COLEGIUL NAȚIONAL “CAROL I”

PROIECT DE ATESTAT
INFORMATICĂ

PROFESOR COORDONATOR,
LICĂ OVIDIU

ELEV,
DARIUS NEGUȚ

MAI 2023

METODE DE SORTARE

CUPRINS

Cuprins

ARGUMENT

Am ales această temă deoarece un capitol importat în programare îl reprezintă sortarea datelor.

Sortarea datelor are multe aplicații în practică: în matematică – statistică matematică, în realizarea unor dicționare

În acest material, pe lângă prezentarea considerațiilor teoretice, vom realiza o comparație a metodelor de sortare în ceea ce privește complexitatea ca timp și spațiu de memorie utilizat, analiză care este în același timp utilă pentru rezolvarea eficientă a problemelor de informatică.

Sortarea datelor

Sortarea este o operaţie fundamentala în informatică, mulţi algoritmi o folosesc ca pas intermediar, are o largă aplicabilitate în informatică şi disciplinele conexe.

Sortarea este procesul prin care elementele unui set de date sunt rearanjate astfel încât valorile lor să se afle într-o anumită ordine (crescătoare sau descrescătoare).

În practică, numerele care trebuie ordonate sunt rareori valori izolate. De obicei, fiecare număr face parte dintr-o colecţie de date numita articol. Fiecare articol conţine o cheie, care este valoarea ce trebuie ordonată, iar restul articolului conţine date adiţionale, care sunt, de obicei, mutate împreuna cu cheia. În practică, atunci când un algoritm de ordonare interschimbă cheile, el trebuie să interschimbe şi datele adiţionale.

Exemplu:

Considerăm setul de valori întregi: (7,18,6,1,4). În acest caz cheia de sortare coincide cu valoarea elementului. Prin sortare crescătoare se obține setul (1,4,6,7,18) iar prin sortare descrescătoare se obține (18,7,6,4,1).

Considerăm un tabel constând din nume ale studenților și note: ((Popescu,9), (Ionescu,10), (Voinescu,8), (Adam,9)). În acest caz cheia de sortare poate fi numele sau nota. Prin ordonare crescătoare după nume se obține ((Adam,9), (Ionescu,10), (Popescu,9), (Voinescu,8)), iar prin ordonare descrescătoare după notă se obține ((Ionescu,10), (Popescu,9), (Adam,9), (Voinescu,8)).

Faptul că sortam numere individuale sau articole mari, ce conţin numere, este irelevant pentru metoda prin care o procedura de ordonare determină ordinea elementelor. Astfel, atunci când ne concentrăm asupra problemei sortării, presupunem, de obicei, că intrarea constă numai din numere.

De aceea algoritmii pe care îi vom prezenta în această lucrare vot utiliza tablouri unidimensionale de numere întregi.

Metodele de sortare pot fi caracterizate prin:

- **Eficiență**: O metodă este considerată eficientă dacă nu necesită un volum mare de resurse.

Eficiența poate fi analizată:

  - Din punctul de vedere al spațiului de memorie: o metodă de sortare este eficientă din punct de vedere al memoriei dacă nu utilizează o zonă de manevră de dimensiunea tabloului.
  - Din punct de vedere al timpului de execuție este important să fie efectuate cât mai puține operații. În general, în analiză se iau în considerare doar operațiile efectuate asupra elementelor tabloului (comparații și interschimbări).
- **Simplitate:** O metodă este considerată simplă dacă este intuitivă și ușor de înțeles.

Metodele de sortare analizate le vom clasifica în:

- Metode de sortare directă
  - Sortarea prin comparare sau interschimbare
  - Sortarea bublle sort
  - Sortarea prin inserție directă
  - Sortarea prin selecție
- Metode avansate
  - Merge sort
  - Quick sort

Sortarea prin comparare

Ideea de bază a algoritmului constă în faptul că:

- Fiecare element al tabloului se compară cu toate elementele următoare
- Dacă un element următor este mai mic decât elementul curent (în cazul sortării în ordine crescătoare) sau dacă elementul următor este mai mare decât elementul curent (în cadrul sortării în ordine descrescătoare) cele două elemente se vor schimba.

