ALGORITMI DE CĂUTARE ȘI SORTARE

CURSURI 12- 18.05.2021

Titular: Șef. Lucr. Dr. Mat. Cărbureanu Mădălina

Universitatea Petrol-Gaze din Ploiești

OBIECTIVE:

- **Algoritmi de căutare:**
  - **Căutarea secvențială și căutarea binară;**
- **Algoritmi de sortare:**
  - **Sortarea prin interschimbare și prin interclasare;**
  - **Sortarea prin selecție și prin numărare;**
  - **Sortarea prin inserție și sortarea rapidă.**

Algoritmi de căutare

- **Căutarea secvențială:**
  - presupune verificarea apartenenței unui element oarecare (unei chei citite de la tastatură) la un șir (vector, tablou unidimensional) de elemente de același tip;
  - presupune identificarea cheii căutate succesiv în componentele unui șir neordonat sau ordonat (crescător sau descrescător).

```
(1) citire de la tastatură valoare căutată (val);
(2) inițializare variabilă booleană găsit←0;
(3) pentru i←1,..., n execută
(4)     dacă val==a[i] atunci găsit←1şi poziție←i;
(5) sfârşit;
(6) dacă găsit==1 atunci afişează poziția (poziție) valorii căutate în şir;
(7)     altfel afisează mesaj "Valoarea căutată nu este element al şirului";
```

Fig. 1. Căutare secvențială-pseudocod

Algoritmi de căutare

- **Exemplificare mod lucru algoritm căutare secvențială:**
- Fie șirul a[10] cu *n=5* elemente numere întregi: a[1]=1, a[2]=5, a[3]=2, a[4]=4 și a[5]=3. Pași necesari în parcurgerea algoritmului ce stă la baza căutării secvențiale a valorii (cheii) *val=4* sunt următorii:
- *găsit←0;*
- pentru *i*=1 se testează dacă valoarea căutată 4==a[1] (a[1]=1); condiția nu este îndeplinită, motiv pentru care se incrementează *i* cu o unitate, deci *i*=2;
- se verifică dacă *i*=2≤*n (n*=5*)* și dacă este îndeplinită, atunci se compară valoarea căutată 4==a[2] (a[2]=5); condiția nu este îndeplinită, motiv pentru care se incrementează *i* cu o unitate, deci *i*=3;
- se verifică dacă *i*=3≤*n (n*=5*)* și dacă este îndeplinită, atunci se compară valoarea căutată 4==a[3] (a[3]=2); condiția nu este îndeplinită, motiv pentru care se incrementează *i* cu o unitate, deci *i*=4;
- se verifică dacă *i*=4≤*n (n*=5*)* și dacă este îndeplinită, atunci se compară valoarea căutată 4==a[4] (a[4]=4); condiția este îndeplinită, motiv pentru care algoritmul se încheie cu succes, *găsit←1* și se afișează poziția *i* (*poziție←i*) pe care se regăsește valoarea căutată în șirul dat.
- în caz contrar algoritmul se încheie cu insucces (valoarea căutată nu este element al șirului, *găsit←0*).

Algoritmi de căutare

- **Căutarea binară:**
  - presupune căutarea unei valori (chei) citite de la tastatură într-un șir cu *n* elemente numere întregi, șirul fiind neapărat ordonat crescător.

(1) citire de la tastatură valoare căutată (val);
(2) inițializare variabilă booleană găsit←0;
(3) inițializare ind_1←1;
(4) inițializare ind_2←n;
(5) execută
(6) m←(ind_1+ ind_2)/2;
(7) dacă (val==a[m]) atunci găsit←1; poziție←m; break;sfârșit;
(8) altfel dacă (val>a[m]) ind1←m+1;
(9) altfel ind2←m-1;
(10) sfârșit;
(11) cât timp $ind_1 \leq ind_2$ și not(găsit);

Fig. 2. Căutarea binară - pseudocod

Algoritmi de căutare

- **Exemplificare mod lucru algoritm căutarea binară**:
- Șir a[10] ordonat crescător cu n=6 elemente numere întregi, respectiv a[1]=1, a[2]=2, a[3]=3, a[4]=4, a[5]=5 și a[6]=6. Pași necesari în parcurgerea algoritmului ce stă la baza căutării binare a valorii (cheii) citite de la tastatură *val*=5 sunt următorii:
- *găsit*←0*; ind*$_1$←1*; ind*$_2$←6;
- se identifică jumătatea șirului, respectiv $m \leftarrow (ind_1 + ind_2)/2$, *respectiv* $m \leftarrow [3.5] = 3$ și se împarte șirul inițial în două subșiruri (subșir1: [1, 2, 3]; subșir 2: [4, 5, 6]) după valoarea lui *m;*
- se verifică dacă 5==a[3] (a[3]=3); după cum se poate observa, condiția nu este îndeplinită, motiv pentru care se testează dacă (5>a[3]) (a[3]=3); condiția este îndeplinită, deci căutarea se va efectua în cel de-al doilea subșir, respectiv [4, 5, 6] prin modificarea $ind_1 \leftarrow 4$, valoarea rămânând aceeași pentru ind_2 ($ind_2 \leftarrow 6$);
- se caută *val*=5 în subșirul 2 [4, 5, 6], se identifică jumătatea acestuia, respectiv $m \leftarrow [5]$ și se împarte și acest subșir, la rândul lui, în alte două subșiruri (subșir$_{21}$: 4; subșir$_{22}$: 6);
- se verifică dacă valoarea căutată 5 este chiar jumătatea subșirului 2 [4, 5, 6], respectiv 5==a[5] (a[5]=5); după cum se poate observa, condiția este îndeplinită, variabila *găsit*←1 și *poziție*←5 caz în care algoritmul se încheie cu succes (valoarea cautată este element al șirului inițial și eventual se afișează și poziția pe care aceasta valoare se găsește în șir); altfel, și acest subșir se împărțea în alte două subșiruri, iar algoritmul se relua pentru subșirurile obținute;
- algoritmul se aplică atâta timp cât valoarea căutată nu este identificată printre elementele șirului dat (*not(găsit)*) și $ind_1 \leq ind_2$.

