1. Căutarea binară

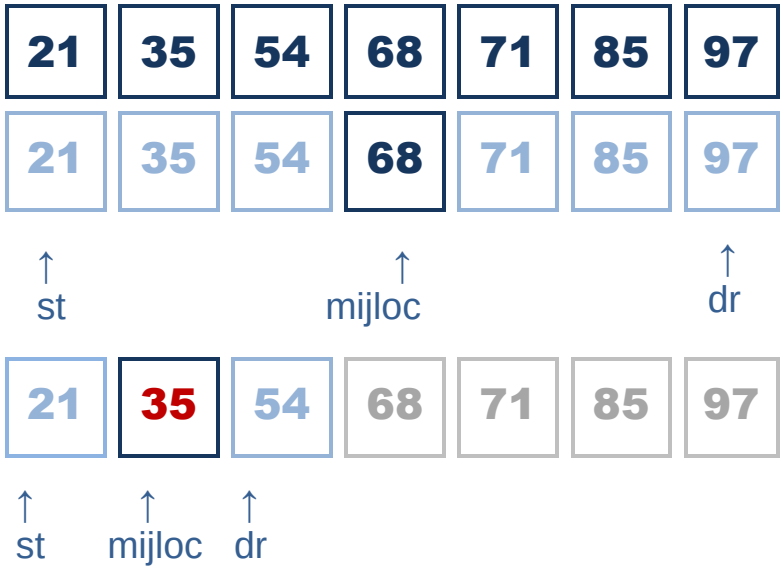
Algoritm:

- Este un algoritm de căutare folosit pentru a găsi un element într-un tablou unidimensional (vector) ordonat.
- Algoritmul funcţionează pe baza tehnicii *divide et impera*. Valoarea căutată este comparată cu cea a elementului din mijlocul vectorului.
  - Dacă este egală cu cea a acelui element, algoritmul se termină.
  - Dacă este mai mare decât acea valoare, algoritmul se reia, de la mijlocul listei până la sfârşit.
  - Dacă este mai mică decât acea valoare, algoritmul se reia pentru elementele de la începutul listei până la mijloc.
- Deoarece la fiecare pas cardinalul mulţimii de elemente în care se efectuează căutarea se înjumătăţeşte, algoritmul are complexitate logaritmică.

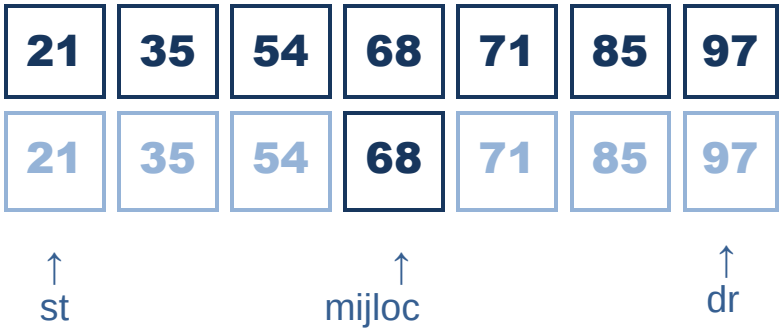
Exemplu:

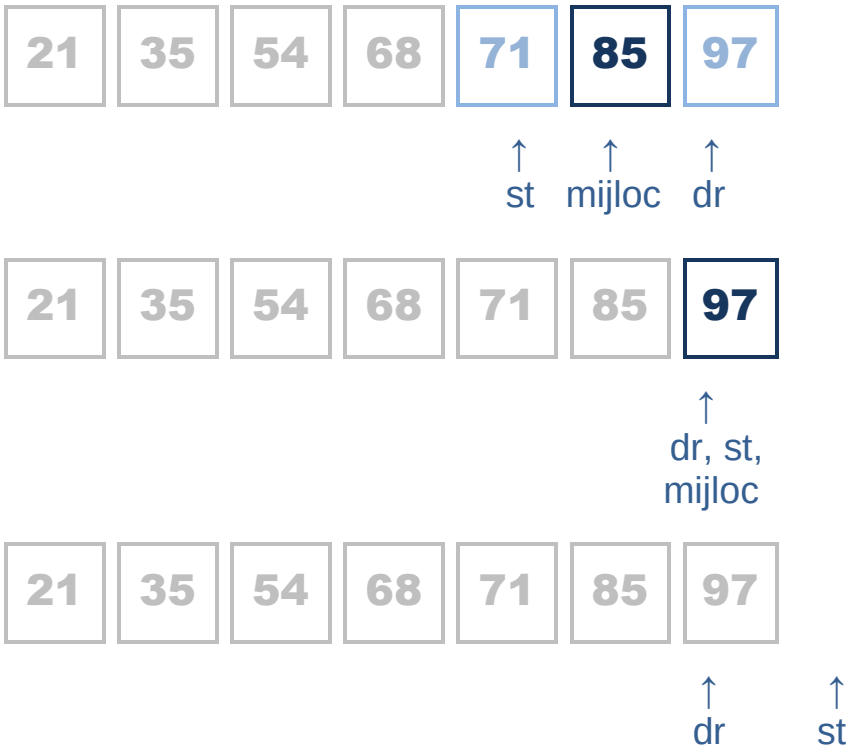
Se dau următoarele numere: 21, 35, 54, 68, 71, 85 şi 97.

a. Se caută numărul 35.



b. Se caută numărul 105.





Problemă:

Se dă un vector cu *n* elemente numere întregi distincte, sortat crescător. Să se afişeze dacă un element *x*, introdus de la tastatură, se găseşte în vector.

```
import java.io.*;
class Caut1
{
        //introducere un string de la tastatura
        public static String getString() throws IOException
        {
                InputStreamReader sir=new InputStreamReader(System.in);
                BufferedReader br=new BufferedReader(sir);
                String s=br.readLine();
                return s;
        }

       // introducere integer de la tastatura
        public static int getInt() throws IOException
        {
                String s=getString();
                return Integer.parseInt(s);
        }

        //cautare binara iterativa
        public static boolean caut_bin_i(int caut,int[] v, int n)
        {
             boolean rezultat=false;
             int mijloc, st=0,dr=n-1;
                do{
                        mijloc =(st+dr)/2;
```

```
                    if(v[mijloc]==caut)
                    rezultat=true;
                  else
                    if(v[mijloc]<caut)
                       st= mijloc +1;
                    else
                      dr= mijloc -1;
          } while (st<=dr && rezultat==false);
    return rezultat;
}

//cautare binara recursiva
public static boolean caut_bin_r(int caut,int[] v, int st, int dr)
{
       int mijloc;
       if (st<=dr)
       {
          mijloc =(st+dr)/2;
          if(v[mijloc]==caut)
             return true;
          else
                  if(v[mijloc]<caut)
                        return caut_bin_r(caut,v, mijloc +1, dr);
                         //este in jumatatea superioara
                  else
                        return caut_bin_r(caut,v,st, mijloc -1);
                         //este in jumatatea inferioara
          }
        else
         return false;
}

public static void main(String[] args) throws IOException
{
          int[] v=new int[100];
          int n, i, caut;
          System.out.println("Dati nr. de elem ale vectorului ");
          n=getInt();
          System.out.println("Dati elem. vectorului, in ordine crescatoare");
          System.out.print("v[0]=");
          v[0]=getInt();
          for (i=1;i<n;i++)
          {
              do {
                System.out.print("v["+i+"]=");
                      v[i]=getInt();
              }while(v[i]<v[i-1]);
          }

          System.out.println("Dati nr. cautat ");
          caut=getInt();
```

```
            boolean gasit;
            //cautare binara iterativa
            System.out.println(" ");
            System.out.println("Cautare binara iterativa");
            gasit=caut_bin_i(caut,v,n);
            if (gasit)
              System.out.println("Am gasit "+ caut);
            else
              System.out.println("Nu am gasit " + caut);

           //cautare binara recursiva
            System.out.println(" ");
            System.out.println("Cautare binara recursiva");
            gasit=caut_bin_r(caut,v,0,n-1);
            if (gasit)
              System.out.println("Am gasit "+ caut);
            else
              System.out.println("Nu am gasit " + caut);
        }
}
```

2. Sortarea prin interschimbare directă - *BubbleSort*

Algoritm:

Este cea mai simplă metodă de sortare a unui vector.

Pentru i fixat, se compară v[j] cu v[j+1] pentru j = 0, 1, . . ., i.

Dacă nu se respectă condiția de sortare crescătoare v[j]<=v[j+1] atunci se realizează interschimbul.

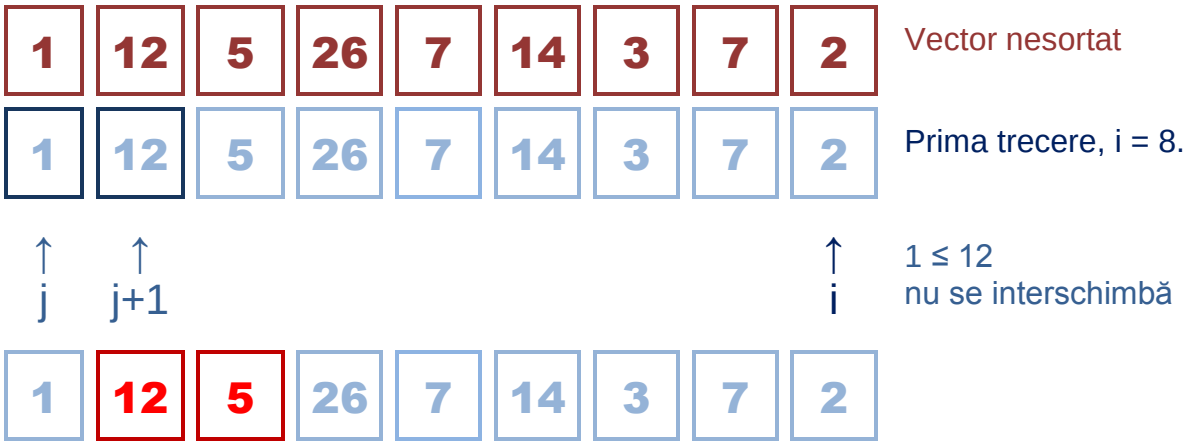
La sfârșitul primei etape, pe locul v[n] ajunge cel mai mare element (i = n).

După a doua trecere, pe locul v[n-1] va ajunge cel mai mare element dintre cele rămase (i=n-1).

