2. Noțiunea de algoritm. Analiza algoritmilor

2.1. Noțiunea de algoritm

- Termenul **algoritm** își are sorgintea în numele autorului persan **Abu Ja'far Mohamed Ibn Musa Al Khowarismi** care în anul 825 î.e.n. a redactat un tratat de matematică în care prezenta pentru prima dată metode de rezolvare generice pentru anumite categorii de probleme.
- În activitatea de programare vom conveni ca prin **algoritm** să înțelegem o *"modalitate de rezolvare a unei probleme utilizând un sistem de calcul"*.
- Un **algoritm** este de fapt o **metodă** sau o **rețetă** pentru a obține un rezultat dorit.
- Un algoritm constă din:
  - (1) Un **set de date inițiale** care abstractizează contextul problemei de rezolvat
  - (2) Un **set de relații de transformare** care sunt operate pe baza unor reguli al căror conținut și a căror succesiune reprezintă însăși substanța algoritmului
  - (3) Un **set de rezultate preconizate** sau **informații finale,** care se obțin trecând de regulă printr-un șir de **informații** (rezultate) **intermediare**.
- Un algoritm se bucură de următoarele proprietăți:
  - **(1) Generalitate** - un algoritm **nu** rezolvă doar o anume problemă ci o clasă generică de probleme de același tip;
  - **(2) Finitudine** - informația finală se obține din cea inițială trecând printr-un număr finit de transformări;
  - **(3) Unicitate** - transformările și ordinea în care ele se aplică sunt univoc determinate de regulile algoritmului. Ori de câte ori se aplică același algoritm asupra aceluiași set de date inițiale se obțin aceleași rezultate.
- **Scopul capitolului**: analiza performanței algoritmilor.

2.2. Analiza algoritmilor

- La ce servește **analiza algoritmilor** ?
  - Permite **precizarea predictivă** a comportamentului algoritmului;
  - Prin analiză pot fi **comparați** diferiți algoritmi și poate fi astfel ierarhizată experiența și îndemânarea diverșilor producători [HS78].
- Analiza algoritmilor se bazează de regulă pe **ipoteze**:
  - (1) Sistemele de calcul sunt considerate **convenționale** adică ele execută câte **o singură instrucție** la un moment dat
  - **(2) Timpul total** de execuție al algoritmului rezultă din însumarea timpilor instrucțiilor individuale componente.

        - Desigur această ipoteză **nu** este valabilă la sistemele de calcul paralelele şi distribuite.
        - În astfel de cazuri aprecierea performanţelor se realizează de regulă prin **măsurători experimentale** şi **metode statistice**.
- În general, **analiza unui algoritm** se desfăşoară în două etape:
    - **(1) Analiza apriorică**
        - Constă în **aprecierea** din punct de vedere **temporal** a operațiilor care se utilizează şi a costului lor relativ.
        - Conduce de regulă la stabilirea **expresiei** unei **funcții** care **mărgineşte timpul de execuție al algoritmului**
    - **(2) Testul ulterior**
        - Constă în stabilirea unui număr suficient de **seturi de date inițiale** care să acopere practic **toate posibilitățile** de comportament ale algoritmului
        - **Testarea** comportamentul algoritmului pentru fiecare set în parte
        - Se finalizează prin culegerea unor date în baza cărora pot fi elaborate o serie de statistici referitoare la consumul de timp specific execuției algoritmului în cauză (**profilul algoritmului**).
- **Concluzie**:
    - **Analiza apriorică** are drept scop principal determinarea teoretică a **ordinului de mărime al timpului de execuție al unui algoritm**
    - **Testul ulterior** are ca scop principal determinarea efectivă a acestui ordin prin stabilirea **profilului algoritmului**

2.3. Notații asimptotice

- **Ordinul de mărime** al timpului de execuție al unui algoritm:
    - Reprezintă o măsură a eficienței algoritmului
    - Permite compararea relativă a variantelor de algoritmi.
- Studiul **eficienței asimptotice** a unui algoritm, se realizează utilizând mărimi de intrare cu dimensiuni suficient de mari pentru a face relevant numai **ordinul de mărime** al **timpului de execuție** al algoritmului.
- Interesează cu precădere limita la care tinde **timpul de execuție al algoritmului** odată cu **creşterea nelimitată** a dimensiunii intrării.
- De regulă, un algoritm care este *"asimptotic mai eficient"* decât alții, va constitui cea mai bună alegere şi pentru intrări de dimensiuni mici şi foarte mici.