Secvența de cod ce realizează ordonarea este următoarea:

Sortarea în ordine crescătoare	Sortarea în ordine descrescătoare

`for(int i=1;i<n;i++)`
`  for(int j=i+1;j<=n;j++)`
`  if(v[i]>v[j])`
`  { int t=v[i];`
`  v[i]=v[j];`
`  v[j]=t;`
`}`
	`for(int i=1;i<n;i++)`
	`  for(int j=i+1;j<=n;j++)`
	`  if(v[i]<v[j])`
	`  { int t=v[i];`
	`  v[i]=v[j];`
	`  v[j]=t;`
	`}`

a1	a2	a3	a4	a5
5	2	6	1	7
2	5	6	1	7
2	5	6	1	7
1	5	6	2	7
1	5	6	2	7
1	5	6	2	7
1	2	6	5	7
1	2	6	5	7
1	2	5	6	7
1	2	5	6	7
1	2	5	6	7

Observăm că după prima iterație a primului for în prima poziție va fi adus cel mai mic element din vector.

Complexitate memorie: O(n) – se folosește un singur tablou unidimensional

Complexitate timp de execuție: $O(n^2)$ - Numărul de iterații este n*(n-1)/2.

Sortarea Bublle Sort

Ideea de bază a algoritmului constă în:

- parcurgerea tabloului A de mai multe ori, până când devine ordonat. La fiecare pas se compară două elemente alăturate. Dacă $a_i > a_{i+1}$, (i = 1, 2, ..., n – 1) (la ordonarea crescătoare) sau $a_i < a_{i+1}$ (la ordonarea descrescătoare), atunci cele două valori se interschimbă între ele. Controlul acţiunii repetitive este dat de variabila booleană ok, care la fiecare reluare a algoritmului primeşte valoarea iniţială adevărat, care se schimbă în fals dacă s-a efectuat o interschimbare de două elemente alăturate.

- în momentul în care tabloul A s-a parcurs fără să se mai efectueze nici o schimbare, ok rămâne cu valoarea iniţială adevărat şi algoritmul se termină, deoarece tabloul este ordonat.

Sortarea în ordine crescătoare	Sortarea în ordine descrescătoare

`do{`
`  ok=1;`
`  for(int i=1;i<n;i++)`
`  if(a[i]>a[i+1])`
`  { int t=a[i];`
`  a[i]=a[i+1];`
`  a[i+1]=t;`
`  ok=0;`
`  }`
`}while(ok==0);`
	`do{`
	`  ok=1;`
	`  for(int i=1;i<n;i++)`
	`  if(a[i]<a[i+1])`
	`  { int t=a[i];`
	`  a[i]=a[i+1];`
	`  a[i+1]=t;`
	`  ok=0;`
	`  }`
	`}while(ok==0);`

a1	a2	a3	a4	a5
5	2	6	1	7
2	5	6	1	7
2	5	6	1	7
2	5	1	6	7
2	5	1	6	7
2	5	1	6	7

2	1	5	6	7
2	1	5	6	7
2	1	5	6	7
1	2	5	6	7
1	2	5	6	7
1	2	5	6	7
1	2	5	6	7

Observăm că după prima iterație a for-ului în ultima poziție va fi adus cel mai mare element din vector.

Complexitate memorie: O(n) – se folosește un singur tablou unidimensional

Complexitate timp de execuție: Spre deosebire de algoritmul anterior numărul de iterații nu mai este fix n*(n-1)/2, algoritmul încheindu-se când vectorul devine ordonat. Pe cazul cel mai favorabil, când vectorul este ordonat crescător complexitatea este O(n), algoritmul încheindu-se după prima parcurgere a vectorului. Pe cazul cel mai defavorabil, vectorul este ordonat descrescător complexitatea este $O(n^2)$.

Sortarea prin Selecție

Ideea de bază a algoritmului constă în:

-În cazul ordonării crescătoare la fiecare pas i, i=1,2, ..., n-1 se determină minimul secvenței a_i, a_{i+1}, ..., a_n și poziția în care apare, fie p poziția acestuia. Elementele a[i] și a[p] se interschimbă dacă a[i]>a[p].

- În cazul ordonării descrescătoare la fiecare pas i, i=1,2, ..., n-1 se determină maximul secvenței a_i, a_{i+1}, ..., a_n și poziția în care apare, fie p poziția acestuia. Elementele a[i] și a[p] se interschimbă dacă a[i]<a[p].