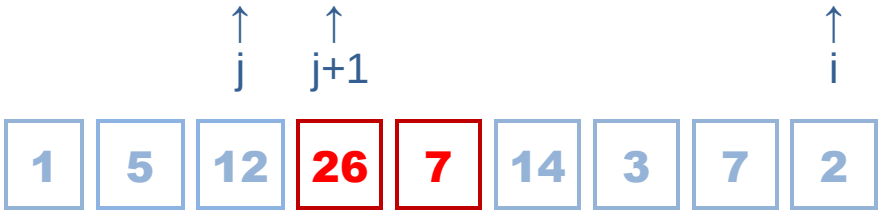
Se procedează analog până se ajunge la primul element, care nu se mai compară cu nimic, fiind cel mai mic.

Numărul de interschimbări este n(n-1)/2, în cazul cel mai defavorabil, atunci când șirul inițial este sortat descrescător.

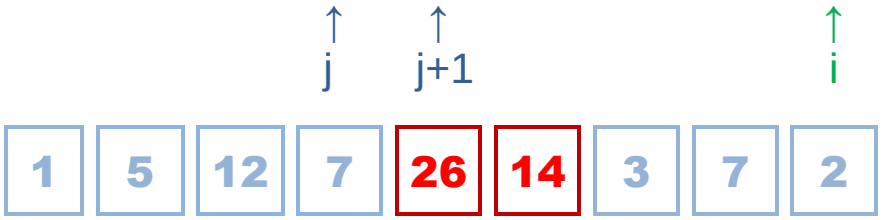
Exemplu:



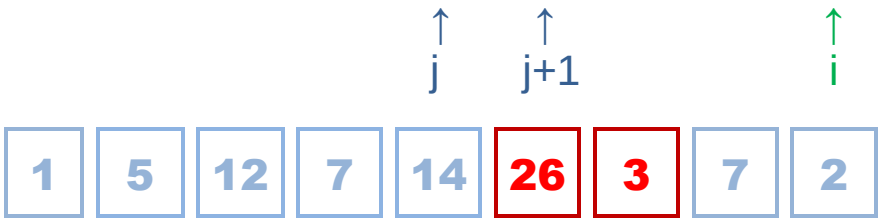
12 ≥ 5
se interschimbă



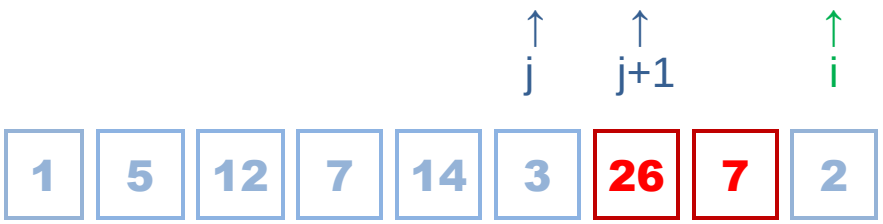
12 ≤ 26
nu se interschimbă



26 ≥ 7
se interschimbă



26 ≥ 14
se interschimbă



26 ≥ 3
se interschimbă

26 ≥ 7
se interschimbă

26 ≥ 2
se interschimbă

s-a terminat prima trecere; ultimul element este pe poziţia finală

...

Vectorul sortat

Problemă:

Se dă un vector cu *n* elemente numere întregi. Să se sorteze crescător acest tablou folosind metoda sortării prin interschimbare directă.

```
import java.io.*;
class Sort2
{
        //introducere un string de la tastatura
        public static String getString() throws IOException
        {
                InputStreamReader sir=new InputStreamReader(System.in);
                BufferedReader br=new BufferedReader(sir);
                String s=br.readLine();
                return s;
        }

       // introducere integer de la tastatura
        public static int getInt() throws IOException
        {
                String s=getString();
                return Integer.parseInt(s);
        }

        //sortare prin interschimbare directa
        public static void sortare(int[] v, int n)
        {
                int i, j,aux;
                for(i=n-1; i>=0; i--)      //ciclu exterior (in ordine descrescatoare)
                  for(j=0;j<i;j++)         //ciclu interior (in ordine crescatoare)
                    if(v[j]>v[j+1])         //interschimbare?
                    {
                        aux=v[j];          //se realizeaza interschimbarea
                        v[j]=v[j+1];
                        v[j+1]=aux;
                    }
        }

        public static void main(String[] args) throws IOException
        {
                int[] v;
                v=new int[100];
                int n, i, caut;
                System.out.println("Dati nr de elem ale vectorului ");
                n=getInt();
                System.out.println("Dati elem vectorului");
```

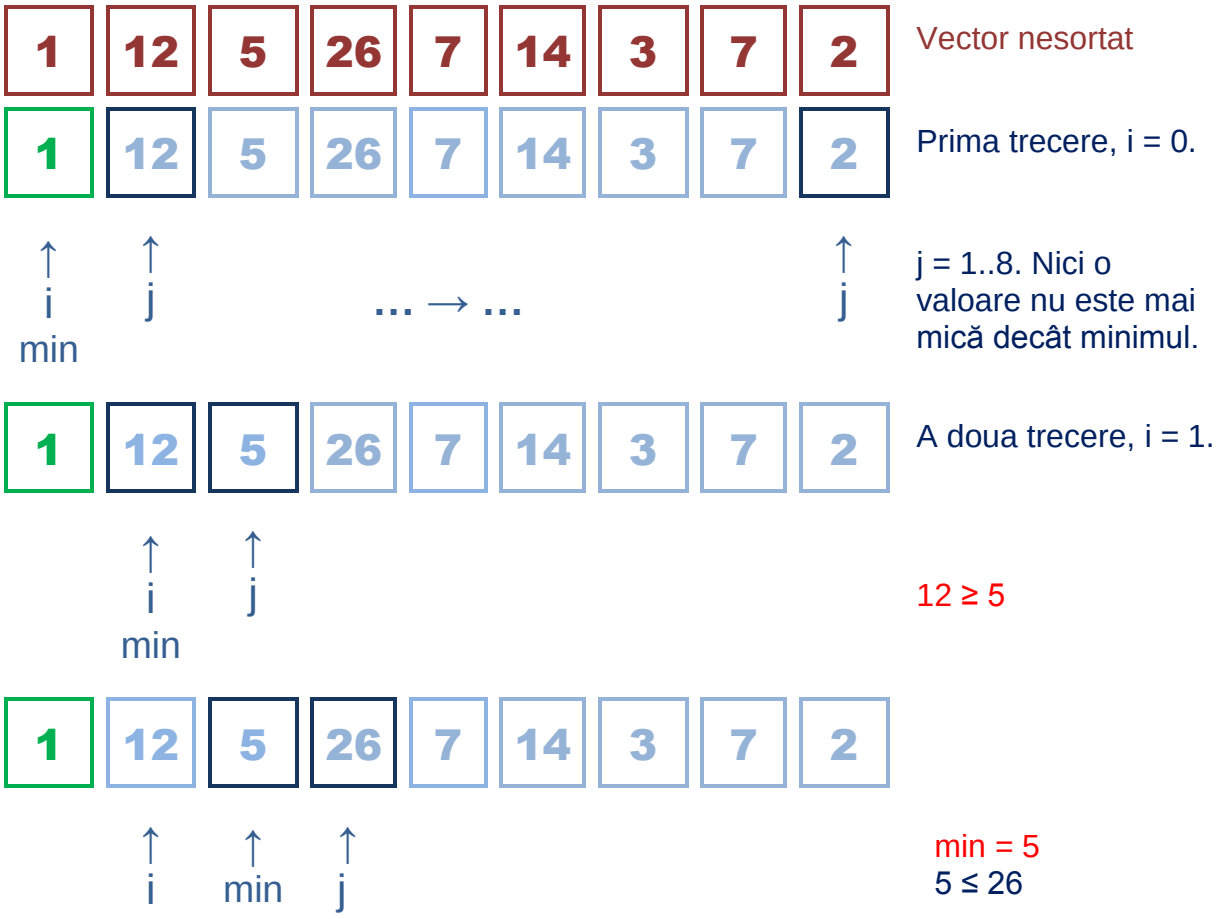
```
            for (i=0;i<n;i++)
            {
                System.out.print("v["+i+"]=");
                    v[i]=getInt();

            }
             sortare(v, n);
             System.out.println(" ");
             System.out.println("Vectorul sortat este:");
             for (i=0;i<n;i++)
                 System.out.print(v[i]+" ");
             System.out.println(" ");

        }
}
```

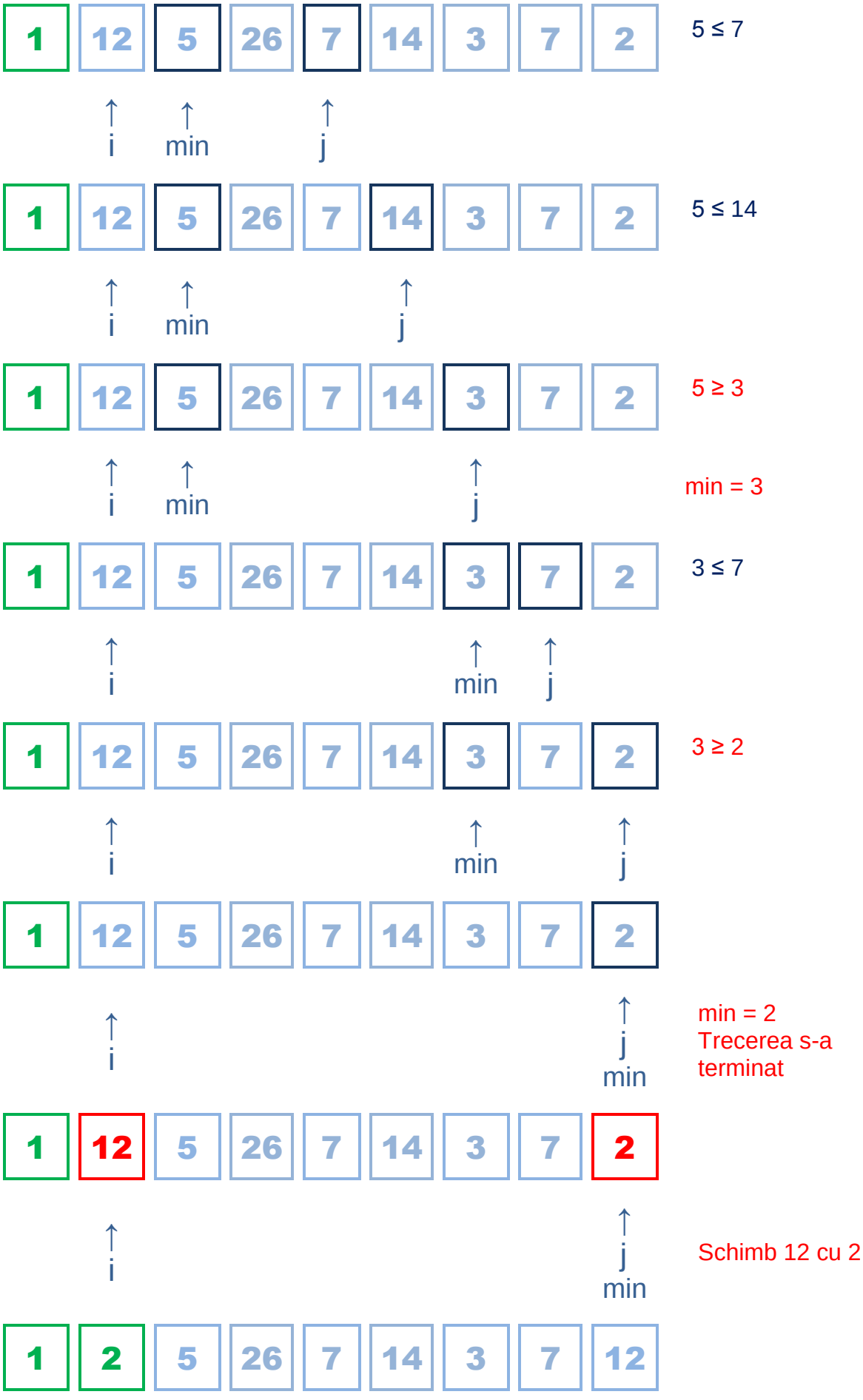
3. Sortarea prin selecţie directă a minimului sau a maximului