2.3.1. Notația O (O mare)

- Notația O desemnează **marginea asimptotică superioară** a unei funcții. Pentru o funcție dată $g(n)$, se defineşte $O(g(n))$ ca şi mulțimea de funcții:

$O(g(n)) = \{ f(n)$: există constantele pozitive c şi n_0 astfel încât
$0 \le f(n) \le c \cdot g(n)$ pentru $\forall\ n \ge n_0 \}$ [2.3.2.a]

- Notația O se utilizează pentru a desemna o **margine superioară** a unei funcții în **interiorul** unui **factor constant**.

- Pentru toate valorile *n* superioare lui n_0, valoarea funcției *f(n)* este pe sau dedesubtul lui *g(n)* (fig.2.3.b).

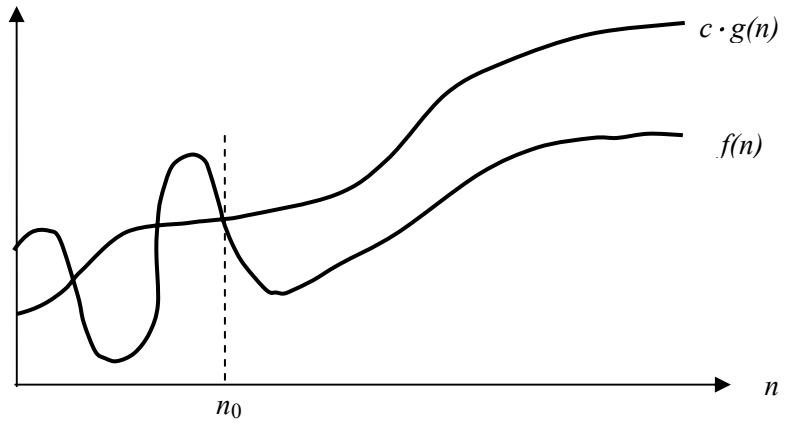

Fig.2.3.b. Reprezentarea lui *f(n)* = O*(g(n))*

- În baza formulei [2.3.2.a], se poate demonstra faptul că orice funcție pătratică $a \cdot n^2 + b \cdot n + c,\ a > 0$ este O(n^2). Pur şi simplu se alege valoarea constantei c astfel încât sa fie mai mare decît a. Termenii de grad mai mic ca 2 se neglijează.

    - Este surprinzător faptul că funcția liniara $a \cdot n + b$ este de asemenea O(n^2), lucru uşor de verificat în baza aceleaşi formule [2.3.2.a], dacă se alege $c = a + |b|$ şi $n_0 = 1$.

- Notația O este de obicei cea mai utilizată în aprecierea **timpului de execuție al algoritmilor** respectiv a **performanței** acestora**.**

- Ea poate fi uneori apreciată direct din **inspectarea** structurii algoritmului, spre exemplu existența unei bucle duble conduce imediat la o margine de ordinul O(n^2)

- Deoarece notația O descrie o **margine superioară**, când este utilizată pentru a mărgini **cazul cel mai defavorabil** de execuție al unui algoritm, prin implicație ea **mărgineşte superior** comportamentul algoritmului în aceeaşi măsură pentru **orice** intrare.