Sortarea în ordine crescătoare	Sortarea în ordine descrescătoare

`for(int i=1;i<n;i++)`
`{ int Min=a[i], p=i;`
`for(int j=i+1;j<=n;j++)`
`if(a[j]<Min){`
`Min=a[j];`
`p=j;`
`}`
`if(p!=i){`
`int t=a[i];`
`a[i]=a[p];`
`a[p]=t;`
`}`
`}`
	`for(int i=1;i<n;i++)`
	`{ int Max=a[i], p=i;`
	`for(int j=i+1;j<=n;j++)`
	`if(a[j]>Max){`
	`Max=a[j];`
	`p=j;`
	`}`
	`if(p!=i){`
	`int t=a[i];`
	`a[i]=a[p];`
	`a[p]=t;`
	`}`
	`}`

Tablou	Element minim	Poziția minimului	Noul tablou
(5, 2, **1**, 6, 7)	1	3	(1, 2, 5, 6, 7)
(1, **2**, 5, 6, 7)	2	2	(1, 2, 5, 6, 7)
(1, 2, 5, 6, 7)	5	3	(1, 2, 5, 6, 7)
(1, 2, 5, 6, 7)	6	4	(1, 2, 5, 6, 7)

Complexitate memorie: O(n) – se folosește un singur tablou unidimensional

Complexitate timp de execuție: $O(n^2)$.

Sortarea prin Inserție directă

Ideea de bază a algoritmului constă în:

- La fiecare pas i, i=2, 3, ..., n secvența a_1, a_2, ..., a_{i-1} este deja ordonată.
- Elementul a[i] va fi inserat în secvența a_1, a_2, ..., a_{i-1} astfel încât aceasta să devină ordonată crescător, în acest sens a[i] va fi comparat cu elementele a[j], unde j ia valori de la i-1, cât timp a[j]>a[i], totodată realizându-se și deplasarea la dreapta a acestora.

Sortarea în ordine crescătoare	Sortarea în ordine descrescătoare

`for(int i=2;i<=n;i++)`
`{ int x=a[i]; j=i-1;`
`while(j>=1&&a[j]>x){`
`a[j+1]=a[j];`
`j=j-1;`
`}`
`a[j+1]=x;`
`}`
	`for(int i=2;i<=n;i++)`
	`{ int x=a[i]; j=i-1;`
	`while(j>=1&&a[j]<x){`
	`a[j+1]=a[j];`
	`j=j-1;`
	`}`
	`a[j+1]=x;`
	`}`

Complexitate memorie: O(n) – se folosește un singur tablou unidimensional

Complexitate timp de execuție: $O(n^2)$.

Sortarea prin Intercalasare (Merge Sort)

Sortarea prin interclasare, sau **Mergesort** este o metodă eficientă de sortare a elementelor unui tablou, bazată pe următoarea idee: dacă prima jumătate a tabloului are elementele sortate și a doua jumătate are de asemenea elementele sortate, prin interclasare se va obține tabloul sortat.

Sortarea prin interclasare este un exemplu tipic de algoritm ***Divide et Impera***: se sortează o secvență delimitată de indicii st și dr:

- dacă st <= dr, problema este elementară, secvența fiind deja sortată
- dacă st < dr:
    - se împarte problema în subprobleme, identificând mijlocul secvenței, m = (st + dr) / 2;
    - se rezolvă subproblemele, sortând secvența delimitată de st și m, respectiv secvența delimitată de m+1 și dr – apeluri recursive;
    - se combină soluțiile, interclasând cele două secvențe; în acest fel, secvența delimitată de st și dr este sortată.

```
int v[100001], tmp[100001];
void MergeSort(int v[], int st, int dr)
{
     if(st < dr)
     {
             int m = (st + dr) / 2;
             MergeSort(v, st , m);
             MergeSort(v, m + 1 , dr);
             //Interclasare
             int i = st, j = m + 1, k = 0;
             while( i <= m && j <= dr )
                     if( v[i] < v[j])
                             tmp[++k] = v[i++];
                     else
                             tmp[++k] = v[j++];
```

```
            while(i <= m)
                    tmp[++k] = v[i++];
            while(j <= dr)
                    tmp[++k] = v[j++];
            for(i = st , j = 1 ; i <= dr ; i ++ , j ++)
                    v[i] = tmp[j];
    }
}
```

Stabilim antetul funcției recursive MergeSort(st, dr) cu semnificația: ordonează crescător elementele vectorului aflate pe poziții cuprinse între st și dr inclusiv. Considerăm că lucrăm cu vectorul global, iar de la un autoapel la altul doar stabilim intervalul de indici pe care îl procesăm.

- Descompunem problema curentă în două subprobleme pe principiul folosit la aplicațiile anterioare.

- Dacă se ajunge la o secvență cu un singur element (st == dr) șirul reprezentat de ea este sortat așa că nu facem nimic.

