12

PROIECTAREA ALGORITMILOR

Adrian Runceanu

Curs 12

Metoda Divide et Impera

Conţinutul cursului

12.1. Prezentarea generala a metodei

12.2. Implementari ale metodei

12.3. Probleme propuse spre rezolvare

12.1. Prezentarea generala a metodei

- O scurta introducere in istorie:
- Expresia *Divide et impera* este atribuita lui Filip al II-lea (rege al Macedoniei (382-336 IC) descriind politica sa asupra oraselor state-grecesti.
- **Dezbină si stăpâneşte** reprezinta, din punct de vedere politic si sociologic, o combinaţie de tactici (politice, militare sau economice) prin care se urmareste castigarea si mentinerea puterii prin divizarea unei populatii in entitati mai mici care luate separat au putere mai mica decat cele care sunt unite, impunandu-si astfel puterea.
- *Aceasta tactica are rezultate atunci cand cei cu mai putina putere si influenta doresc sa-si impuna puterea asupra celor care, daca s-ar uni, ar avea o putere mare.*

12.1. Prezentarea generala a metodei

- In informatica, aceasta strategie reprezinta o metoda de rezolvare a problemelor.
- *Ideea de baza consta in impartirea unei probleme in 2 sau mai multe subprobleme care se rezolva separat, apoi se trece la combinarea rezultatelor problemelor rezolvate obtinandu-se, astfel, solutia finala.*

- La baza problemelor rezolvabile prin aceasta metoda sta urmatorul enunt:
- Se da un sir de valori (secventa de valori) a_1, a_2, a_3,…, a_n.
- Aceasta secventa trebuie prelucrata.
- *Prelucrarea se va realiza in felul urmator:*
  1. Sirul se imparte in 2 sau mai multe subsiruri
  2. Fiecare subsir se va imparti, dupa aceiasi metoda, in 2 sau mai multe subsiruri pana cand se ajunge la o problema rezolvabila sau un rezultat cunoscut
  3. Din aproape in aproape, prin combinarea rezultatelor obtinute, se obtine rezultatul final

12.1. Prezentarea generala a metodei

Presupunem că pentru orice p, q $\in$ N, 1 $\leq$ p < q $\leq$ n, ($\exists$) m din mulţimea { p,...,q-1 } astfel încât:

- prelucrarea secvenţei { a_p,...,a_q }, se face
- prelucrând secvenţele vecine următoare {a_p,..., a_m}, { a_{m+1},...,a_q }
- şi apoi combinând rezultatele pentru a obţine prelucrarea întregii secvenţe { a_p,...,a_q }.

12.1. Prezentarea generala a metodei

- Următoarea procedură scrisa in pseudocod, exemplifică metoda ***Divide et Impera***.

- Dacă dimensiunea problemei iniţiale sau a subproblemelor aparute este mai mică decât o valoare ε, atunci problema se rezolvă direct prin procedura SOL, soluţia fiind în vectorul a.

- Soluţia se pune în vectorul a, iar *soluţiile parţiale se pun în vectorii b, respectiv c*.

12.1. Prezentarea generala a metodei

- Procedura DIVIDE împarte secvenţa în două subsecvenţe $\{ a_p, \ldots, a_m \}$ şi $\{ a_{m+1}, \ldots, a_q \}$, obţinând rezultatul în m.
- *Prin cele două autoapelări ale procedurii DIVIDE_IMPERA se rezolvă subproblemele punând rezultatele prelucrărilor în b şi respectiv în c.*
- Procedura COMB obţine din soluţiile subproblemelor, soluţia problemei date.
- La început procedura DIVIDE_IMPERA se apelează cu p=1 şi q=n.