- Cele mai obişnuite ordine de mărime ale notației O se află în relațiile de ordine prezentate în [2.3.2.c], iar reprezentarea lor grafică la scară logaritmică apare în fig.2.3.c.

$$O(1) < O(\log n) < O(n) < O(n \cdot \log n) < O(n^2) < O(n^3) < < O(2^n) < O(10^n) < O(n!) < O(n^n) \qquad [2.3.2.c]$$

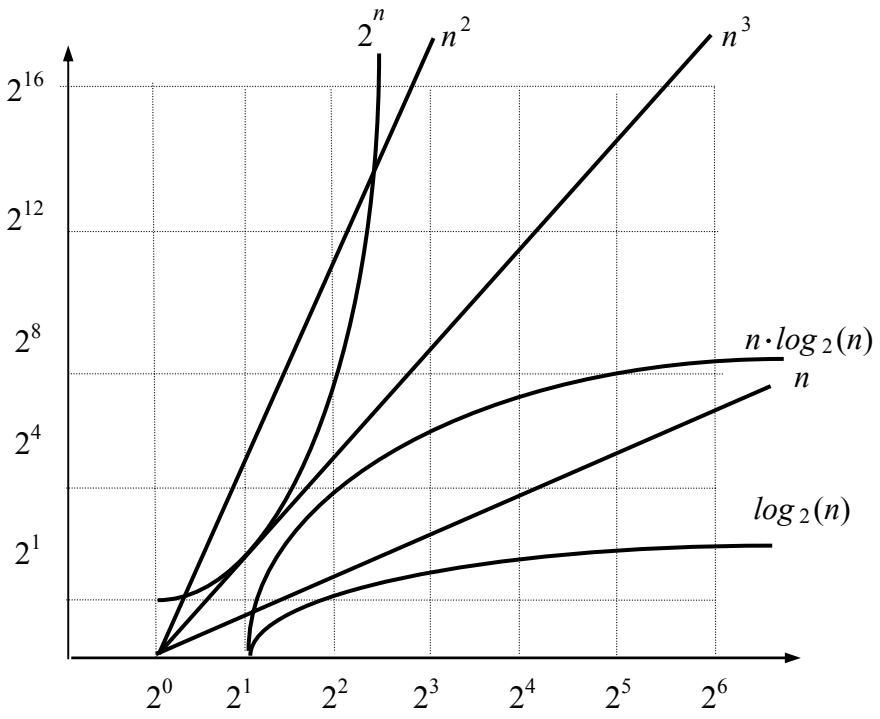

Fig. 2.3.c. Ordine de mărime ale notației O

- În același context în [2.3.2.d] se prezintă câteva **sume întregi** utile care sunt frecvent utilizate în **calculul complexității algoritmilor**.

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2} = \mathrm{O}(n^2)$$

$$\sum_{i=1}^{n} i^2 = \frac{n \cdot (n+1) \cdot (2 \cdot n+1)}{6} = \mathrm{O}(n^3)$$

$$\sum_{i=1}^{n} i^3 = \frac{n^2 \cdot (n^2+1)}{4} = \mathrm{O}(n^4)$$

[2.3.2.d]

$$\sum_{i=1}^{n} i^k = \frac{n^{k+1}}{k+1} + \cdots = \mathrm{O}\left(\frac{n^{k+1}}{k+1}\right) = \mathrm{O}(n^{k+1})$$

$$\sum_{i=2}^{n} (i-1) = \sum_{i=1}^{n-1} i$$

2.3.2. Aritmetica ordinală a notației O

- În vederea aprecierii complexității algoritmilor în raport cu notația O, a fost dezvoltată o **aritmetică ordinală** specifică care se bazează pe o serie de **reguli formale** [De89].

- Aceste reguli sunt prezentate în continuare.

- **Regula 1**. $O(k) < O(n)$ pentru $\forall\, k$.
- **Regula 2**. Ignorarea constantelor: $k \cdot f(n) = O(f(n))$ pentru $\forall\, k$ şi $\forall\, f$ sau $O(k \cdot f) = O(f)$
- **Regula 3**. Tranzitivitate: dacă $f(n) \to O(g(n))$ şi $g(n) \to O(h(n))$ atunci $f(n) \to O(h(n))$

Demonstraţie:

$f(n) = O(g(n)) \Rightarrow \exists\ c_1, n_1 \quad f(n) \le c_1 \cdot g(n)$ pentru $\forall\, n > n_1$
$g(n) = O(h(n)) \Rightarrow \exists\ c_2, n_2 \quad g(n) \le c_2 \cdot h(n)$ pentru $\forall\, n > n_2$

- Se alege $n_3 = max\{n_1, n_2\}$.
- În relaţia $f(n) \le c_1 \cdot g(n)$ se înlocuieşte $g(n)$ cu expresia de mai sus. Se obţine relaţia:
  $f(n) \le c_1 \cdot (c_2 \cdot h(n))$
- Alegând $c_3 = c_1 \cdot c_2$ se obţine $f(n) \le c_3 \cdot h(n)$ pentru $\forall\, n > n_3$ deci $f(n) = O(h(n))$.