Algoritm:

Se găseşte minimul şi se pune pe prima poziţie; se găseşte minimul dintre elementele rămase şi se pune pe a doua poziţie etc.

Exemplu:




1 12 5 26 7 14 3 7 2
i min j
5 ≤ 7
1 12 5 26 7 14 3 7 2
i min j
5 ≤ 14
1 12 5 26 7 14 3 7 2
i min j
5 ≥ 3
min = 3
1 12 5 26 7 14 3 7 2
i min j
3 ≤ 7
1 12 5 26 7 14 3 7 2
i min j
3 ≥ 2
1 12 5 26 7 14 3 7 2
i min j
1 12 5 26 7 14 3 7 2
i j min
min = 2
Trecerea s-a terminat
1 12 5 26 7 14 3 7 2
i j min
Schimb 12 cu 2
1 2 5 26 7 14 3 7 12

Problemă:

Se dă un vector cu *n* elemente numere întregi. Să se sorteze crescător acest tablou folosind metoda sortării prin selecţie directă a minimului.

```
import java.io.*;
class Sort3
{
        //introducere un string de la tastatura
        public static String getString() throws IOException
        {
                InputStreamReader sir=new InputStreamReader(System.in);
                BufferedReader br=new BufferedReader(sir);
                String s=br.readLine();
                return s;
        }

       // introducere integer de la tastatura
        public static int getInt() throws IOException
        {
                String s=getString();
                return Integer.parseInt(s);
        }

        public static void sort_selectie_min(int[] v, int n)
        {
                int i, j, min,k;
                for(i=0; i<n; i++)          //ciclu exterior
                {
                  min=v[i];
                  k=i;
                  for(j=i+1;j<n;j++)     //ciclu interior
                    if(v[j]<min)
                    {
                        min=v[j];
                        k=j;
                    }

                  v[k]=v[i];
                  v[i]=min;
                }
        }
```

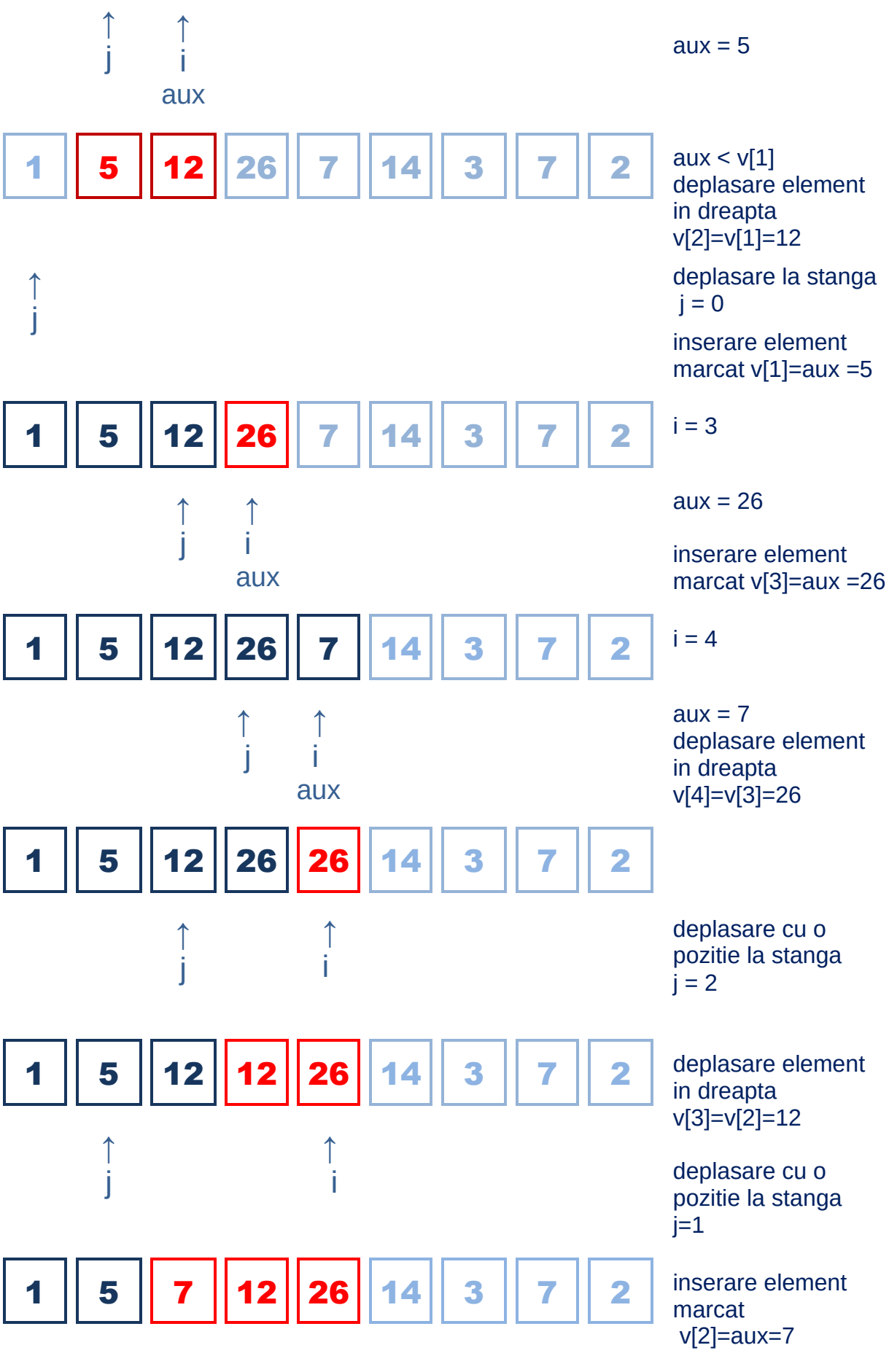
```
    public static void main(String[] args) throws IOException
    {
            int[] v;
            v=new int[100];
            int n, i;
            System.out.println("Dati nr de elem ale vectorului ");
            n=getInt();
            System.out.println("Dati elem vectorului");

            for (i=0;i<n;i++)
            {
                System.out.print("v["+i+"]=");
                v[i]=getInt();
            }

           sort_selectie_min(v, n);
            System.out.println(" ");
            System.out.println("Vectorul sortat este: ");
            for (i=0;i<n;i++)
                System.out.print(v[i]+" ");
            System.out.println(" ");
             }
}
```

4. Sortarea prin inserţie directă

Algoritm:

- Construieşte pas cu pas lista de elemente sortate, adăugând la aceasta câte un element la un moment dat.
- La fiecare pas un element este extras din lista iniţială şi este introdus în lista de elemente sortate.
- Elementul este inserat în poziţia corectă în lista sortată, astfel încât ea să rămână sortată în continuare.

Exemplu:


j
i
aux
aux = 5
1 5 12 26 7 14 3 7 2
aux < v[1]
deplasare element
in dreapta
v[2]=v[1]=12
j
deplasare la stanga
j = 0
inserare element
marcat v[1]=aux =5
1 5 12 26 7 14 3 7 2
i = 3
j
i
aux
aux = 26
inserare element
marcat v[3]=aux =26
1 5 12 26 7 14 3 7 2
i = 4
j
i
aux
aux = 7
deplasare element
in dreapta
v[4]=v[3]=26
1 5 12 26 26 14 3 7 2
j
i
deplasare cu o
pozitie la stanga
j = 2
1 5 12 12 26 14 3 7 2
deplasare element
in dreapta
v[3]=v[2]=12
j
i
deplasare cu o
pozitie la stanga
j=1
1 5 7 12 26 14 3 7 2
inserare element
marcat
v[2]=aux=7
...