12.1. Prezentarea generala a metodei

```
procedure DIVIDE_IMPERA(p, q, a)
begin
        if q-p≤ε then
                SOL(p, q, a)
        else
        begin
                DIVIDE(p, q, m)
                DIVIDE_IMPERA(p, m, b)
                DIVIDE_IMPERA(m+1, q, c)
                COMB(b, c, a)
        end
end
```

12.1. Prezentarea generala a metodei

Exemple de probleme rezolvabile prin aceasta metoda:

- ✓ Calculul sumei/produsului elementelor unui sir
- ✓ Determinarea minimului/maximului dintr-un sir
- ✓ Sa se verifice anumiti termeni din sir care au o anumita proprietate data (sunt pare / sunt prime / sunt pozitive, etc)
- ✓ Sortarea elementelor dintr-un sir de valori

Subprogram Divide_Et_Impera(inceput, sfarsit, rezultat)

/* inceput - pozitia primului termen din sir/subsir;

sfarsit - pozitia ultimului termen din sir/subsir */

1. Daca < s-a terminat divizarea sirului in unu sau doi termeni > **atunci**

1.1. Prelucrez_secventa_divizata(inceput, sfarsit, rezultat)

altfel

1.2. Divizez_secventa(inceput, sfarsit, m) /* nu in toate problemele m reprezinta pozitia elementului de la mijloc */

1.3. Divide_Et_Impera(inceput, m, rezultat1)

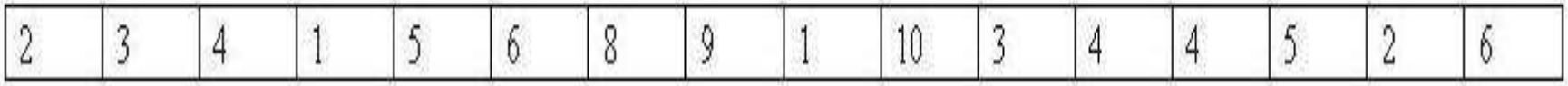
1.4. Divide_Et_Impera(m, sfarsit, rezultat2)

1.5. Combin_rezultate(rezultat1, rezultat2, rezultat)

Sfarsit subprogram

12.1. Prezentarea generala a metodei

De exemplu dorim sa realizam *suma numerelor dintr-un vector dat cu n elemente numere intregi*.



Se imparte vectorul in doi vectori:

Fiecare se dintre cei doi vectori se poate imparti in continuare in alti doi vectori:

La fel se procedeaza in continuare:

Apoi:

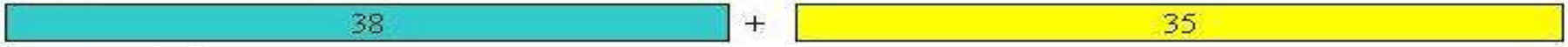
Dat fiind ca s-au obtinut secvente de vector de lungime 1, nu se mai realizeaza descompunerea.

Se compun solutiile subsecventelor si se determina solutiile corespunzatoare:

Si in continuare:

Apoi:

La fel:



La sfarsit:

Conţinutul cursului

12.1. Prezentarea generala a metodei

12.2. Implementari ale metodei

12.3. Probleme propuse spre rezolvare

12.2. Implementari ale metodei

- Prezentăm în continuare metoda Divide et Impera aplicată pentru a sorta un vector A cu n elemente prin ***interclasare***(Mergesort).

- *Descompunerea unui vector în alţi doi vectori ce urmează a fi sortaţi, are loc până când avem de sortat vectori cu maxim două componente.*

12.2. Implementari ale metodei

1. Procedura SORT ordonează crescător un vector de maxim doua elemente şi corespunde procedurii SOL din schema generală.
2. Procedura INTERCLASARE interclasează rezultatele şi corespunde procedurii COMB din schema generală.
3. *Procedura DIVIDE_IMPERA implementează strategia generala a metodei.*
4. Procedura DIVIDE este realizată prin instrucţiunea:

m <- [(p + q) / 2]

12.2. Implementari ale metodei

Procedurile implementate in pseudocod:

```
procedure SORT(p, q, A)
begin
    if a_p > a_q then
                begin
                        t <- a_p
                        a_p <- a_q
                        a_q <- t
                end
end
```