- **Regula 4**. $f(n) + g(n) = O(max\{f(n), g(n)\})$
- **Regula 5**. Dacă $f_1(n) = O(g_1(n))$ şi $f_2(n) = O(g_2(n))$ atunci $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

- Utilizând aceste reguli se poate calcula o **estimare** a lui O pentru o expresie aritmetică dată, **fără** a avea valori explicite pentru k şi n.

- **Exemplul 2.3.2.** Se consideră expresia
  $8 \cdot n \cdot log(n) + 4 \cdot n^{3/2}$
  şi se cere o estimare a sa în termenii notaţiei O.

- Se procedează după cum urmează:
  - $log(n) = O(n^{1/2})$ deoarece logaritmul unui număr este mai mic decât orice putere pozitivă a numărului în baza următoarei **teoreme**:
    - Fie $a > 0$ şi $a \neq 1$;
    - Pentru $\forall$ exponent $d > 0$ există un număr N astfel încât pentru $\forall\, x > N$ avem $log_a x < x^d$.
  - $n \cdot log(n) = O(n \cdot n^{1/2}) = O(n^{3/2})$ (Regula 5).
  - $8 \cdot n \cdot log(n) = O(n^{3/2})$ (Regula 2).
  - $4 \cdot n^{3/2} = O(n^{3/2})$ (Regula 2).
  - $8 \cdot n \cdot log(n) + 4 \cdot n^{3/2} = O(max\{8 \cdot n \cdot log(n), 4 \cdot n^{3/2}\}) =$
    $= O(max\{n^{3/2}, n^{3/2}\}) = O(n^{3/2})$ (Regula 4).
- Rezultă în urma estimării că expresia $8 \cdot n \cdot log(n) + 4 \cdot n^{3/2}$ este $O(n^{3/2})$.

2.4. Aprecierea timpului de execuție

- Cu ajutorul notației O mare se poate aprecia **timpul de execuție** al unui algoritm.
- Se face sublinierea că se poate aprecia timpul de execuție al unui **algoritm abstract** şi **nu** al unui **program** întrucât acesta din urmă depinde de mai mulți factori:
  - (1) Dimensiunea şi natura datelor de intrare
  - (2) Caracteristicile sistemului de calcul pe care se rulează programul
  - (3) Eficiența codului produs de compilator.
- Notația O mare permite **eliminarea** factorilor care **nu** pot fi controlați (spre exemplu viteza sistemului de calcul) concentrându-se asupra comportării algoritmului **independent** de program.
- În general un algoritm a cărui complexitate temporală este $O(n^2)$ va rula ca şi program în $O(n^2)$ unități de timp indiferent de limbajul sau sistemul de calcul utilizat.
- În aprecierea timpului de execuție se porneşte de la **ipoteza simplificatoare** deja enunțată, că fiecare **instrucție** utilizează în medie aceeaşi cantitate de timp.
- Instrucțiunile care **nu** pot fi încadrate în această medie de timp sunt:
  - (1) Instrucțiunea **IF**
  - (2) Secvențele repetitive (buclele)
  - (3) Apelurile de proceduri şi funcții.
- Presupunând pentru moment că apelurile de proceduri şi funcții se ignoră, se adoptă prin **convenție** următoarele simplificări:
  - (1) Se presupune că o instrucțiune **IF** va consuma întotdeauna timpul necesar execuției ramurii celei mai lungi, dacă nu există rațiuni contrare justificate;
  - (2) Se presupune că întotdeauna instrucțiunile din interiorul unei bucle se vor executa de numărul maxim de ori permis de condiția de control.