- La revenire considerăm că venim de jos în sus cu cele două secvențe sortate (și când se pornește de la frunze acestea reprezintă, cum am spus, secvențe sortate, deci pornim corect) așa că rămâne să le reunim pentru a obține sortată toată secvența.

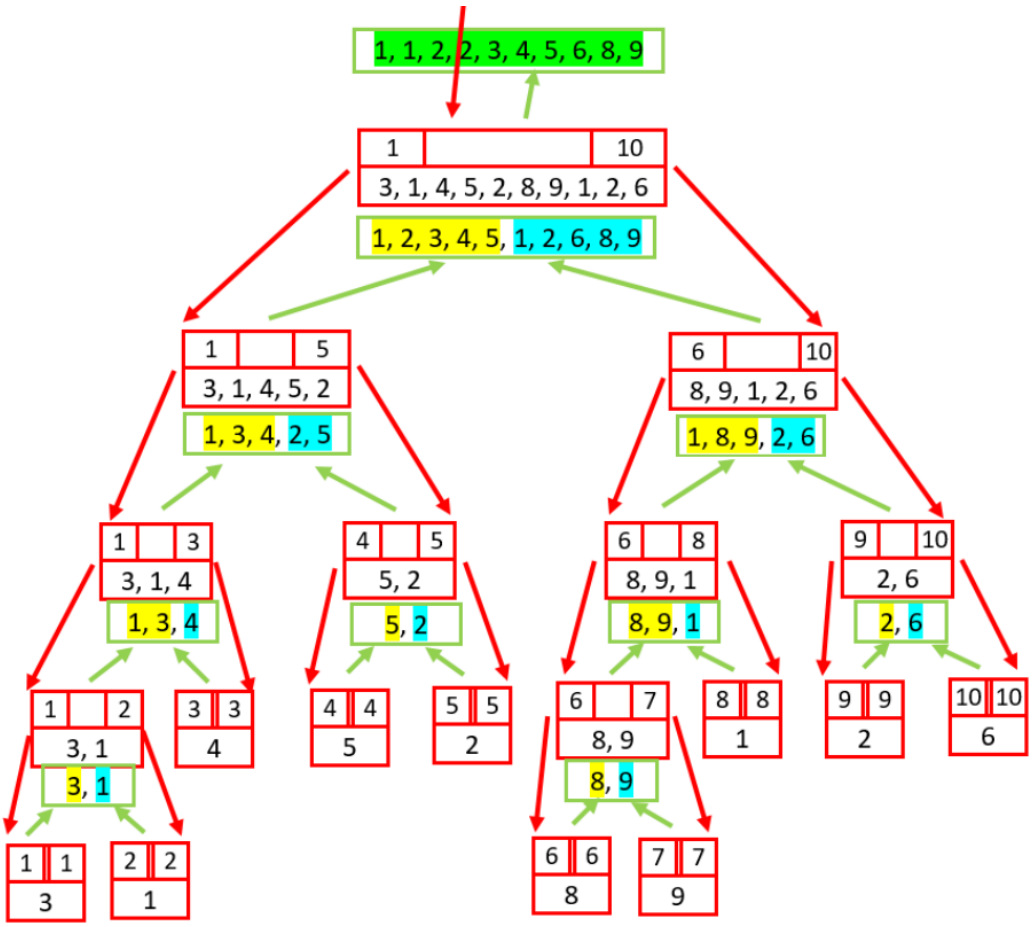
- Pentru a pune împreună sortate cele două jumătăți vom folosi un algoritm de interclasare optimă, despre care știm că are timp de executare de ordinul numărului total de elemente de interclasat. Funcția interclaseaza(st, mid, dr) este cea care realizează această etapă, de combinare. Ea primește sortată secvența de la st la mid și tot sortată pe cea de la mid+1 la dr urmând să le reunească sortate. Este esențial să observăm că nu putem lucra direct în vectorul v (s-ar putea ca primul element din dreapta sa fie mai mic decât primul din stânga și l-ar suprascrie). Așadar folosim un vector auxiliar, w în care construim rezultatul interclasării și apoi îi copiem elementele, în ordinea sortată, în v. Este foarte important să declarăm acest vector suplimentar global pentru a evita alocarea sa în memorie la fiecare inițiere de autoapel, lucru foarte costisitor.