12.2. Implementari ale metodei

```
procedure DIVIDE_IMPERA(p, q, A)
begin
   if q – p ≤ 2 then SORT(p, q, A)
                else
                begin
                        m=[( p + q ) / 2]
                        DIVIDE_IMPERA(p, m, A)
                        DIVIDE_IMPERA(m+1, q, A)
                        INTERCLASARE(p, q, m, A)
                end
end
```

```
procedure INTERCLASARE(p, q, m, A)
vector A, B
    /* B - vectorul in care se construieste vectorul ordonat prin interclasarea celor 2 subvectori */
    /* introducem in B elementele din cei 2 subvectori in ordine crescatoare */
begin
    i<-p, j<-m+1, k<-1
    while (i≤m) and (j≤q)
    begin
        if a_i≤a_j then /* daca elementul curent din primul subvector este mai mare decat
    elementul curent din cel de-al doilea subvector, se va introduce in vectorul intermediar cel
    din cel de-al doilea subvector, altfel se va introduce cel din primul subvector */
        begin
                b_k<-a_i
                i<-i+1
                k<-k+1
        end
        else
        begin
                b_k<-a_j
                j<-j+1
                k<-k+1
        end
end
```

```
/*introducem in b elementele ramase in subvectorul din care nu s-au copiat toate
    elementele*/
        if i≤m then
                for j=i to m do
                begin
                        b_k<-a_j
                        k<-k+1
                end
        else
                for i=j to q do
                begin
                        b_k<-a_i
                        k<-k+1
                end
/*copiem elementele vectorului intermediar in vectorul initial - cel care va avea
    elementele sortate*/
        k<-1
        for i=p to q do
                begin
                        a_i<- b_k
                        k<=k+1
                end
end
```

12.2. Implementari ale metodei

Programul principal:

```
begin
   for i=1 to n do read(A_i)
   DIVIDE_IMPERA(1, n, A)
   for i=1 to n do write(A_i)
end.
```

Implementarea metodei:

```
#include <iostream.h>
int n,i,p,q,a[100];
void sort(int p,int q,int a[100])
{
    int t;
    if(a[p]>a[q])
    {
        t=a[p];          a[p]=a[q];     a[q]=t;
    }
    return;
}
```