- **Exemplul 2.4.a.**
- Se consideră o procedură care:
  - (1) Caută într-un tablou `a` cu `n` elemente, un element egal cu `cheie`
  - (2) Returnează în variabila `unde` ultima locație în care este găsită cheia sau zero în caz contrar [2.4.a].

```
-------------------------------------------------------------
PROCEDURE CautareLiniara(n,cheie: integer,
                a: TablouNumeric; VAR unde: integer);
VAR indice: integer;
  BEGIN
    unde: = 0
    FOR indice:= 1 TO n DO
      IF a[indice]= cheie THEN                    [2.4.a]
         unde:= indice
  END; {CautareLiniara}
-------------------------------------------------------------
```

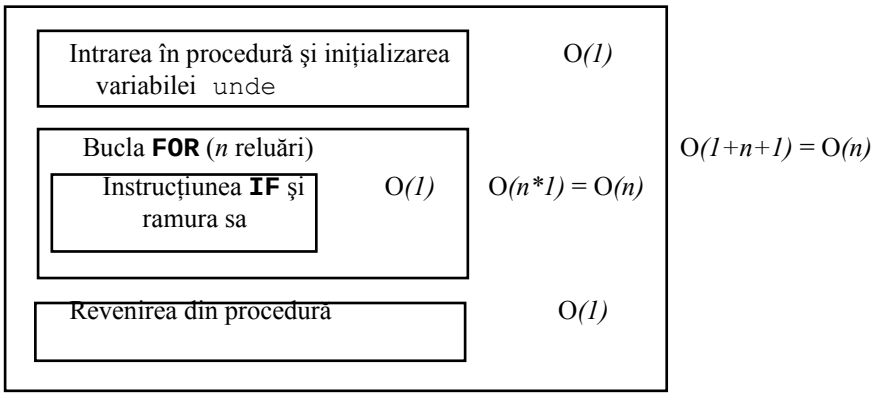

Fig. 2.4.a. Schema temporală a algoritmului [2.4.a]

- În figura 2.4.a apare **schema temporală** a algoritmului împreună cu estimarea O mare corespunzătoare.

- În analiză au fost introduse şi operaţiile de intrare şi revenire din procedură care **nu** apar ca părţi explicite ale rutinei.

- Pe viitor apelul şi revenirea din procedură **vor fi omise** din analiza temporală deoarece ele contribuie cu un **timp constant** la execuţie şi **nu** influenţează estimarea O mare

- **Exemplul 2.4.b**. Se consideră următoarea porţiune de cod [2.4.b].

```
-----------------------------------------------------
FUNCTION SumProd(n: integer): integer;
  VAR rezultat,k,i: integer;
  BEGIN
    rezultat:= 0;
    FOR k:= 1 TO n                                  [2.4.b]
      FOR  i:= 1 TO k
         rezultat:= rezultat+k*i;
    SumProd:= rezultat
  END;{SumProd}
-----------------------------------------------------
```

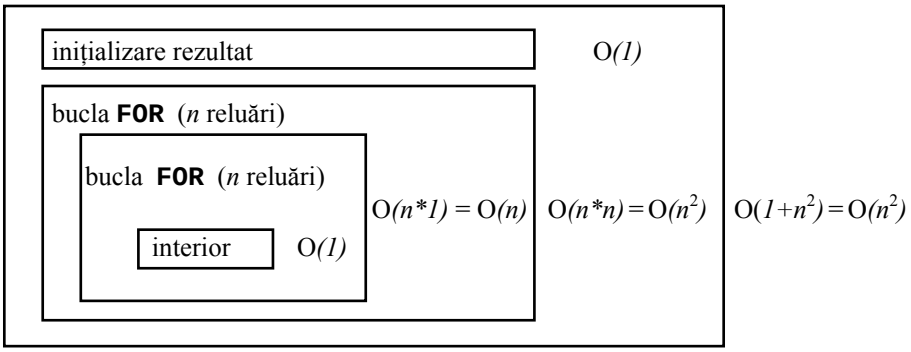

Fig. 2.4.b. Schema temporală a algoritmului din secvenţa [2.4.b]

- La analiza complexităţii temporale se face presupunerea că ambele bucle se reiau de **numărul maxim** de ori posibil

    - În realitate acest lucru **nu** este adevărat deoarece bucla interioară se execută cu limita $k \leq n$ (şi nu cu limita n)

- Se va **demonstra** că prin această simplificare estimarea temporală finală **nu** este afectată.

