Algoritmi elementari

- **Metode de căutare**
  - **secvenţială**
  - **binară**

- **Metode de ordonare**
  - **metoda bulelor**
  - **metoda inserţiei**
  - **metoda selecţiei**
  - **metoda numărării**

- **Interclasare**

- **Analiza complexităţii unui algoritm**

Metode de căutare

Datele se află în memoria internă, într-un şir . Vom căuta un element în șir și returnăm poziţia elementului căutat (dacă acesta există).

Se dă (date): un șir de numere și numărul care se caută
Se cere (rezultate): poziția elemetului în șir

Dacă lucrăm cu șiruri ordonate este util să aflăm nu numai dacă există un element în șir ci şi să găsim în caz contrar locul în care ar trebui inserat un nou element, astfel încât să se păstreze ordinea existentă.

Căutare secvenţială

Ideea:

Verificăm pe rând elementele din șir până când găsim elementul căutat

Exemplu:

sir = 1, 7, 3, 5, 2
x = 3
Rezultat: 2

sir = 1, 7, 3, 5, 2
x = 4
Rezultat: -1

Căutare secvenţială - Implementare în C

```
/**
 * cauta elementul x in sirul v
 * returneaza pozitia elementului x in v
 */
int cautareSecventiala(int x, vector& v) {
	int poz = -1;
	int i = 0;
	while (i < v.n && poz == -1) {
		if (v.e[i] == x) {
			poz = i;
		} else {
			i++;
		}
	}
	return poz;
}
```

Căutare secvenţială - Implementare în Pascal

```
function cautSecv(v:vector;x:integer):integer;
var poz,i:integer;
begin
   poz:=0;
   i:=1;
   while (i<=v.n) and (poz=0) do
     if (v.e[i]=x) then poz:=i
       else i:=i+1;
   cautSecv:=poz;
end;
```

Analiza complexităţii unui algoritm

Putem analiza complexitatea unui algoritm în raport cu:

- durata de execuție
- cantitatea de memorie necesară;

Timpul necesar executiei unui program depinde de

- numarul operațiilor ce trebuie executate.
- calculatorul (hardware), sistemul de operare, limbajul folosit pentru implementare, etc

Cand analizam complexitatea ne raportam la numărul de operații executate, acesta depinde de datele de intrare

Analiză complexităţii ne dă ordinul lor de mărime a timpului de execuție (sau spațiului utilizat) în raport cu dimensiunea datelor de intrare

Complexitate căutare secvenţială: O(n)

Căutare binară

Doar pentru căutare în șir ordonat

Ideea:

Se determină în ce relaţie se află elementul aflat în mijlocul șirului cu elementul ce se caută. În urma acestei verificări căutarea se continuă doar într-o jumătate a șirului. În acest mod, prin înjumătăţiri succesive se micşorează volumul șirului rămas pentru căutare

Exemplu:

sir ordonat= 1, 3, 5, 7, 8
x = 3
Rezultat: 1

sir ordonat = 1, 3, 5, 7, 8
x = 4
Rezultat: 2

Căutare binară - Implementare în C

```
/**
 * cauta x in subsirul [st,dr]
 */
int cautBinar_rec(vector& v, int x, int st, int dr) {
	if (st >= dr - 1) {
		return dr;
	} else {
		int m = (st + dr) / 2;
		if (x <= v.e[m]) {
			return cautBinar_rec(v, x, st, m);
		} else {
			return cautBinar_rec(v, x, m, dr);
		}
	}
}

/*
 * v este ordonat crescator
 * rezultatul e pozitia pe care apare x sau pe care ar trebui inserat x
 */
int cautBinar(vector& v, int x) {
	if (x <= v.e[0]) {
		return 0;
	}
	if (x > v.e[v.n - 1])
		return v.n;
	else
		return cautBinar_rec(v, x, 0, v.n - 1);
}
```

Căutare binară - Implementare în Pascal

```
function cautBinar_rec(v:vector;x,st,dr:integer):integer;
var m:integer;
begin
  if (st>=dr-1) then begin
     cautBinar_rec:=dr
  end else begin
      m:=(st+dr) div 2;
      if x<=v.e[m] then cautBinar_rec:=cautBinar_rec(v,x,st,m)
        else cautBinar_rec:=cautBinar_rec(v,x,m,dr);
  end;
end;


function cautBinar(v:vector;x:integer):integer;
{v este ordonat crescator}
{rezultatul e pozitia pe care apare x sau pe care ar trebui inserat x}
var poz:integer;
begin
    if (x<=v.e[1]) then poz:=1
       else
         if (x>v.e[v.n]) then poz:=v.n+1
           else
             poz:=cautBinar_rec(v,x,1,v.n);
end;
```

Complexitate: $O(\log_2 n)$

Metode de ordonare

Rearanjare a unui șir de elemente aflate în memoria internă astfel încât elementele să fie ordonate crescător (eventual descrescător).