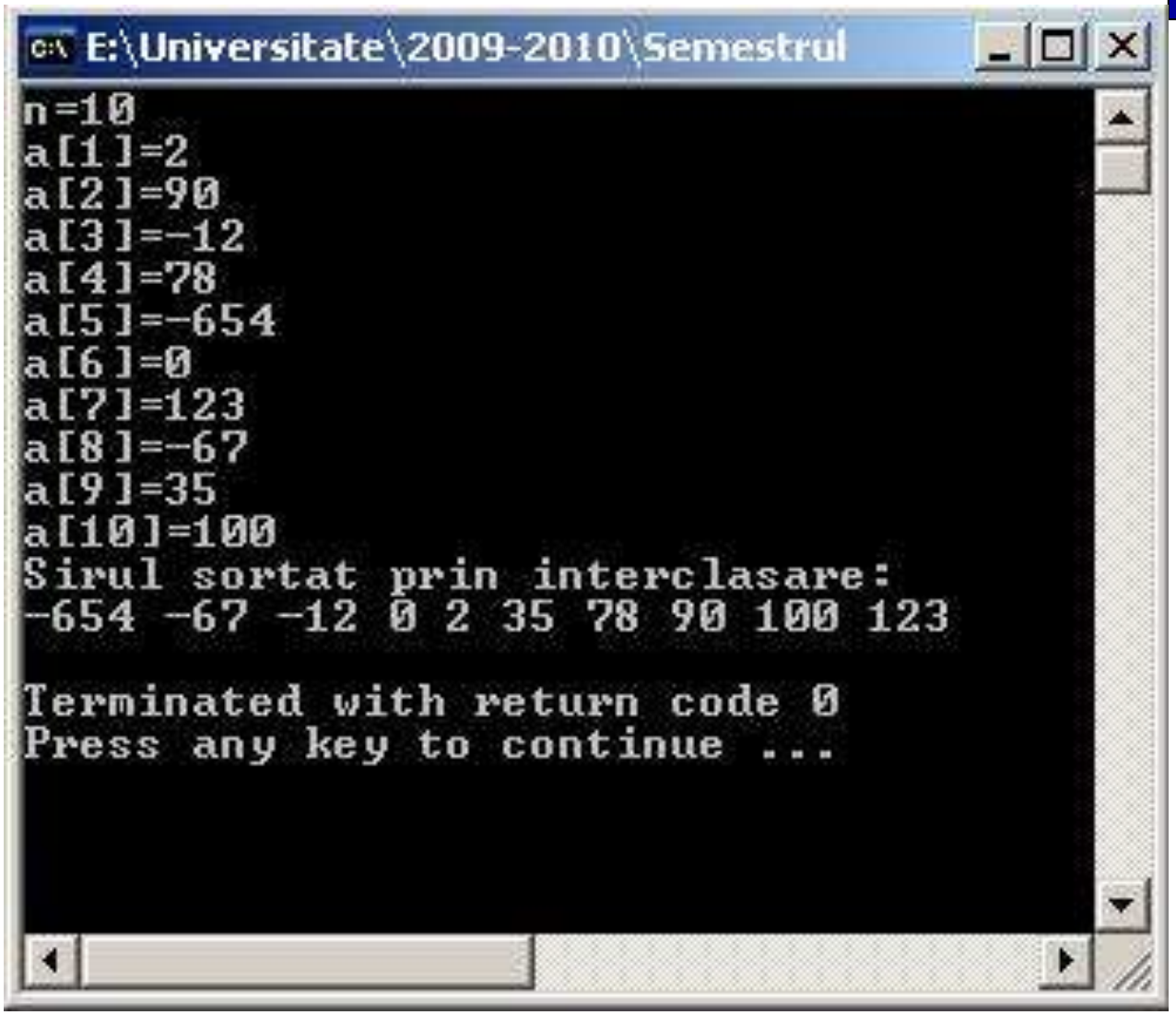
```
void interclasare(int p,int q,int m, int a[100])
{
   int i, j, k, b[100];
   i=p;
   j=m+1;
   k=1;
   while( (i<=m) && (j<=q) )
    if(a[i]<=a[j])
    {
        b[k]=a[i];      i++;    k++;
    }
    else
    {
        b[k]=a[j];      j++;    k++;
    }
```

```
if(i<=m)
    for(j=i;j<=m;j++)
    {
            b[k]=a[j];
            k++;
    }
    else
    for(i=j;i<=q;i++)
    {
            b[k]=a[i];
            k++;
    }
```

```
        k=1;
        for(i=p;i<=q;i++)
         {
                a[i]=b[k];
                k++;
        }
        return;
}
```

```
void divide_impera(int p,int q, int a[100])
{
   int m;
   if(q-p<=2) sort(p,q,a);
   else
   {
        m=(p+q)/2;
        divide_impera(p,m,a);
        divide_impera(m+1,q,a);
        interclasare(p,q,m,a);
   }
   return;
}
```

```
int main()
{
   cout<<"Dati numarul de elemente ";          cin>>n;
   cout<<"\nDati elementele vectorului \n";
   for(i=1;i<=n;i++)
   {
        cout<<"a["<<i<<"]= ";
        cin>>a[i];
   }
   divide_impera(1,n,a);
   cout<<"Sirul sortat prin interclasare: ";
   for(i=1;i<=n;i++) cout<<a[i]<<" ";
}
```