- La o analiză mai aprofundată se ține cont de faptul că bucla **FOR** interioară se execută prima oară o dată, a doua oară de 2 ori ş.a.m.d, a *k*-oară de *k* ori.

    - În consecință numărul exact de reluari este cel precizat de expresia de mai jos şi după cum se observă el este $O(n^2)$

$$1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2} = O(n^2) \qquad [2.4.c]$$

- În mod analog în cadrul secvenței [2.4.d] se ajunge la estimarea: $O(n \cdot n \cdot n) = O(n^3)$.

```
FOR i:= 1 TO n DO
  FOR j:= 1 TO i DO
    FOR k:= 1 TO i DO                                    [2.4.d]
      Secvență {O(1)};
```

- Calculul **exact** al numărului de reluări conduce la valoarea precizată de relația [2.4.e], ținând cont de faptul că cele două bucle interioare se execută de i^2 ori la fiecare reluare a buclei exterioare **FOR**.

$$1^2 + 2^2 + 3^2 + \ldots + n^2 = \frac{n \cdot (n+1) \cdot (2 \cdot n + 1)}{6} = O(n^3) \qquad [2.4.e]$$

2.5. Profilarea unui algoritm

- Presupunând că un algoritm a fost conceput, implementat, testat şi depanat pe un sistem de calcul țintă

- Ne interesează de regulă **profilul performanței** sale, adică **timpii precişi** de execuție ai algoritmului pentru diferite seturi de date, eventual pe diferite sisteme țintă.

- Pentru aceasta sistemul de calcul țintă trebuie să fie dotat cu un **ceas intern** şi cu **funcții** sistem de acces la acest ceas.

- După cum s-a mai precizat, determinarea profilului performanței face parte din **testul ulterior** al unui algoritm şi are drept scop determinarea precisă a **ordinul de mărime** al timpului de execuție al algoritmului.

- Informațiile rezultate sunt utilizate de regulă pentru a valida sau invalida, respectiv pentru a nuanța rezultatele estimării apriorice.

- Se presupune un algoritm implementat în forma unui program numit `Algoritm(X: Intrare,Y: Iesire)` unde `X` este intrarea iar `Y` ieşirea.

- Pentru a construi **profilul algoritmului** este necesar să fie concepute:

    - (1) Seturile de **date de intrare** a căror dimensiune creşte între anumite limite, pentru a studia comportamentul algoritmului în raport cu **dimensiunea intrării**

- (2) Seturile de date care în principiu se referă la **cazurile extreme** de comportament.

- (3) O **procedură** cu ajutorul căreia poate fi construit profilul algoritmului în baza seturilor de date anterior amintite.

```
------------------------------------------------------------
procedure Profil;
{procedura construieşte profilul procedurii Algoritm(X:
Intrare,Y: Ieşire)}
begin
    *initializează procedura Algoritm;
    *afiseaza("Testul lui Algoritm. Timpii în milisecunde");
     repeat
        *citeste(SetDeDate);                        [2.5.a]
        *afiseaza("Un nou set de date:", SetDeDate);
        *apel TIME(t);  {atribuie lui t valoarea curentă
                         a ceasului sistem}
        *apel Algoritm(SetDeDate, Rezultate);
        *apel TIME(t1);
        *afiseaza("TimpExecuţie=", t - t1);
     until sfârşit(SetDeDate)
end {Profil}
------------------------------------------------------------
```

- Procedura `Profil` poate fi utilizată în mai multe **scopuri** funcție de obiectivele urmărite.

  - (1) Evidențierea **performanței intrinseci** a unui algoritm precizat.

    - Pentru aceasta se aleg, aşa cum s-a mai precizat, seturi de date cu dimensiuni din ce în ce mai mari.

    - Rezultatul va fi **profilul** algoritmului.

    - Pentru deplina conturare a profilului se testează de asemenea şi cazurile de **comportament extrem** ale algoritmului respectiv cazul cel mai favorabil şi cel mai defavorabil.