Date: un șir de elemente
Rezultate: Șirul ordonat crescator (descrescătpor)

Metoda bulelor

Ideea:
Compară două câte două elemente consecutive iar în cazul în care acestea nu se află în relaţia dorită, ele vor fi interschimbate. Procesul de comparare se va încheia în momentul în care toate perechile de elemente consecutive sunt în relaţia de ordine dorită

Exemplu:

şir: 4, 1, 6,-1, 2
Rezultat: -1, 1, 2, 4, 6

Metoda bulelor - Implementare în C

```
/**
 * sortare folosind metoda bulelor
 */
void sortareBule(vector& v) {
	bool ordonat = false;
	while (!ordonat) {
		ordonat = true; //presupunem sirul este ordonat
		for (int i = 1; i < v.n; i++) {
			if (v.e[i - 1] > v.e[i]) {
				//interschimbam
				int aux = v.e[i - 1];
				v.e[i - 1] = v.e[i];
				v.e[i] = aux;
				ordonat = false;
			}
		}
	}
}
```

Metoda bulelor - Implementare în Pascal

```
procedure sortBule(var v:vector);
var i,t:integer;
   ordonat:boolean;
begin
  repeat
    ordonat:=true;
    for i:=2 to v.n do
      if v.e[i-1]>v.e[i] then begin
          t:=v.e[i-1];
          v.e[i-1]:=v.e[i];
          v.e[i]:=t;
          ordonat:=false;
     end;
  until ordonat;
end;
```

Complexitate metoda bulelor: $O(n^2)$

Metoda inserţiei

Ideea:

traversăm elementele, inserăm elementul curent pe poziția corectă în subșirul care este deja ordonat. În acest fel elementele care au fost deja procesate sunt în ordinea corectă. După ce am traversat tot șirul toate elementele vor fi sortate.

Exemplu:

șir: 4, 1, 6,-1, 2
Rezultat: -1, 1, 2, 4, 6

Metoda inserţiei - Implementare în C

```
/**
 * sorteaza v
 */
void sortareInsertie(vector& v) {
      for (int i = 1; i < v.n; i++) {
            int ind = i - 1;
            int x = v.e[i];
            while (ind >= 0 && x < v.e[ind]) {
                  v.e[ind + 1] = v.e[ind];
                  ind--;
            }
            v.e[ind + 1] = x;
      }
}
```

Metoda inserţiei - Implementare în Pascal

```
procedure sortInsertie(var v:vector);
var i,j,ind,x:integer;
begin
   for i:=2 to v.n do begin
     ind:=i-1;
     x:=v.e[i];
     while (ind>0) and (x<v.e[ind]) do begin
       v.e[ind+1] := v.e[ind];
       ind:=ind -1;
     end
   end
end;
```

Complexitate: $O(n^2)$

Metoda selecţiei

Ideea:

Se determină poziţia elementului cu valoare minimă (respectiv maximă), după care acesta se va interschimba cu primul element. Acest procedeu se repetă pentru subșirul rămas, până când mai rămâne doar elementul maxim.