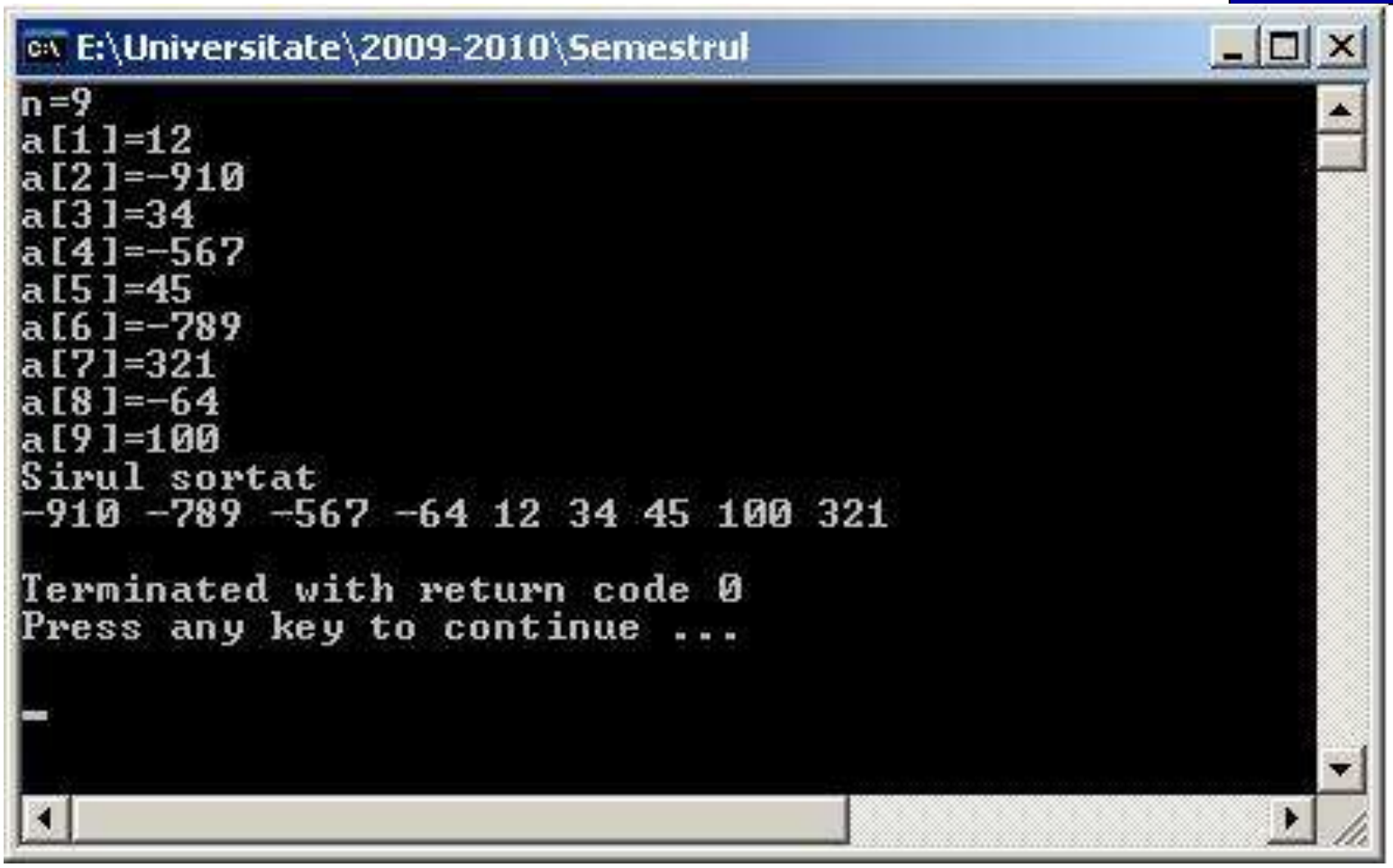

E:\Universitate\2009-2010\Semestrul
n=10
a[1]=2
a[2]=90
a[3]=-12
a[4]=78
a[5]=-654
a[6]=0
a[7]=123
a[8]=-67
a[9]=35
a[10]=100
Sirul sortat prin interclasare:
-654 -67 -12 0 2 35 78 90 100 123
Terminated with return code 0
Press any key to continue ...

- Algoritmul de la Sortare rapida(quickSort) se bazeaza pe metoda Divide et Impera.
- *Imparte*: Sirul de pe pozitiile p...u este impartit (rearanjat) in doua siruri nevide de elemente.
  - Elementele celor 2 subsiruri se gasesc pe pozitiile p...m si respectiv m+1...u.
  - Primul sir are proprietatea ca fiecare element din primul sir este mai mic decat orice element din al doilea sir.
- *Stapaneste*: Cele doua siruri sunt ordonate prin apeluri succesive ale algoritmului de sortare rapida (quickSort)
- *Combina*: Deoarece cele doua siruri sunt sortate pe loc, nu este nevoie de nicio ordonare.

```
#include <iostream.h>
int a[50], n, i;
void quickSort (int a[ ], int
    p, int u)
{
   int i = p, j = u;
   int m;
   int pivot = a[(p + u) / 2];

   while (i <= j)
   {
      while (a[i] < pivot)   i++;
      while (a[j] > pivot)   j--;
      if (i <= j)
       {
            m=a[i];
            a[i] =a[j];
            a[j] = m;
            i++;
            j--;
       }
   }
   if (p < j) quickSort(a, p, j);
   if (i < u) quickSort(a, i, u);
}
```

```
int main()
{
    int n, i;
    cout<<"n="; cin>>n;
    for(i=1;i<=n;i++)
    {
        cout<<"a["<<i<<"]=";
        cin>>a[i] ;
    }
    quickSort(a,1,n);
    cout<<"Sirul sortat : "<<endl;
    for(i=1;i<=n;i++)
        cout<<a[i]<<" ";
}
```