Problemă:

Se dă un vector cu *n* elemente numere întregi. Să se sorteze crescător acest tablou utilizând metoda sortării prin inserţie directă.

```
import java.io.*;
class sort4
{
        //introducere un string de la tastatura
        public static String getString() throws IOException
        {
                InputStreamReader sir=new InputStreamReader(System.in);
                BufferedReader br=new BufferedReader(sir);
                String s=br.readLine();
                return s;
        }

       // introducere integer de la tastatura
        public static int getInt() throws IOException
        {
                String s=getString();
                return Integer.parseInt(s);
        }

        static  void sort_insertie(int[] v, int n)
       {
         int aux,j,i;
        for(i=1;i<n;i++)                    //i este linia de demarcare dintre
        {                                   //elementele sortate si cele nesortate
         aux=v[i];                          //sterge elementul marcat
         j=i-1;                             //permutarile incep de la acest indice
         while((j>=0)&&(aux<v[j]))  //pana se gaseste un element mai mic
        {
                v[j+1]=v[j];              //deplasare element in dreapta
                 j--;                    //deplasare cu o pozitie la stanga
        }
        v[j+1]=aux;        //inserare element marcat
        }
        }

        public static void main(String[] args) throws IOException
        {
                int[] v;
                v=new int[100];
                int n, i;
                System.out.println("Dati nr de elem ale vectorului ");
                n=getInt();
                System.out.println("Dati elem vectorului");
```

```
            for (i=0;i<n;i++)
            {
                System.out.print("v["+i+"]=");
                v[i]=getInt();
            }

            sort_insertie(v, n);
            System.out.println(" ");
            System.out.println("Vectorul sortat este: ");
            for (i=0;i<n;i++)
                System.out.print(v[i]+" ");
            System.out.println(" ");
        }
}
```

5. Sortarea prin interschimbare folosind partiţii - Quicksort

Algoritm:

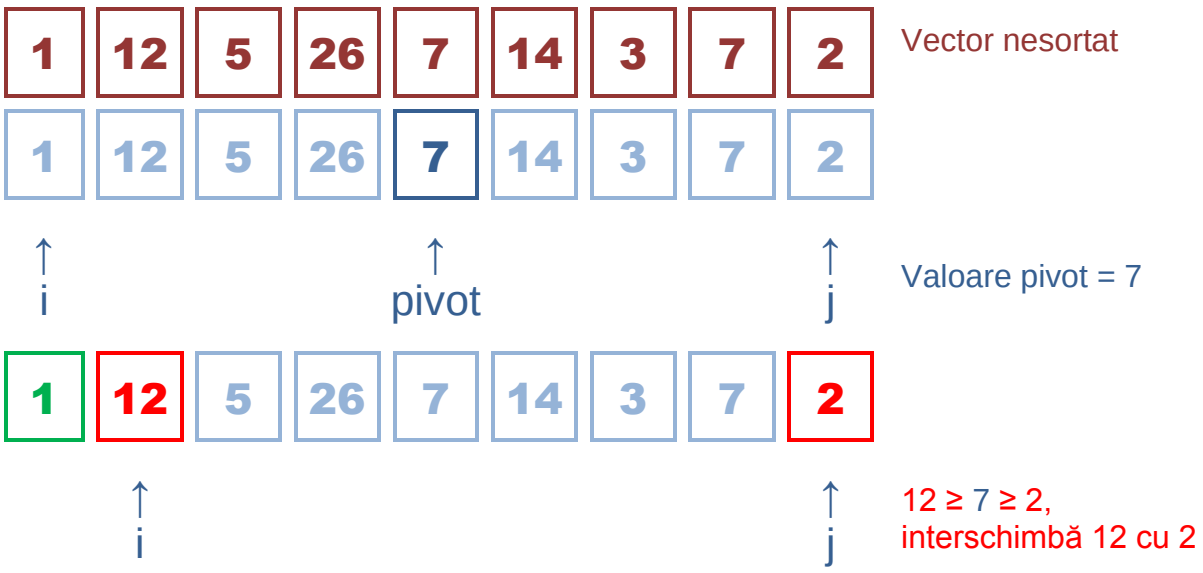
Sortarea se efectuează bazându-se pe o strategie *divide et impera*. Se împarte lista de sortat în două subliste mai uşor de sortat.

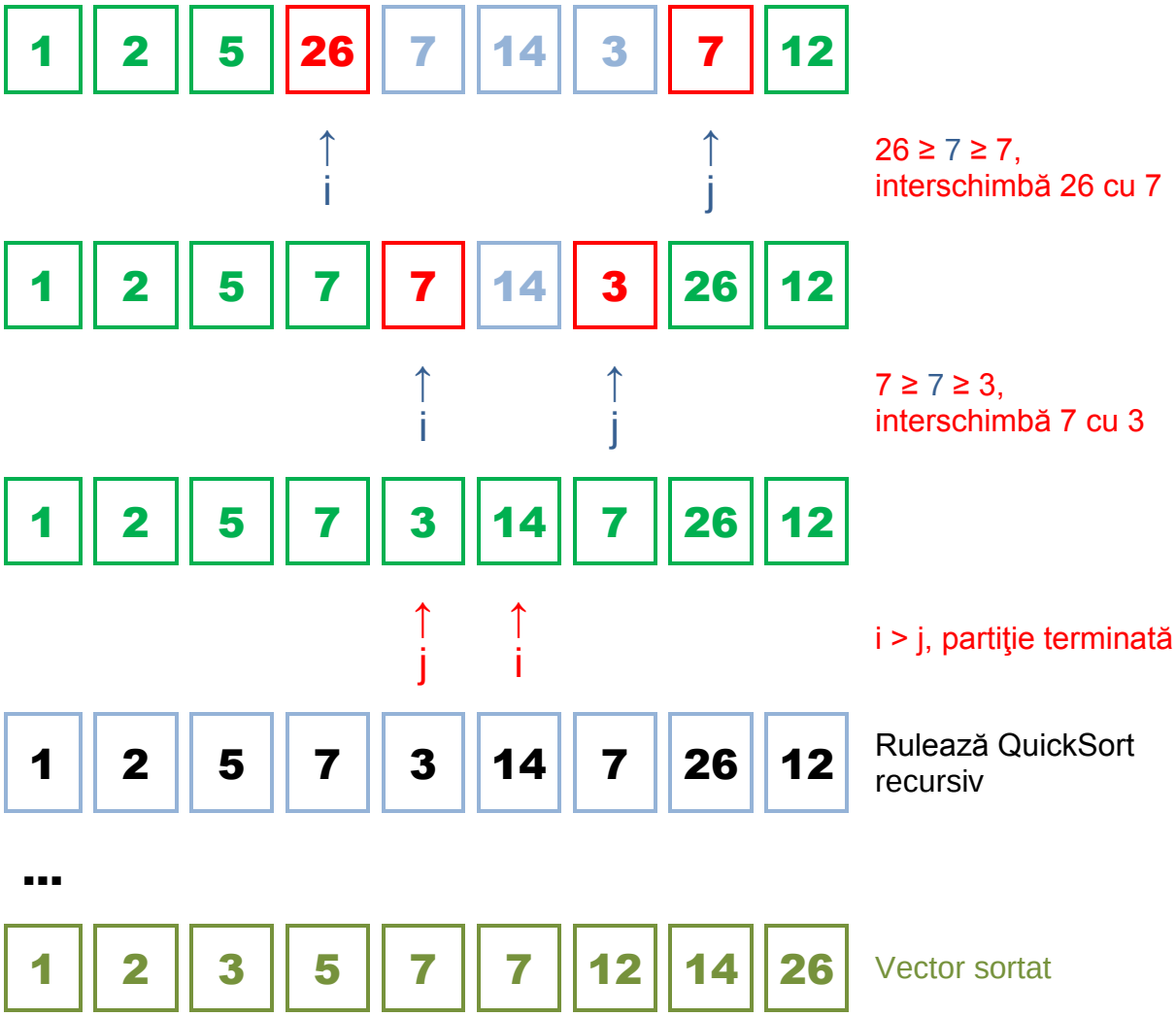
Paşii algoritmului sunt:

- se alege un element al listei, denumit **pivot**;
- se reordonează lista astfel încât toate elementele mai mici decât pivotul să fie plasate înaintea pivotului şi toate elementele mai mari să fie după pivot; după această partiţionare, pivotul se află în poziţia sa finală;
- se sortează recursiv sublista de elemente mai mici decât pivotul şi sublista de elemente mai mari decât pivotul;

O listă de dimensiune 0 sau 1 este considerată sortată.

Exemplu





Problemă:

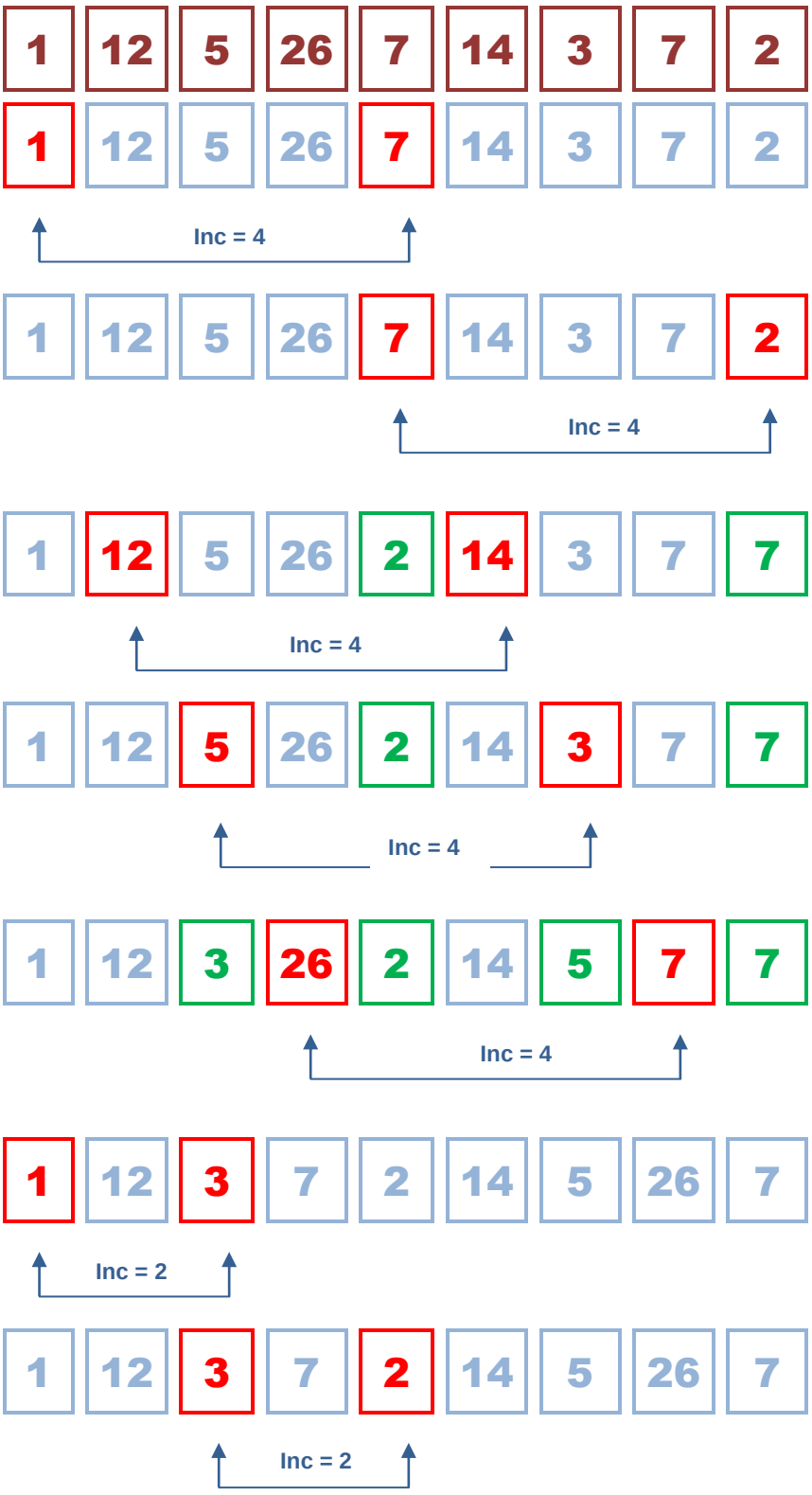
Se dă un vector cu *n* elemente numere întregi. Să se sorteze crescător acest tablou utilizând metoda sortării prin interschimbare folosind partiţii (Quicksort).