Exemplu:

șir: 4, 1, 6,-1, 2
Rezultat: -1, 1, 2, 4, 6

Metoda selecţiei - Implementare în C

```
/**
 * sortare folosind metoda selectiei
 */
void sortareSelectie(vector& v) {
	for (int i = 0; i < v.n - 1; i++) {
		//selectam minimul din restul sirului
		int minPoz = i;
		for (int j = i + 1; j < v.n; j++) {
			if (v.e[minPoz] > v.e[j]) {
				minPoz = j;
			}
		}
		//mutam minimul pe pozitia curenta
		if (minPoz != i) {
			int aux = v.e[minPoz];
			v.e[minPoz] = v.e[i];
			v.e[i] = aux;
		}
	}
}
```

Metoda selecţiei - Implementare în Pascal

```
procedure sortSelectie(var v:vector);
{selectia minimului}
var i,j,ind,t:integer;
begin
  for i:=1 to v.n-1 do begin
     ind:=i;
     for j:=i+1 to v.n do
       if v.e[j]<v.e[ind] then ind:=j;
     if i<ind then begin
          t:=v.e[i];
          v.e[i]:=v.e[ind];
          v.e[ind]:=t;
     end;
  end;
end;
```

Complexitate: $O(n^2)$

Metoda numărării

Ideea:

Se da un sir de elemente distincte. Pentru fiecare element numarăm elementele care sunt mai mici, astfel aflăm poziția elementului în șirul ordonat.

Exemplu:

șir: 4, 1, 6,-1, 2
Rezultat: -1, 1, 2, 4, 6

Metoda numărării - Implementare în C

```
/**
 * sorteaza v
 */
void sortareNumarare(vector& v) {
	//se face o copie a sirului
	vector v2 = v;
	for (int i = 0; i < v.n; i++) {
		//numaram cate elemente sunt mai mici
		int nrMaiMici = 0;
		for (int j = 0; j < v.n; j++) {
			if (v2.e[j] < v2.e[i]) {
				nrMaiMici++;
			}
		}
		v.e[nrMaiMici] = v2.e[i];
	}
}
```

Metoda numărării - Implementare în Pascal

```
procedure sortNumarare(var v:vector);
var i,k,j,x:integer;
    v1:vector;
begin
  v1:=v;
  for i:=1 to v1.n do begin
    {numar cate elemente sunt mai mici decat v.e[i]}
    k:=0;
    x:=v1.e[i];
    for j:=1 to v1.n do
      if v1.e[j]<x then k:=k+1;
    v.e[k+1]:=x;
  end;
end;
```

Complexitate: $O(n^2)$

Interclasare

Fiind date două șiruri de numere, ordonate crescător (sau descrescător), se cere să se obţină un șir care să fie de asemenea ordonat crescător (respectiv descrescător) şi care să fie formată din elementele șirurilor date.

Ideea:

Șirul rezultat se poate obţine direct (fără o sortare a șirului final) prin parcurgerea secvenţială a celor două șiruri, simultan cu generarea șirului cerut. Prin compararea a două elemente din listele de intrare se va decide care element va fi adăugat în lista de ieşire.

Exemplu:

șir 1: 1, 3, 4, 5
șir 1:-1, 2, 3
Rezultat: -1, 1, 2, 3, 3, 4, 5

Interclasare - Implementare în C

```
void adaugaSf(vector& v, int x) {
	v.e[v.n] = x;
	v.n++;
}

void interclasare(vector v1, vector v2, vector& rez) {
	rez.n = 0;
	int i = 0;
	int j = 0;
	while (i < v1.n && j < v2.n) {
		if (v1.e[i] <= v2.e[j]) {
			adaugaSf(rez, v1.e[i]);
			i++;
		} else {
			adaugaSf(rez, v2.e[j]);
			j++;
		}
	}
	for (int k = i; k < v1.n; k++) {
		adaugaSf(rez, v1.e[k]);
	}
	for (int k = j; k < v2.n; k++) {
		adaugaSf(rez, v2.e[k]);
	}
}
```

Interclasare - Implementare în Pascal

```
procedure adaugaSf(var v:vector;x:integer);
begin
  v.n:=v.n+1;
  v.e[v.n]:=x;
end;

procedure interclasare(v1,v2:vector;var rez:vector);
var i,j,k:integer;
begin
  rez.n:=0;
  i:=1;
  j:=1;
  while (i<=v1.n) and (j<=v2.n) do
    if (v1.e[i]<=v2.e[j]) then begin
       adaugaSf(rez,v1.e[i]);
       i:=i+1;
     end
      else begin
       adaugaSf(rez,v2.e[j]);
       j:=j+1;
      end;
  for k:=i to v1.n do
    adaugaSf(rez,v1.e[k]);
  for k:=j to v2.n do
    adaugaSf(rez,v2.e[k]);
end;
```

Complexitate: O(m + n)