- La fiecare nivel al arborelui se fac așadar interclasări și observăm că pe fiecare nivel orice element al vectorului apare o singură dată și, în plus, el participă la exact o interclasare. Concluzia este că

pe fiecare nivel timpul total de calcul este de ordin n. Pe de altă parte, avem număr de nivele de ordin log2n. Obținem așadat un algoritm cu timp de calcul nlog2n, chiar dacă este nevoie de spațiu dublu de memorie. Pentru a înțelege mai bine modul de funcționare a algoritmului, este utilă și urmărirea modului în care se avansează pe arborele de autoapeluri generat de un apel inițial MegeSort(1, n);

- Cu roșu am simbolizat ce se întâmplă la împărțirea în subprobleme, deci când se coboară în arborele autoapelurilor.

- Cu verde simbolizăm ce se întâmplă la revenire. Se observă acum cum cele două subsecvențe revin ordonate și apoi ele urmează să fie interclasate.



Complexitate memorie: O(2*n) – algoritmul folosește un vector suplimentar pentru a realiza interclasarea

Complexitate timp de execuție: O (n log n)

Sortarea Rapidă (Quick Sort)

QuickSort sau **Sortarea rapidă** este o metodă eficientă de sortare a unui tablou, descoperită în 1960 de programatorul britanic **C.A.R. Hoare**. Pentru un set de n valori oarecare algoritmul efectuează O(nlogn) comparații, dar în cazul cel mai nefavorabil se efectuează $O(n^2)$ comparații. Algoritmul este de tip **Divide et Impera**; el sortează o secvență a tabloului (inițial întreg tabloul), astfel:

- se alege un element special al listei, numit pivot;
- elementul pivot va fi plasat în poziția pe care o ocupă în vectorul ordonat crescător și elementele mai mici decât pivotul vor fi plasate în partea stângă a pivotului iar elementele mai mari în partea dreaptă.

- în timpul pivotării secvența va fi parcursă de la ambele capete, fie i capătul din stânga iar j capătul din dreapta:
    - la fiecare iterație, doar una dintre variabilele i și j se modifică: sau crește i, sau scade j
    - pivotul este elementul cu indicele care nu se modifică
- elementul pivot împarte secvența în două subsecvențe ce vor fi prelucrate în mod similar prin apel recursiv mai întâi cu secvența din stânga pivotului și apoi cu secvența din dreapta pivotului.

```
void QuickSort(int v[], int st, int dr)
{
  if(st < dr)
  {
        //pivotul este inițial v[st]
        int m = (st + dr) / 2;
        int aux = v[st];
        v[st] = v[m];
        v[m] = aux;
        int i = st , j = dr, d = 0;
```

```
        while(i < j)
        {
                if(v[i] > v[j])
                {
                        aux = v[i];
                        v[i] = v[j];
                        v[j] = aux;
                        d = 1 - d;
                }
                i += d;
                j -= 1 - d;
        }
        QuickSort(v, st , i - 1);
        QuickSort(v, i + 1 , dr);
  }
}
```

Rămâne acum să discutăm modul de plasare a pivotului.

- Pe parcursul operației, vom gestiona o pereche de indici i și j, cu i<=j astfel: pivotul este pe una dintre pozițiile i și j, elementele din sânga poziției i să fie mai mici decât pivotul, elementele din dreapta poziției j să fie mai mari decât pivotul. De exemplu, pe parcurs am putea să ne aflăm în starea:

		i			j		
3	1	5	4	3	7	6	9

Inițializarea o facem astfel: i=st; j=dr; Observăm că ea este o situație particulară dar care respectă restricțiile. La pasul curent comparăm v[i] cu v[j]. Întrucăt sunt în ordinea cerută nu vom interschimba, dar vom scădea pe j cu 1, i rămânând pe loc (acolo unde se află pivotul).

		i		j			
3	1	5	4	3	7	6	9

Acum v[i] > v[j], vom interschimba v[i] cu v[j], dar acum pivotul ajunge pe poziția j, deci va trebui să nu mai modificăm pe j, dar vom crește i (valoarea de pe vechea poziție a lui i este acum mai mică decât pivotul). Analizând atent, generalizăm astfel:

- Comparăm v[i] cu v[j], iar dacă sunt în ordinea cerută le lăsăm așa și continuăm cu modul de avansare setat curent. Dacă le interschimbăm, facem și comutarea în în celălalt mod de avansare.
- Observăm că avansarea se face prin incrementari de forma (i+d, j+(1-d)) unde d va fi fie 0 fie 1

- O modalitate compactă de a comuta dintr-un mod în altul este următoarea: de fiecare dată când interschimbăm doua elemente d ve deveni 1-d.