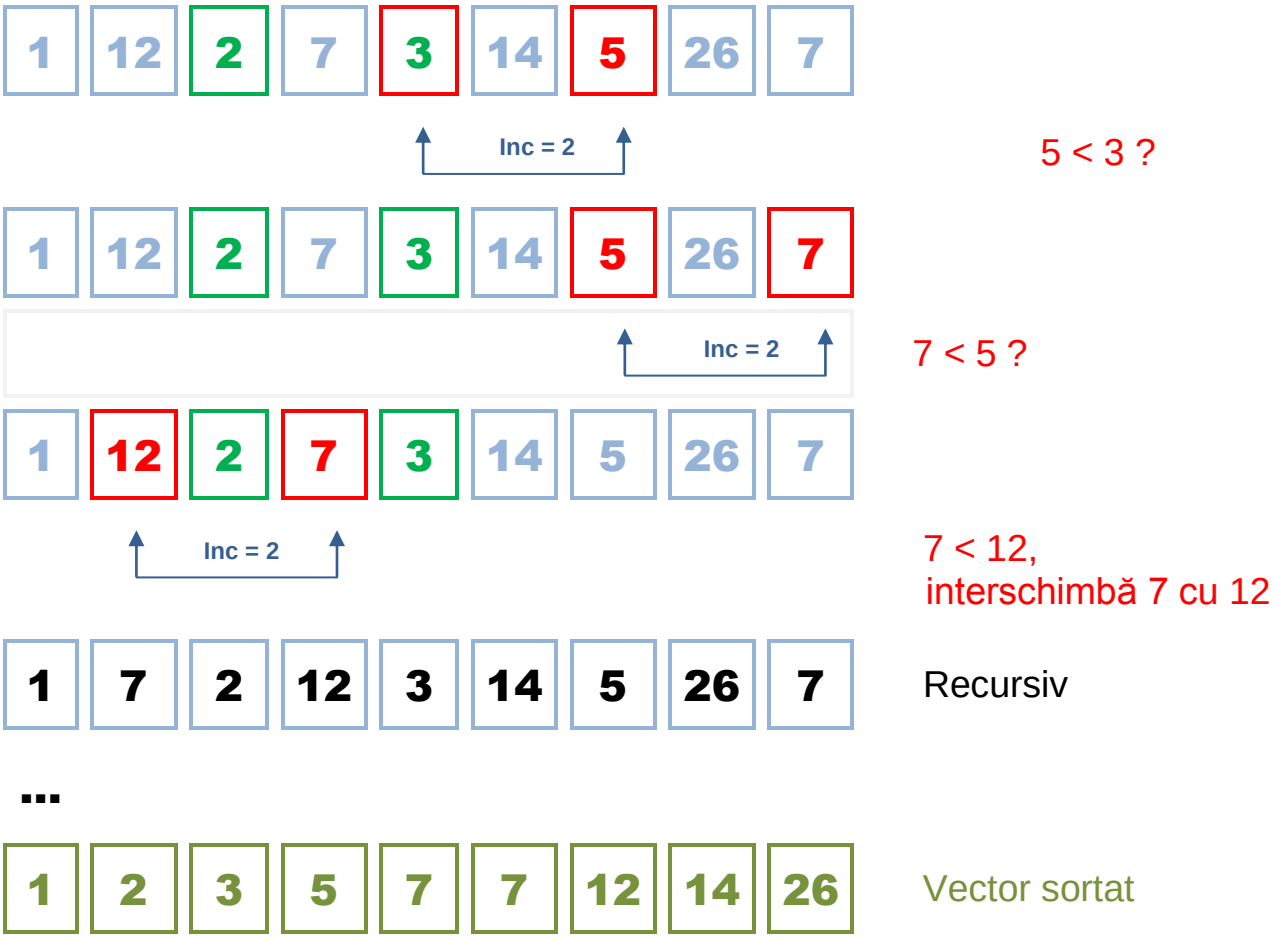
```
import java.io.*;
class sort5
{
        //introducere un string de la tastatura
        public static String getString() throws IOException
        {
                InputStreamReader sir=new InputStreamReader(System.in);
                BufferedReader br=new BufferedReader(sir);
                String s=br.readLine();
                return s;
        }

       // introducere integer de la tastatura
        public static int getInt() throws IOException
        {
                String s=getString();
                return Integer.parseInt(s);
        }
```

```
static int partitionare(int v[], int stanga, int dreapta)
        {
            int i = stanga, j = dreapta;
            int tmp;
            int pivot = v[(stanga + dreapta) / 2];

            while (i <= j) {
                while (v[i] < pivot)
                    i++;
                while (v[j] > pivot)
                    j--;
                if (i <= j) {
                    tmp = v[i];
                    v[i] = v[j];
                    v[j] = tmp;
                    i++;
                    j--;
                }
            };
            return i;
        }

static void quicksort(int v[], int stanga, int dreapta) {
            int index = partitionare(v, stanga, dreapta);
            if (stanga < index - 1)
                quicksort(v, stanga, index - 1);
            if (index < dreapta)
                quicksort(v, index, dreapta);
        }

 public static void main(String[] args) throws IOException
{
          int[] v;
          v=new int[100];
          int n, i;
          System.out.println("Dati nr de elem ale vectorului ");
          n=getInt();
          System.out.println("Dati elem vectorului");

          for (i=0;i<n;i++)
          {
               System.out.print("v["+i+"]=");
                    v[i]=getInt();
          }

           quicksort(v, 0, n-1);
           System.out.println(" ");
           System.out.println("Vectorul sortat este: ");
           for (i=0;i<n;i++)
                System.out.print(v[i]+" ");
           System.out.println(" ");
}}
```

6. Sortarea prin inserţie cu micşorarea incrementului - Shellsort

Exemplu:





Problemă:

Se dă un vector cu *n* elemente numere întregi. Să se sorteze crescător acest tablou utilizând metoda sortării prin inserţie cu micşorarea incrementului.

```
import java.io.*;
class sort6
{
        //introducere un string de la tastatura
        public static String getString() throws IOException
        {
                InputStreamReader sir=new InputStreamReader(System.in);
                BufferedReader br=new BufferedReader(sir);
                String s=br.readLine();
                return s;
        }

       // introducere integer de la tastatura
        public static int getInt() throws IOException
        {
                String s=getString();
                return Integer.parseInt(s);
        }

        // Sorteaza prin insertie cu micsorarea incrementului
```

```
    public static void shellsort(int v[], int n)
    {
     int increment = n / 3 + 1;

     while ( increment > 1 )
     {
      for ( int start = 0; start < increment; start++ )
      insertSort(v, start,n, increment);

      increment = increment / 3 + 1;
     }

     insertSort(v, 0, n, 1);
   }

 public static void insertSort(int v[], int start,  int n, int increment)
{
 int j, k, temp;

 for ( int i = start + increment; i < n; i += increment )
 {
   j = i;
   k = j - increment;
   if ( v[j] < v[k] )
   {
     // Interschimba toate elementele folosind incrementul curent
     // pana cand se gaseste indexul potrivit
     temp = v[j];
     do
     {
       v[j] = v[k];
       j = k;
       k = j - increment;
     } while ( j != start && v[k] > temp );
     v[j] = temp;
   }
  }
 }

    public static void main(String[] args) throws IOException
     {
              int[] v;
              v=new int[100];
              int n, i;
              System.out.println("Dati nr de elem ale vectorului ");
              n=getInt();
              System.out.println("Dati elem vectorului");

              for (i=0;i<n;i++)
              {
                    System.out.print("v["+i+"]=");
                          v[i]=getInt();
              }
```

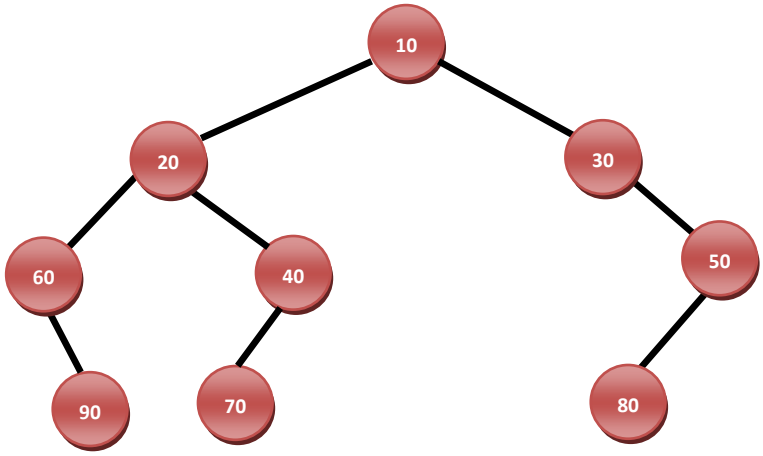
```
            shellsort(v,n);
            System.out.println(" ");
            System.out.println("Vectorul sortat este: ");
            for (i=0;i<n;i++)
                System.out.print(v[i]+" ");
            System.out.println(" ");
        }
}
```

7. Metode de parcurgere a arborilor binari

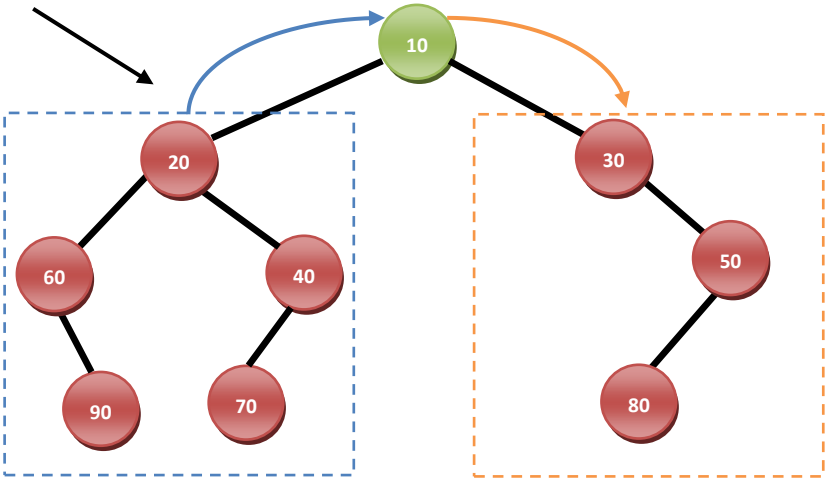
Definiţii:

- Un arbore constă din noduri care sunt unite prin muchii.
- Singurul mod de a ajunge de la un nod la altul este de a urma un drum de-a lungul muchiilor.
- **Cale (drum) –** ne deplasăm de la un nod la altul de-a lungul muchiilor prin care acestea sunt conectate; secvenţa de noduri rezultată se numeşte cale.
- **Rădăcină –** nodul din vârful arborelui.
- **Părinte –** orice nod (cu excepţia rădăcinii) are exact o muchie orientată în sus spre un alt nod. Nodul situat mai sus este numit părintele nodului curent.
- **Fiu –** orice nod poate avea una sau mai multe muchii orientate în jos, spre alte noduri. Aceste noduri situate mai jos sunt numite fii nodului curent.
- **Frunză –** un nod fără fii.
- **Subarbore –** un nod poate fi considerat ca rădăcină pentru un subarbore, care cuprinde fii acelui nod, fii fiilor săi etc (până se ajunge la frunze).
- **Vizitare –** un nod este vizitat atunci când se execută operaţii asupra sa; o simplă trecere printr-un nod cu scopul de ajunge la alt nod nu este considerată vizitare.
- **Parcurgere –** vizitarea tuturor nodurilor din arbore.
  - parcurgere în inordine (subarbore stâng – rădăcină – subarbore drept)
  - parcurgere în preordine (rădăcină – subarbore stâng – subarbore drept)
  - parcurgere în postordine (subarbore stâng – subarbore drept – rădăcină)
- **Nivel –** nivelul unui nod este numărul de muchii parcurse pentru a ajunge de la rădăcină la acel nod. Rădăcina arborelui este pe nivelul 0, fii acesteia pe nivelul 1 etc.
- **Cheie –** valoarea nodului.
- **Arbore binar** - un arbore în care orice nod poate avea cel mult doi fii.

Exemplu:



1. Parcurgere în inordine (subarbore stâng – rădăcină – subarbore drept)



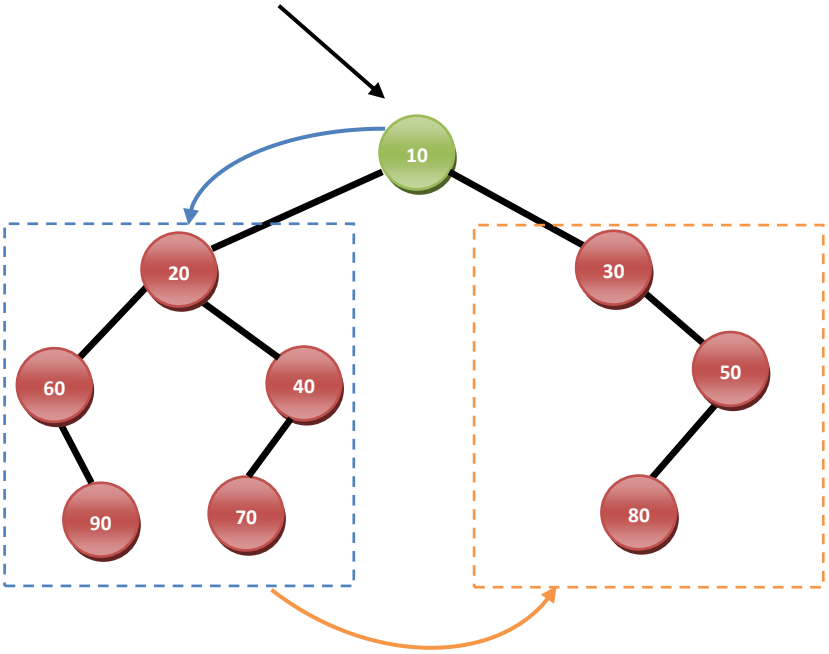
(20) 10 (30)

(60) 20 (40) 10 ( ) 30 (50)

( ) 60 (90) 20 (70) 40 ( ) 10 30 (80) 50 ( )

60 90 20 70 40 10 30 80 50

2. Parcurgere preordine (rădăcină – subarbore stâng – subarbore drept)



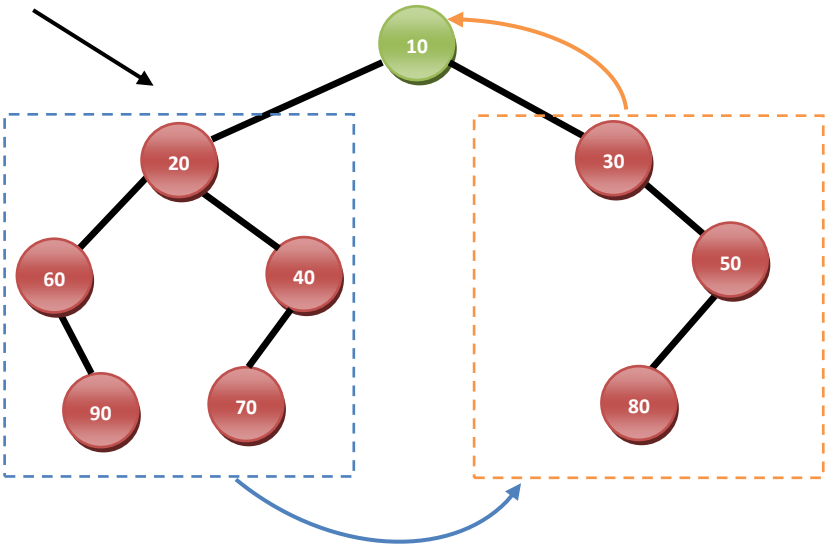
10 (20) (30)

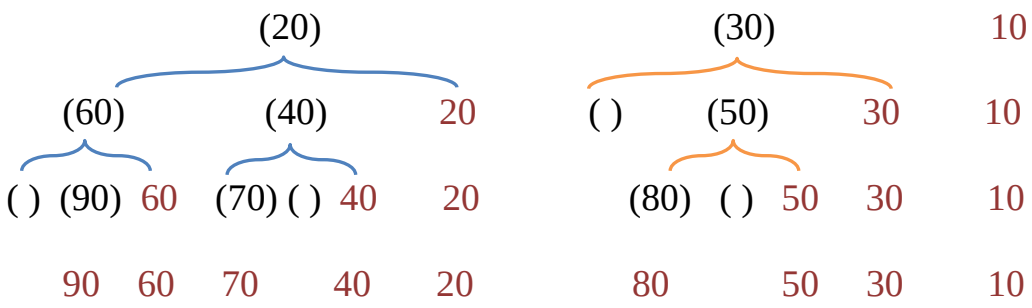
10 20 (60) (40) 30 ( ) (50)

10 20 60 ( ) (90) 40 (70) () 30 50 (80) ( )

10 20 60 90 40 70 30 50 80

3. Parcurgere în postordine (subarbore stâng – subarbore drept – rădăcină)





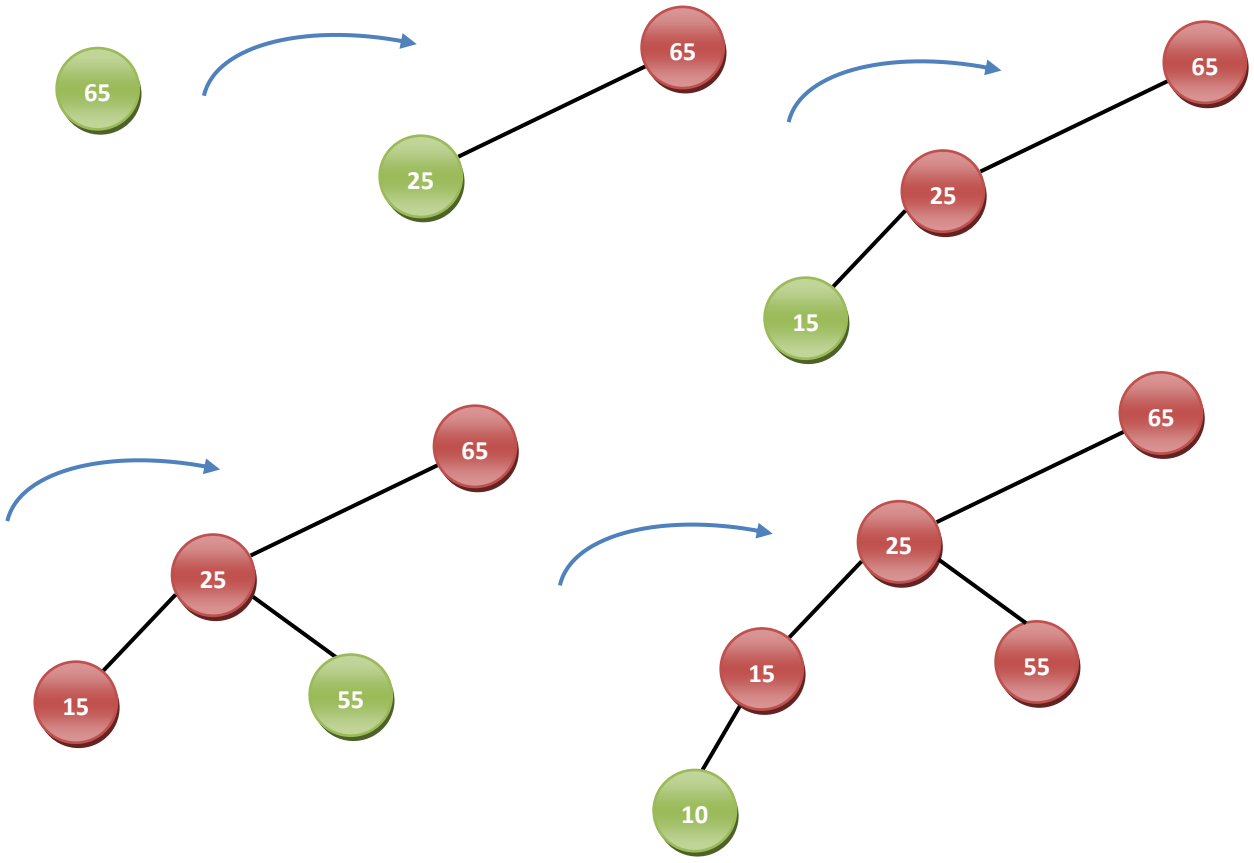
8. Arbori binari de căutare

Definiţie:

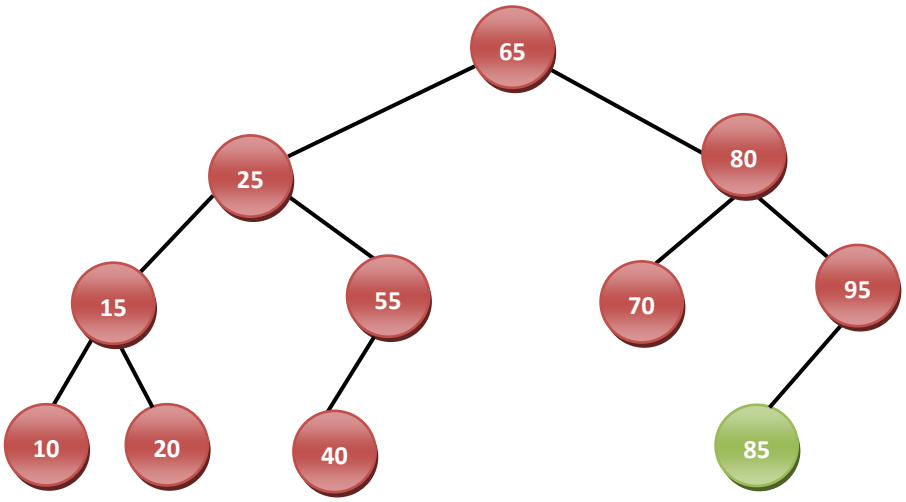
- Toate nodurile care sunt descendenţii stângi ai unui nod X au chei mai mici decât X.
- Toate nodurile care sunt descendenţii drepţi ai unui nod X au chei mai mari decât X.
- În acest caz, parcurgerea în inordine realizează parcurgerea nodurilor în ordine crescătoare a cheilor.

Exemplu:

Se dă următorul şir: 65, 25, 15, 55, 10, 20, 40, 80, 70, 95, 85. Să se formeze arborele binar de căutare asociat acestor numere.



...



Problemă:

Să se implementeze următoarele operaţii pentru un arbore binar de căutare:

- inserare nod;
- ştergere nod;
- parcurgere în inordine;
- parcurgere în postordine;
- parcurgere în preordine.

```
import java.io.*;
class Nod
{
	int valoare;  //informatia din nod
	Nod st;       //fiul din stanga
	Nod dr;       //fiul din dreapta

	public void afiseaza()
	{
		System.out.println(valoare+" ");
	}
}

class Arbore
{
	private Nod radacina; //primul nod al arborelui

	//constructor
	public Arbore()
	{
		radacina=null;  //arbore vid
	}
```

```
//metoda de cautare
public Nod cauta(int n)
{
        Nod curent=radacina;
        if(curent==null)
          return null;
        while(curent.valoare!=n)
        {
                if (n<curent.valoare)     //deplasare la stanga?
                        curent=curent.st;
                 else
                        curent=curent.dr;
                 if(curent==null)          //daca nu exista descendenti
                        return null;       //cheia nu este in arbore
                }
         return curent;
}
//metoda inserare nod nou
public void insereaza(int n)
{
        //creeaza un nou nod
        Nod nou=new Nod();
        nou.valoare=n;

        if(radacina==null)          //arborele era vid
                radacina=nou;
        else
        {
          Nod curent=radacina;      //se incepe cu radacina
          Nod parinte;
          while (true)              //se termina intern
          {
                parinte=curent;
                 if(n<curent.valoare) //deplasare la stanga?
                 {
                        curent=curent.st;
                         //daca este capatul liniei se insereaza nodul al stanga
                        if(curent==null)
                        {
                                parinte.st=nou;
                                return;
                        }
                 }
                 else    //deplasare la dreapta?
                 {
                        curent=curent.dr;
                        if(curent==null)
                         //daca este capatul liniei se insereaza nodul al dreapta
                        {
                                parinte.dr=nou;
                                return;
                        }
                 }
```

```
            }
        }
    }

//metoda stergere nod
public boolean sterge(int n)
{
        Nod curent=radacina;
        Nod parinte=radacina;
        boolean t=true;
        while (curent.valoare!=n)
        {
                parinte=curent;

                if(n<curent.valoare)   //deplasare la stanga?
                {
                        t=true;          //e fiul din stanga
                        curent=curent.st;
                }
                else
                {
                        t=false;  //nu e fiul din stanga ci e fiul din dreapta
                        curent=curent.dr;
                  }
                 if(curent==null)   //nu s-a gasit cheia
                        return false;
                }
                //am gasit nodul care trebuie sters
                //daca nodul nu are copii se sterge direct
                if((curent.st==null)&&(curent.dr==null))
                {
                  if (curent==radacina)  //daca e radacina atunci arborele devine vid
                                radacina=null;
                  else //se elimina legatura cu parintele nodului
                     if(t)  //daca e fiul din stanga
                        parinte.st=null;
                     else  //daca e nodul din dreapta
                        parinte.dr=null;
                }
                //nodul de sters are un fiu
                //daca nu exista fiu in dreapta, se inlocuieste cu subarborele stang
                else
                  if(curent.dr==null)
                     if(curent==radacina)
                        radacina=curent.st;
                     else
                        if(t)
                           parinte.st=curent.st;
                        else
                           parinte.dr=curent.st;
                  //daca nu exista fiu in stanga, inlocuieste cu subarborele drept
                  else
                    if(curent.st==null)
```

```
                        if(curent==radacina)
                                radacina=curent.dr;
                        else
                            if(t)
                                parinte.st=curent.dr;
                             else
                                parinte.dr=curent.dr;
                    //dc nodul are 2 fii, atunci se inlocuieste cu succesorul inordine
                    else
                    { //determina succesorul nodului de sters (curent)
                            Nod succesor=cautaSuccesor(curent);
                       //succesor devine fiul parintelui lui curent
                       if(curent==radacina)
                            radacina=succesor;
                      else
                          if(t)
                                parinte.st=succesor;
                          else
                                parinte.dr=succesor;
                       //subarborele stang al nodului curent  este atasat la succesor
                       succesor.st=curent.st;
                    }
        //succesor nu poate avea subarbore stang
        return true;
}

//intoarce nodul cu valoarea imediat mai mare decat valoarea de sters
//parcurge fiul drept apoi descendentii din stanga
private Nod cautaSuccesor(Nod nd)
{
        Nod parinteSuccesor=nd;
        Nod succesor=nd;
        Nod curent=nd.dr;  //avans la fiul drept
        while (curent!=null) //pana cand nu mai sunt fii stangi
        {
                parinteSuccesor=succesor;
                succesor=curent;
                curent=curent.st; //avans la fiul stang
        }
        //daca succesorul nu e fiul drept modifica referintele
        if(succesor!=nd.dr)
         {
                parinteSuccesor.st=succesor.dr;
                succesor.dr=nd.dr;
         }
        return succesor;
   }

//metoda de parcurgere
public void parcurge(int opt)
        {switch(opt)
                {case 1: System.out.println("parcurgere in preordine-radacina,st,dr");
                        preordine(radacina);
```

```
                        break;
                    case 2: System.out.println("parcurgere in inordine-st,radacina,dr");
                        inordine(radacina);
                        break;
                    case 3: System.out.println("parcurgere in postordine-st,dr,radacina");
                        postordine(radacina);
                        break;
                    }
                System.out.println();
                }
        //parcurgere in preordine
        public void preordine(Nod m)
                {if (m!=null)
                 {m.afiseaza();
                  preordine(m.st);
                  preordine(m.dr);
                 }
                }
         //parcurgere in inordine
        public void inordine(Nod m)
           {if (m!=null)
             {inordine(m.st);
                        m.afiseaza();
                        inordine(m.dr);
                 }
           }
         //parcurgere in postordine
        public void postordine(Nod m)
        {
           if(m!=null)
             {postordine(m.st);
                        postordine(m.dr);
                        m.afiseaza();
             }
        }

   }

class arbori
{
        // introducere string
        public static String getString() throws IOException{
                InputStreamReader isr=new InputStreamReader(System.in);
                BufferedReader br=new BufferedReader(isr);
                String s=br.readLine();
                return s;
        }
        // introducere integer
        public static int getInt() throws IOException{
                String s = getString();
                return Integer.parseInt(s);
        }
```

```
public static void main(String[] args) throws IOException
{
        int valoare;
        Arbore arb=new Arbore();
        arb.insereaza(65);
        arb.insereaza(25);
        arb.insereaza(15);
        arb.insereaza(55);
        arb.insereaza(10);
        arb.insereaza(20);
        arb.insereaza(40);
        arb.insereaza(80);
        arb.insereaza(70);
        arb.insereaza(95);
        arb.insereaza(85);

        System.out.println("1.Cautare");
        System.out.println(" ");
        System.out.println("2.Inserare");
        System.out.println(" ");
        System.out.println("3.Stergere ");
        System.out.println(" ");
        System.out.println("4.Parcurgere ");
        System.out.println(" ");
        System.out.println("5.Exit ");
        System.out.println(" ");

        boolean var=true;
        int opt;
        while(var){
        System.out.print("Alege optiune ");
        opt=getInt();
        if(opt!=5)
                var=true;
        else
                var=false;
        int nr_i,nr_dupa;
        switch(opt)
        {
           case 1:System.out.print("Dati valoarea cautata: ");
                valoare=getInt();
                Nod gasit=arb.cauta(valoare);
                if(gasit!=null)
                        System.out.print("Valoare "+valoare+" gasita.");
                else
                        System.out.print("Valoarea nu a fost gasita.");
                break;
            case 2:System.out.print("Dati valoarea pe care doriti sa o inserati: ");
                valoare=getInt();
                arb.insereaza(valoare);
                 break;
            case 3:System.out.print("Dati valoarea pe care doriti sa o stergeti: ");
                valoare=getInt();
```

```
                    boolean sters=arb.sterge(valoare);
                    if(sters)
                            System.out.println("Valoare "+valoare+" stearsa.");
                    else
                            System.out.println("Valaorea nu a fost gasita.");
                    break;
                case 4:System.out.println("Dati valoarea 1(preordine),2(inordine),3(postordine)");
                    valoare=getInt();
                    arb.parcurge(valoare);
                    break;
                case 5:System.exit(0);
                    System.out.println(" ");
                    break;
            }
}
}
}
```

