0; j < n; j++) {
        S; //secventa de ordin O(1)
}
```

$\Rightarrow n*n*O(1) = O(n^2)$

3. Doua bucle *for*

```
for (i = 0; i < n; i++)
   for (j = 0; j < i; j++) {
        S; //secventa de ordin O(1)
}
```

$\sum i = \frac{n(n+1)}{2} \Rightarrow O(n^2)$

Masurarea eficientei unei solutii

Exemple

Atentie la sintaxa! Nu întotdeauna un for este de complexitate O(n)

4. O buclă *while*

```
h=1;
while(h<=n){
        S; //secventa de ordin O(1)
        h = 2*h;
}
```

pas	1	2	3	4.....	k
h	2^1	2^2	2^3	2^4.....	$2^k \le n$

$$2^k \le n \Rightarrow k \le \log_2 n \Rightarrow O(\log n)$$

5. Bucla *while* ce contine bucla *for*

```
h=n;
while(h>10^-6) {
    for (i = 0; i < n; i++)
        S; //secventa de ordin O(1)
    h=h/2;
}
```

$$\left.\begin{array}{l} f_1 = O(g_1(n)) \\ f_2 = O(g_2(n)) \end{array}\right\} \Rightarrow f_1 \cdot f_2 = O(g_1(n) \cdot g_2(n))$$

$$\text{while} - O(\log n);\ \text{for} - O(n) \Rightarrow O(n \cdot \log n)$$

Masurarea eficientei unei solutii

Exemple

Sortarea prin inserţie (Insertion Sort)

```
INSERTION_SORT(A,n)
FOR k=2 TO n DO
      Temp ← A[k];
      i←k-1;
      WHILE(i>=1 and A[i]>temp) DO
            A[i+1] ← A[i]; i← i-1;
      END_WHILE
      A[i+1] ← temp;
END_FOR
END
```

Instrucţiunea WHILE poate fi înlocuită cu FOR după *j*, luând valori descrescător de la *i-1* până când *temp< A[j]* sau *j* a ajuns pe prima poziţie în vector.

Astfel, două bucle FOR imbricate $\Rightarrow O(n^2)$

Demonstrăm altfel

- Se iau în considerare toate operaţiile elementare
- În cazul cel mai defavorabil A[] este sortat descrescător

$$N_{operatii} = T(n) = \sum_{k=2}^{n}\left(3+\sum_{i=1}^{k-1}3\right) = \sum_{k=2}^{n}(3+3k-3) = 3\sum_{k=2}^{n}k = 3\left(\frac{n(n+1)}{2}-1\right) \Rightarrow O(n^2)$$

Masurarea eficientei unei solutii

Exemple

InsertionSort

```
Void InsrtSort(int a[], int n)
{
    int temp,k,i;
    assert(n>0);
    for(k=1;k<n;k++)
    {
        temp=a[k];
        for(i=k-1; i>=0 && temp<a[i]; i--)
        a[i+1]=a[i];
        a[i+1]=temp;
     }
}
```

Masurarea eficientei unei solutii

Exemple

Căutare binară (Binary Search)
Se caută valoare b în vectorul A[n]
st – limita din stânga
dr - limita din dreapta
m – mijlocul (st+dr)/2

varianta iterativă

```
BINARY_SEARCH(A,n,b)
st←1; dr←n;
WHILE(st<=dr) DO
        m←(st+dr)/2;
        IF A[m]=b  THEN RETURN m;
        ELSE
          IF A[m]>b THEN dr←m-1;
          ELSE st←m+1;
          END_IF
        END_IF
END_WHILE
RETURN 0
END
```

Se consideră $n=2^k$, k- numărul de înjumătăţiri

$k = \log_2 n \Rightarrow$ timpul de execuţie

$T(n) \leq \log_2 n + 1 \Rightarrow O(\log n)$

Masurarea eficientei unei solutii

Exemple

Varianta recursivă

```
Int binSearch(int a[],int b,int st,int dr)
{int m;
 assert(st<=dr);
 if(st= =dr)
        return(b= = a[dr]) ? dr: -1        //*1
 else {                                    //*2
        m=(st+dr)/2;
        if(b<=a[m])
                return binSearch(a,b,st,m);
        else
                return binSearch(a,b,m+1,dr);
        }
}
```

a- timpul pentru secvenţa *1
b- timpul pentru secvenţa *2

T(n) satisface relaţia de recurenţă

$$T(n) = \begin{cases} T(n/2) + a & \text{daca } n > 1 \\ b & \text{daca } n = 1 \end{cases}$$

se demonstrează prin inducţie că dacă T(n) satisface relaţia de recurenţă de mai sus atunci:

$$T(n) \leq a * \log(n) + b \Rightarrow O(\log n)$$

Masurarea eficientei unei solutii

Empiric (masurarea timpului de executie): util pentru a compara soluții alternative implementate pe același calculator, pentru același algoritm

C++: <chrono>

```
start = std::chrono::system_clock::now()
…
end = std::chrono::system_clock::now()
std::chrono::duration<double> elapsed_seconds = end-start;
elapsed_seconds.count()
```