Anexe

```
#include <iostream>
using namespace std;
int n, v[100001], tmp[100001];
void citire (int n, int v[]){
    for(int i=1;i<=n;i++)
        cin>>v[i];
}
void afisare(int n, int v[]){
   for(int i=1;i<=n;i++)
      cout<<v[i]<<" ";
   cout<<"\n";
}
void comparare(int n, int v[]){
  for(int i=1;i<n;i++)
     for(int j=i+1;j<=n;j++)
       if(v[i]>v[j])
          swap(v[i],v[j]);

}
void selectie(int n, int v[]){
  for(int i=1;i<n;i++){
     int p=i;
     for(int j=i+1;j<=n;j++)
        if(v[j]<v[p])
           p=j;
     if(p!=i)
       swap(v[p],v[i]);
  }
```

```
}
void insertie(int n, int a[]){
  for(int i=2;i<=n;i++)
 { int x=a[i], j=i-1;
   while(j>=1&&a[j]>x){
       a[j+1]=a[j];
       j=j-1;
   }
 a[j+1]=x;
}
}
void bubble_sort(int n, int v[]){
    int ok;
    do
    {
        ok=1;
        for(int i=1; i<n; i++)
            if(v[i]>v[i+1])
            {
                swap(v[i],v[i+1]);
                ok=0;
            }
  }while(ok==0);
}
void MergeSort(int v[],int st, int dr){
if(st < dr)
            {
                        int m = (st + dr) / 2;
                        MergeSort(v, st , m);
                        MergeSort(v, m + 1 , dr);
                        //Interclasare
```

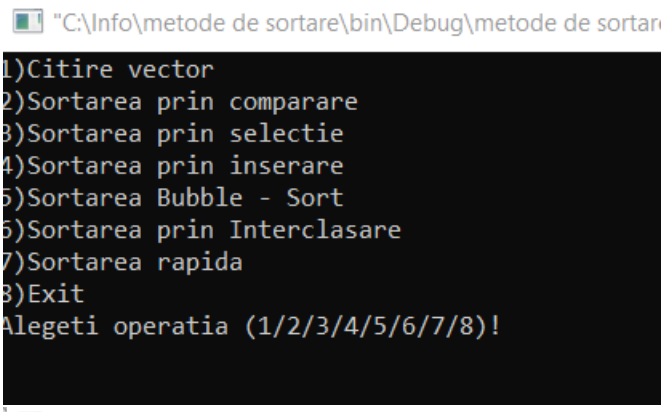
```
		int i = st, j = m + 1, k = 0;
		while( i <= m && j <= dr )
			if( v[i] < v[j])
				tmp[++k] = v[i++];
			else
				tmp[++k] = v[j++];
		while(i <= m)
			tmp[++k] = v[i++];
		while(j <= dr)
			tmp[++k] = v[j++];
		for(i = st , j = 1 ; i <= dr ; i ++ , j ++)
			v[i] = tmp[j];
	}
}
void QuickSort(int v[], int st, int dr)
{
	if(st < dr)
	{
		//pivotul este inițial v[st]
		int m = (st + dr) / 2;
		int aux = v[st];
		v[st] = v[m];
		v[m] = aux;
		int i = st , j = dr, d = 0;
		while(i < j)
		{
			if(v[i] > v[j])
			{
				aux = v[i];
				v[i] = v[j];
				v[j] = aux;
```

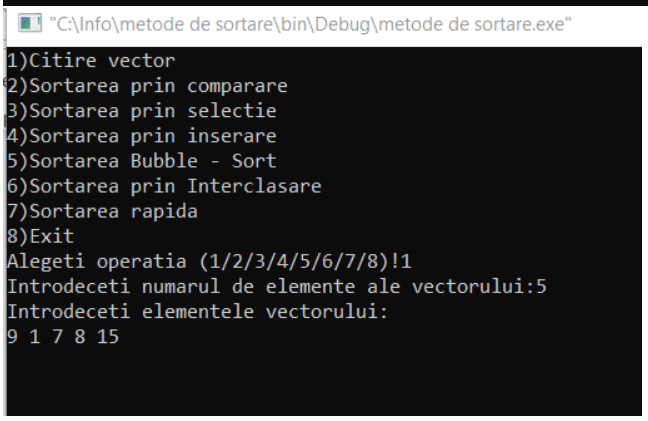
```
                    d = 1 - d;
                }
                i += d;
                j -= 1 - d;
            }
            QuickSort(v, st , i - 1);
            QuickSort(v, i + 1 , dr);
        }
}

int main()
{
int op;
do{
   system("cls");
   cout<<"1)Citire vector\n";
   cout<<"2)Sortarea prin comparare\n";
   cout<<"3)Sortarea prin selectie\n";
   cout<<"4)Sortarea prin inserare\n";
   cout<<"5)Sortarea Bubble - Sort\n";
   cout<<"6)Sortarea prin Interclasare\n";
   cout<<"7)Sortarea rapida\n";
   cout<<"8)Exit\n";
   cout<<"Alegeti operatia (1/2/3/4/5/6/7/8)!";
   cin>>op;
   switch (op){
     case 1: cout<<"Introdeceti numarul de elemente ale vectorului:";
           cin>>n;
           cout<<"Introdeceti elementele vectorului:\n";
           citire(n,v);
           system("pause");
```