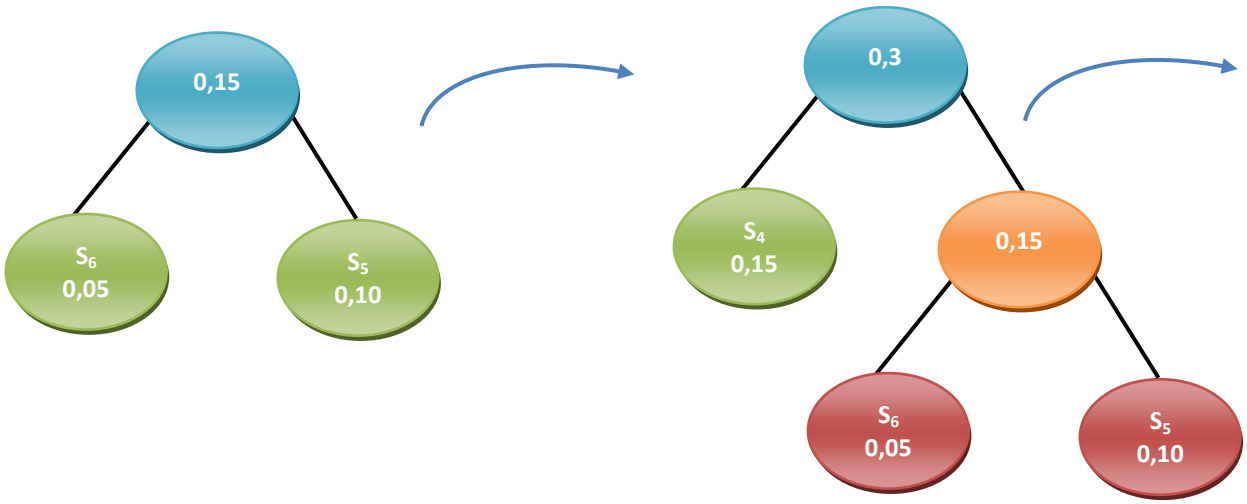
8. Algoritmul lui Huffman

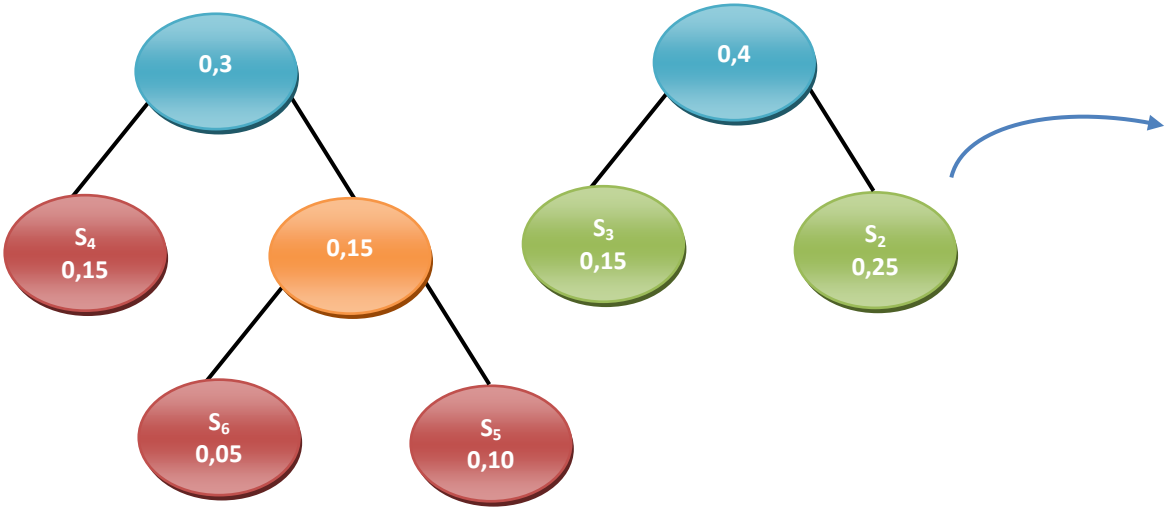
Exemplu:

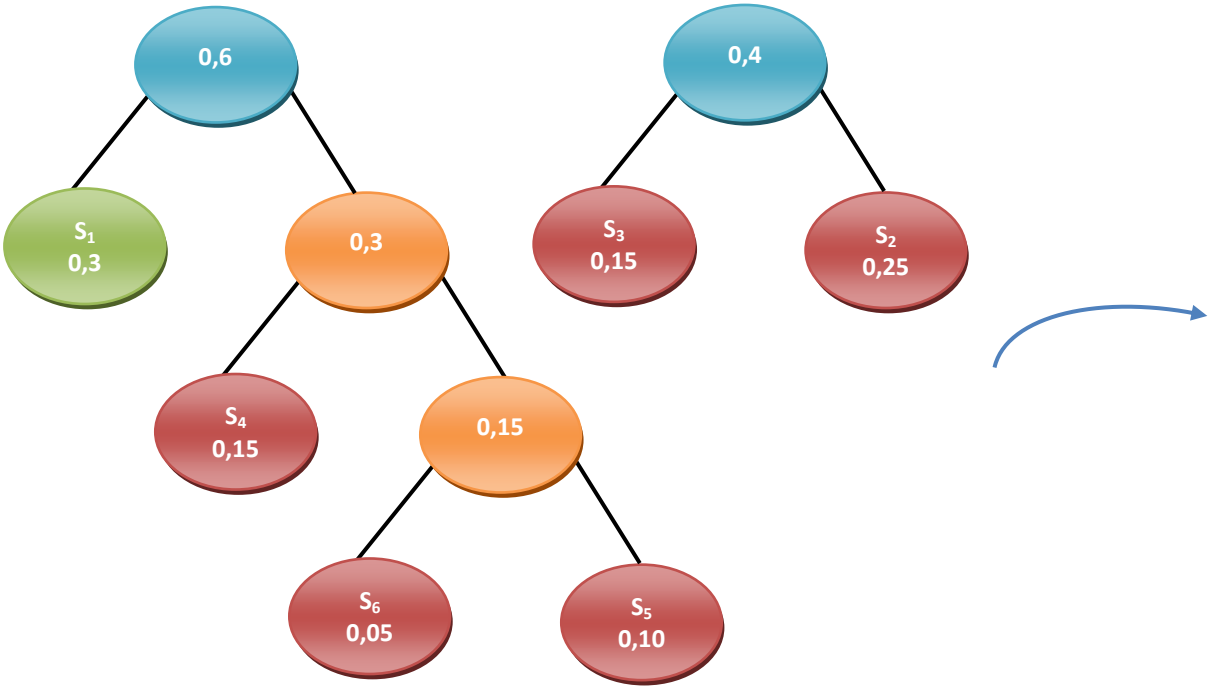
Se dă un şir de frecvenţe:

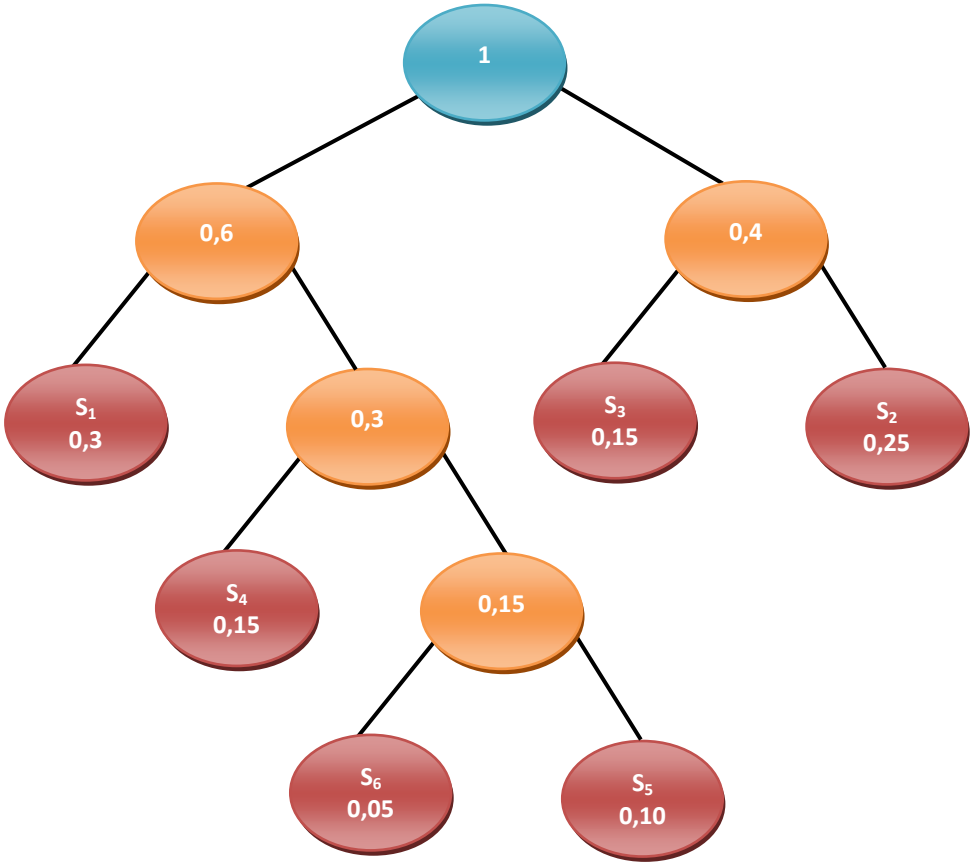
$$s = \begin{pmatrix} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ 0{,}3 & 0{,}25 & 0{,}15 & 0{,}15 & 0{,}10 & 0{,}05 \end{pmatrix}$$

- Se găsesc cele mai mici valori x_1 şi x_2;
- Se rezolvă problema pentru şirul $x_1 + x_2, x_3, \ldots, x_n$ şi se înlocuieşte nodul $x_1 + x_2$ cu frunzele x_1 şi x_2.




0,3
S4
0,15
0,15
S6
0,05
S5
0,10
0,4
S3
0,15
S2
0,25


0,6
S1
0,3
0,3
S4
0,15
0,15
S6
0,05
S5
0,10
0,4
S3
0,15
S2
0,25



Codarea se obţine adăugând 1 pentru dreapta şi 0 pentru stânga.