E:\Universitate\2009-2010\Semestrul
n=9
a[1]=12
a[2]=-910
a[3]=34
a[4]=-567
a[5]=45
a[6]=-789
a[7]=321
a[8]=-64
a[9]=100
Sirul sortat
-910 -789 -567 -64 12 34 45 100 321
Terminated with return code 0
Press any key to continue ...

12.2. Implementari ale metodei

CAUTAREA BINARĂ

Fie un vector v cu n *elemente ordonate crescător* şi un număr nr.

Să se caute acest număr în vector şi în caz că este găsit să se afiseze indicele pe care se găseşte.

12.2. Implementari ale metodei

Paşi:

1. verificam dacă x =v$\left[\frac{n}{2}\right]$, adică dacă este egal cu valoarea din mijlocul vectorului. (Atenţie: în C operatorul "*/*" are ca rezultat *împărţirea întreagă*, adică partea întreagă a lui n/2.).

2. Dacă da atunci funcţia se opreşte cu succes. *Am găsit elementul x pe poziţia n/2*

12.2. Implementari ale metodei

3.dacă nu

- dacă x < $v\left[\frac{n}{2}\right]$ cautam în jumătatea $\left[v[0], v\left[\frac{n}{2}-1\right]\right]$
- dacă x > $v\left[\frac{n}{2}\right]$ cautam în jumătatea $\left[v\left[\frac{n}{2}+1\right], v[n-1]\right]$

- căutarea în unul dintre cele două intervale se face în mod similar: comparam cu jumătatea intervalului.

- Dacă elementul este egal cu x, ne oprim, dacă nu verificam în care parte se poate afla x.

Recursivitatea: de fapt, la fiecare pas cautam în intervalul cuprins între poziţia i şi poziţia j (iniţial i = 0 şi j = n).

(1) Dacă i>j

algoritmul se încheie cu insucces (s-a căutat în tot vectorul şi nu s-a găsit x)