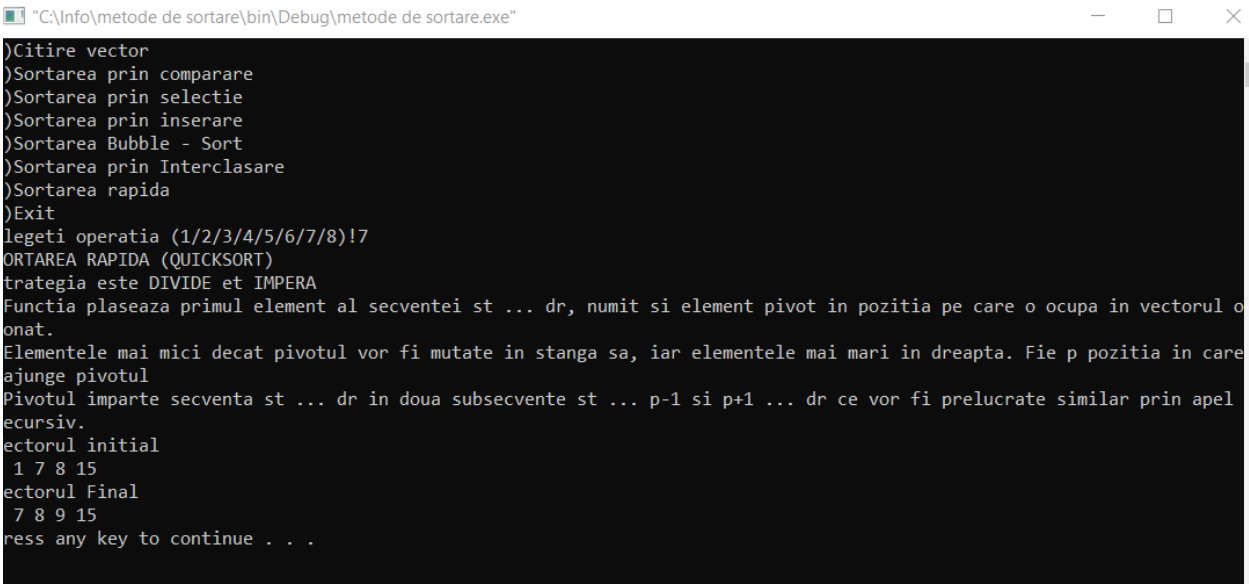
```
            break;
    case 2: cout<<"SORTAREA PRIN COMPARARE\n";
            cout<<"Ideea de baza a algoritmului consta in faptul ca:\n";
            cout<<"-Fiecare element al tabloului se compara cu toate elementele urmatoare\n";
            cout<<"-Daca un element urmator este mai mic decat elementul curent cele două
elemente se vor interschimba.\n";
            cout<<"Vectorul initial\n";
            afisare(n,v);
            comparare(n,v);
            cout<<"Vectorul final\n";
            afisare(n,v);
            system("pause");
            break;
    case 3:
           cout<<"SORTAREA SELECTIE\n";
            cout<<"Ideea de baza a algoritmului consta in faptul ca:\n";
            cout<<"-La fiecare pas i se determina minimul secevntei i ... n, fie p pozitia
minimului\n";
            cout<<"-Daca p este diferit de i (minimul nu se afla in pozitia i), se interschimba v[i] cu
v[p].\n";
            cout<<"Vectorul initial\n";
            afisare(n,v);
            selectie(n,v);
            cout<<"Vectorul Final\n";
            afisare(n,v);
            system("pause");
            break;
    case 4:
        cout<<"SORTAREA PRIN INSERTIE DIRECTA\n";
            cout<<"Ideea de baza a algoritmului consta in faptul ca:\n";
            cout<<"-La fiecare pas i secventa 1 ... i-1 este deja ordonata crescator\n";
```

```
        cout<<"- Se determina pozitia in care poate fi inserat v[i] in secventa 1 ... i-1 astfel incat secventa sa fie ordonata\n";
        cout<<"Vectorul initial\n";
        afisare(n,v);
        insertie(n,v);
        cout<<"Vectorul Final\n";
        afisare(n,v);
        system("pause");
        break;
    case 5:
      cout<<"SORTAREA BUBBLE-SORT\n";
        cout<<"Ideea de baza a algoritmului consta in faptul ca:\n";
        cout<<"-Cat timp vectorul nu este ordonat";
        cout<<"-Se parcurge vectorul si se compara doua elemente alaturate v[i] si v[i+1]. Daca nu sunt ordonate se interschimba\n";
        cout<<"Vectorul initial\n";
        afisare(n,v);
        bubble_sort(n,v);
        cout<<"Vectorul Final\n";
        afisare(n,v);
        system("pause");
        break;
    case 6:
      cout<<"SORTAREA PRIN INTERCLASARE\n";
        cout<<"Strategia este DIVIDE et IMPERA\n";
        cout<<"-Se imparte in mod repetat secventa in subsecvente pana se ajunge la secventa formate dintr-un singur element\n";
        cout<<"-Pe revenirea din recursivitate se ordoneaza fiecare secventa st ... dr prin interclasarea secventelor st ... mid si mid+1 ... dr, unde mid=(st+dr)/2\n";
        cout<<"Vectorul initial\n";
        afisare(n,v);
```

```
        MergeSort(v,1,n);
        cout<<"Vectorul Final\n";
        afisare(n,v);
        system("pause");
        break;
    case 7:
        cout<<"SORTAREA RAPIDA (QUICKSORT)\n";
        cout<<"Strategia este DIVIDE et IMPERA\n";
        cout<<"-Functia plaseaza primul element al secventei st ... dr, numit si element pivot in pozitia pe care o ocupa in vectorul oronat.\n";
        cout<<"-Elementele mai mici decat pivotul vor fi mutate in stanga sa, iar elementele mai mari in dreapta. Fie p pozitia in care ajunge pivotul\n";
        cout<<"-Pivotul imparte secventa st ... dr in doua subsecvente st ... p-1 si p+1 ... dr ce vor fi prelucrate similar prin apel recursiv.\n";
        cout<<"Vectorul initial\n";
        afisare(n,v);
        QuickSort(v,1,n);
        cout<<"Vectorul Final\n";
        afisare(n,v);
        system("pause");
        break;
   }
}while(op!=8);
   return 0;
}
```