  - (2) Evidențierea **performanței relative** a doi sau mai mulți **algoritmi diferiți** care îndeplinesc aceeaşi sarcină.

    - În acest scop se execută procedura `Profil` pentru fiecare din algoritmi în parte, cu aceleaşi seturi de date inițiale, pe un acelaşi sistem de calcul.

    - Compararea profilelor rezultate permite **ierarhizarea performanțelor** algoritmilor analizați.

  - (3) Evidențierea **performanței relative** a două sau mai multe **sisteme de calcul.**

    - În acest scop se rulează procedura `Profil` pentru un acelaşi algoritm, cu aceleaşi date inițiale pe sistemele de calcul țintă supuse analizei.

- Compararea profilelor rezultate permite **ierarhizarea performanțelor** sistemelor de calcul analizate.

- De regulă, pe piața sistemelor de calcul se utilizează în acest scop algoritmi consacrați, în anumite cazuri standardizați, cunoscuți sub denumirea de "**bench marks**" în baza cărora sunt evidențiate performanțele diferitelor arhitecturi de sisteme de calcul.

- Trebuie însă acordată o **atenție specială** procedurii `TIME` a căror rezultate în anumite circumstanțe pot fi nerevelevante.

    - Astfel, în cazul sistemelor **time-sharing**, al **sistemelor complexe** care lucrează cu întreruperi sau al **sistemelor multi-procesor**, timpul furnizat de această funcție poate fi complet needificator.

    - În astfel de cazuri, una din soluții este aceea de a executa programul de un număr suficient de ori şi de a apela la **metode statistice** pentru evidențierea performanțelor algoritmului testat.

2.6. Rezumat

- Prin **algoritm** se înțelege o modalitate de rezolvare a unei probleme utilizând un sistem de calcul. Un **algoritm** este de fapt o **metodă** sau o **rețetă** pentru a obține un rezultat dorit.
- Din punct de vedere formal în definirea unui algoritm se porneşte de la un **set de date inițiale** care abstractizează contextul problemei de rezolvat, asupra căruia se aplică un **set de relații de transformare**
- Un algoritm se bucură de următoarele **proprietăți**: **generalitate, finitudine** şi **unicitate**
- **Analiza** unui algoritm se desfăşoară în două etape: (1) analiza apriorică care are drept scop principal determinarea teoretică a **ordinului de mărime al timpului de execuție al unui algoritm** şi (2) testul posterior are ca scop principal stabilirea **profilului algoritmului**.
- Pentru aprecierea ordinului de mărime al timpului de execuție al unui algoritm se utilizează notațiile asimptotice
- Notația O este o notație asimptotică care desemnează **marginea asimptotică superioară** a unei funcții. Notația O este utilizată în aprecierea **timpului de execuție** al algoritmilor. Ea poate fi uneori apreciată din inspectarea structurii algoritmului. Pentru a simplifica această activitate se utilizează regulile **aritmeticii ordinale a notației O**
- **Aprecierea timpului execuție al unui algoritm** se realizează de regulă analizând schema sa temporală în termenii notației O.
- **Profilarea unui algoritm** se realizează pe baza măsurării timpilor de execuției ai algoritmului pentru diverese seturi de date.

2.7. Exerciții

1) Definiți *noțiunea de algoritm*. Care sunt *elementele constitutive* ale unui algoritm?
2) Care sunt *proprietățile* de care se bucură un algoritm?
3) La ce serveşte *analiza algoritmilor*? Care sunt *etapele analizei* unui algoritm?
4) Cum se defineşte *notația* O?
5) Care este *relația de ordonare* a celor mai obişnuite ordine de mărime al e notației O?
6) Care sunt *regulile aritmeticii ordinale* a notației O?
7) Se cere să se estimeze în termenii funcției O următoarea expresie: $3n^2+5n+25n*\log(n)$
8) Se cere să se estimeze în termenii notaației O timpul de execuție al algoritmuli de căutare liniară respectiv al algoritmului de căutare binară.
9) Ce este *profilarea unui algoritm*? Schițați o funcție cu ajutorul căreia se poate construi profilul unui algoritm oarecare. Care sunt *etapele construcției* profilului unui algoritm?