(2) Dacă i<=j atunci

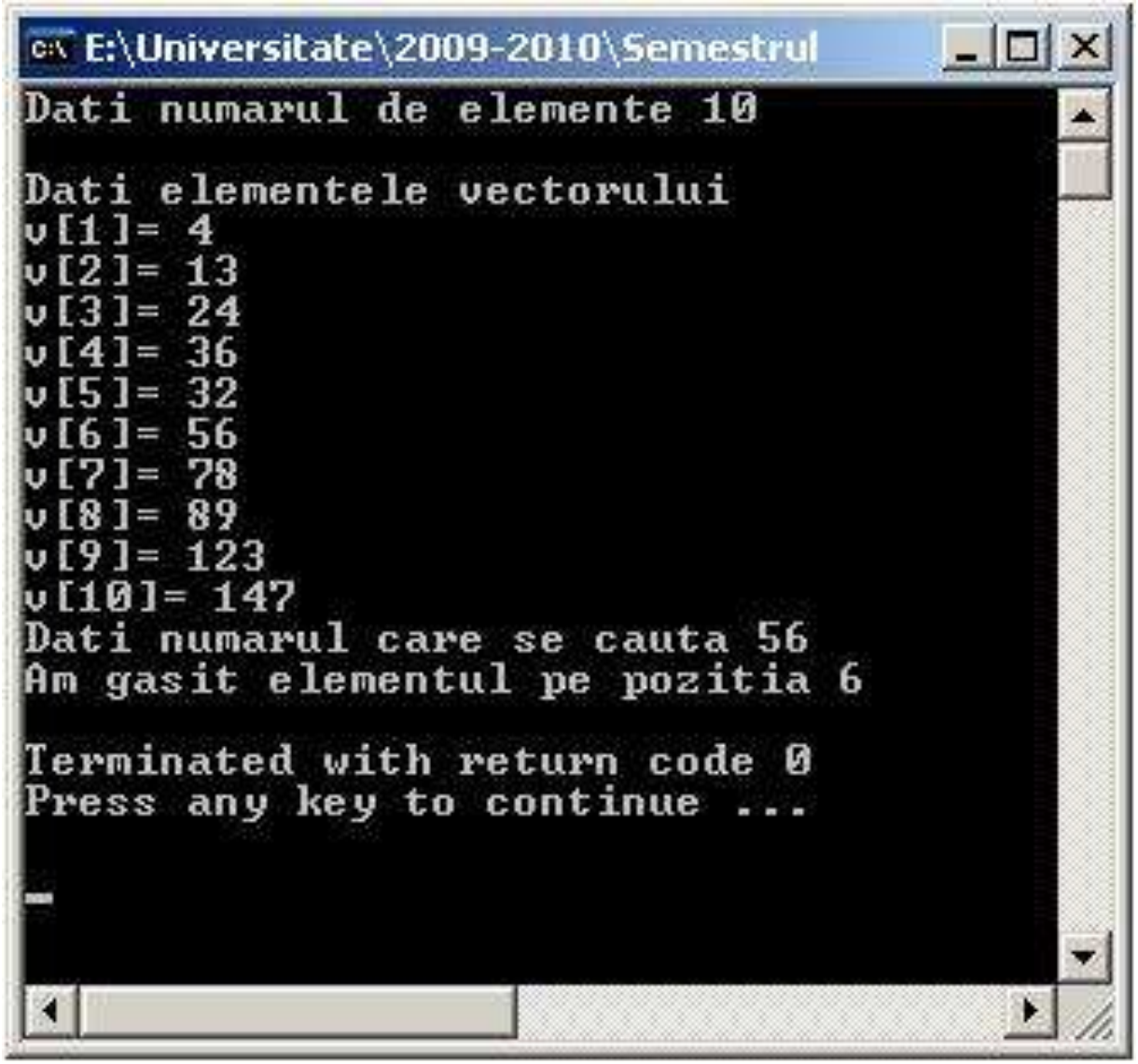
- dacă x = $v\left[\frac{i+j}{2}\right]$- funcţia se opreşte cu succes
- altfel $v\left[\frac{i+j}{2}\right]$compar x cu. Dacă x e mai mare cautam la dreapta lui $v\left[\frac{i+j}{2}\right]$, altfel caut la stânga lui $v\left[\frac{i+j}{2}\right]$.

```
#include<iostream.h>

int n,i,nr,v[100];

int caut(int i,int j)
{
   if(nr==v[(i+j)/2]) return ((i+j)/2);
   else
     if(i<j)
       if(nr < v[(i+j)/2]) return(caut(i, (i+j)/2 - 1));
                          else  return(caut((i+j)/2 + 1, j));
}
```

```
int main(void)
{
   cout<<"Dati numarul de elemente ";cin>>n;
   cout<<"\nDati elementele vectorului \n";
   for(i=1;i<=n;i++)
   {
        cout<<"v["<<i<<"]= ";
        cin>>v[i];
   }
   cout<<"Dati numarul care se cauta ";
   cin>>nr;
   cout<<"Am gasit elementul pe pozitia
   "<<caut(1,n);
}
```


E:\Universitate\2009-2010\Semestrul
Dati numarul de elemente 10
Dati elementele vectorului
v[1]= 4
v[2]= 13
v[3]= 24
v[4]= 36
v[5]= 32
v[6]= 56
v[7]= 78
v[8]= 89
v[9]= 123
v[10]= 147
Dati numarul care se cauta 56
Am gasit elementul pe pozitia 6
Terminated with return code 0
Press any key to continue ...

12.2. Implementari ale metodei

TURNURILE DIN HANOI

Se dau trei tije numerotate cu A, B, C şi n discuri perforate având diametre diferite.