"C:\Info\metode de sortare\bin\Debug\metode de sortar
1)Citire vector
2)Sortarea prin comparare
3)Sortarea prin selectie
4)Sortarea prin inserare
5)Sortarea Bubble - Sort
6)Sortarea prin Interclasare
7)Sortarea rapida
8)Exit
Alegeti operatia (1/2/3/4/5/6/7/8)!


"C:\Info\metode de sortare\bin\Debug\metode de sortare.exe"
1)Citire vector
2)Sortarea prin comparare
3)Sortarea prin selectie
4)Sortarea prin inserare
5)Sortarea Bubble - Sort
6)Sortarea prin Interclasare
7)Sortarea rapida
8)Exit
Alegeti operatia (1/2/3/4/5/6/7/8)!1
Introdeceti numarul de elemente ale vectorului:5
Introdeceti elementele vectorului:
9 1 7 8 15


"C:\Info\metode de sortare\bin\Debug\metode de sortare.exe"
)Citire vector
)Sortarea prin comparare
)Sortarea prin selectie
)Sortarea prin inserare
)Sortarea Bubble - Sort
)Sortarea prin Interclasare
)Sortarea rapida
)Exit
legeti operatia (1/2/3/4/5/6/7/8)!7
ORTAREA RAPIDA (QUICKSORT)
trategia este DIVIDE et IMPERA
Functia plaseaza primul element al secventei st ... dr, numit si element pivot in pozitia pe care o ocupa in vectorul o
onat.
Elementele mai mici decat pivotul vor fi mutate in stanga sa, iar elementele mai mari in dreapta. Fie p pozitia in care
ajunge pivotul
Pivotul imparte secventa st ... dr in doua subsecvente st ... p-1 si p+1 ... dr ce vor fi prelucrate similar prin apel
ecursiv.
ectorul initial
1 7 8 15
ectorul Final
7 8 9 15
ress any key to continue . . .

BIBLIOGRAFIE

1. https://www.pbinfo.ro/
2. **Programarea in limbajul C/C++ pentru liceu. autori Emanuela Cerchez, Marinel Serban, editura Polirom**