Iniţial toate discurile sunt asezate pe tija A, în ordinea crescătoare a diametrelor lor, considerând sensul de la vârful tijei la baza ei.

Să se mute toate discurile pe tija B în aceeaşi ordine, folosind tija C şi respectând următoarele reguli:

- la fiecare pas se mută un singur disc
- în permanenţă pe fiecare tijă deasupra fiecărui disc pot apare numai discuri de diametre mai mici.

12.2. Implementari ale metodei

Rezolvarea acestei probleme se bazeaza pe urmatoarele considerente logice:

- daca n=1, atunci mutarea este imediata A→B (mutam discul de pe A pe B)
- daca n=2, atunci sirul mutarilor este: A→C, A→B, C→B
- daca n>2 procedam astfel:
  - mutam (n-1) discuri A→C
  - mutam un disc A→B
  - mutam cele (n-1) discuri C→B

12.2. Implementari ale metodei

Observam ca problema initiala se descompune in trei subprobleme mai simple, similare problemei initiale:

- mutam (n-1) discuri A → C
- mutam ultimul disc pe B
- mutam cele (n-1) discuri C → B

Dimensiunile acestor subprobleme sunt: n-1, 1, n-1.

- Aceste subprobleme sunt independente, deoarece tija initiala (pe care sunt dispuse discurile), tija finala si tija intermediare sunt diferite.

12.2. Implementari ale metodei

Notam:

H(n,A,B,C) = sirul mutarilor a n discuri de pe A pe B, folosind C.

PENTRU

n=1 A→B

n>1 H(n,A,B,C) = H(n-1,A,C,B), A→B, H(n-1,C,B,A)

Implementarea solutiei:

```
#include <iostream.h>
int n;
void hanoi(int n, char a, char b, char c)
{
   if(n==1) cout<<a<<" -> "<<b<<endl; /* daca avem un singur disc pe tija A il mutam pe tija B */
   else
   {
        hanoi(n-1, a, c, b); /* mutam n-1 discuri de pe tija A, pe tija C folosind ca tija intermediara tija B */
        cout<<a<<" -> "<<b<<endl; /* mutam discul ramas de pe tija A pe tija B */
        hanoi(n-1, c, b, a); /* mutam cele n-1 discuri de pe tija C, pe tija B folosind ca tija intermediara tija B */
   }
}
```

12.2. Implementari ale metodei

```
int main()
{
    cout<<"Numarul de discuri pe tija A = ";
    cin>>n;
    cout<<"Mutarile sunt urmatoarele:\n";
    hanoi(n, 'A', 'B', 'C'); /* mutam n discuri de
   pe tija A, pe tija B folosind tija intermediara C
   - A,B,C sunt numele discurilor */
}
```


E:\Universitate\2009-2010\Semestrul
Numarul de discuri pe tija A = 3
Mutarile sunt urmatoarele:
A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
Terminated with return code 0
Press any key to continue ...

Conţinutul cursului

12.1. Prezentarea generala a metodei

12.2. Implementari ale metodei

12.3. Probleme propuse spre rezolvare

1. Sa se determine maximul (minimul) a n numere intregi.
2. Sa se determine cel mai mare divizor comun a n valori dintr-un vector.
3. Sa se caute o valoare intr-un vector. Daca se gaseste se va afisa pozitia pe care s-a gasit, altfel se va afisa un mesaj.
4. Sa se numere cate valori sunt egale cu x dintr-un sir de numere intregi citite.
5. **Ghicirea unui număr ascuns**

   Fie următorul joc: avem un număr x natural între 0 şi 3000 care este ascuns de o persoană. O altă persoană va trebui să găsească numărul ascuns prin încercări cât mai puţine, pe baza informaţiilor date de persoana care a ascuns numărul, aceasta precizând dacă numă ascuns este mai mare sau mai mic decât numărul presupus.
6. Fiind dat un sir ordonat, sa se scrie un program recursiv care realizeaza cautarea binara, prin impartirea multimii initiale in doua multimi care contin 1/3 respectiv 2/3 din elementele multimii initiale.

Întrebări?