Algoritmi de sortare

- **Sortarea prin interschimbare (Bubble Sort):**
  - deși este cea mai lentă metodă de sortare (nu este cea mai eficientă), din punct de vedere conceptual este cea mai simplă;
  - algoritmul constă în compararea succesivă a câte doi termeni din vectorul ce se dorește a se sorta (crescător sau descrescător); inițial, se compară primii doi termeni ai șirului și dacă valoarea numerică a primului termen este mai mare decât valoarea numerică a celui de-al doilea termen, atunci se interschimbă pozițiile acestor doi termeni, astfel încât primul termen trece pe poziția pe care se află al doilea termen în cadrul tabloului ce trebuie sortat.;
  - algoritmul se repetă până când nu mai există nicio neconcordanță între valorile termenilor din vector (nu a mai fost efectuată nicio interschimbare).

Algoritmi de sortare

- **Sortarea prin interschimbare (Bubble Sort):**

Varianta 1	Varianta 2	Varianta 3
(1) execută	pentru i←1,..., n-1	pentru i←1,..., n-1
(2) t←0;	pentru j←i+1,...,n execută	pentru j←1,...,n-i execută
(3) pentru i←1,..., n-1 execută	dacă v[i]>v[j] atunci	dacă v[j]>v[j+1] atunci
(4) dacă v[i]>v[i+1] atunci	aux←v[i];	aux←v[j];
(5) aux←v[i];	v[i] ←v[j];	v[j] ←v[j+1];
(6) v[i] ←v[i+1];	v[j] ←aux;	v[j+1] ←aux;
(7) v[i+1] ←aux;	sfârşit;	sfârşit;
(8) t←1;	sfârşit;	sfârşit;
(9) sfârşit;		
(10) sfârşit;		
(11) sfârşit;		
(12) cât timp t<>0;		

Fig. 3. Variante implementare Bubble Sort- pseudocod

Exemplificare alg. Bubble Sort (varianta 2), pag. 214.

Algoritmi de sortare

- **Sortarea prin interclasare (Merge Sort):**
  - constă în construirea unui șir cu $n+m$ elemente numere întregi, format cu elementele a două șiruri ordonate crescător (condiție obligatorie), primul șir cu n elemente și al doilea cu m elemente;
  - presupune fuzionarea a două șiruri ordonate crescător, în mod eficient, astfel încât să rezulte un șir de asemenea ordonat crescător, care să conţină elementele celor două șiruri inițiale;
  - în acest sens, se va nota cu i indicele elementului la care s-a ajuns în primul vector (a[10] cu n elemente), cu j indicele elementului la care s-a ajuns în al doilea vector (b[10] cu m elemente) și cu k indicele elementului care urmează a fi trecut în cel de-al treilea vector (c[20] cu $n+m$ elemente).

Algoritmi de sortare

- **Sortarea prin interclasare (Merge Sort):**
- **Algoritmul Merge Sort** are la bază comparaţiile repetate între elementele şirului a[10] cu elementele şirului b[10], atâta timp cât nu s-a atins sfârşitul primului şir ($i \leq n$) şi nici sfârşitul celui de-al doilea şir ($j \leq m$) , astfel:
- dacă a[i]>b[j] ($i=1,\ldots, n; j=1,\ldots,m$), atunci se introduce în şirul c[k] elementul b[j] şi indicele j este incrementat cu o unitate ($j \leftarrow j+1$), respectiv se trece la următorul element din şirul b[10];
- dacă a[i]<b[j] ($i=1,\ldots, n; j=1,\ldots,m$), atunci se introduce în şirul c[k] elementul a[i] şi indicele i este incrementat cu o unitate ($i \leftarrow i+1$), respectiv se trece la următorul element din şirul a[10];
- în ambele situaţii mai sus menţionate, se incrementează indicele k cu o unitate ($k \leftarrow k+1$), respectiv creşte numărul de elemente din şirul rezultat c[20];
- dacă mai rămân elemente în şirul a[10] care nu mai au cu ce elemente din şirul b[10] să fie comparate, şirul b[10] fiind epuizat ca elemente testate, acestea pur şi simplu se adaugă la sfârşitul şirului c[20], iar algoritmul se încheie;
- dacă mai rămân elemente în şirul b[10] care nu mai au cu ce elemente din şirul a[10] să fie comparate, şirul a[10] fiind epuizat ca elemente testate, acestea pur şi simplu se adaugă la sfârşitul şirului c[20], iar algoritmul se încheie;
- se afişează şirul ordonat crescător obţinut, respectiv c[20].

Algoritmi de sortare

(1) i←1; j←1; k←1;
(2) cât timp (i≤n) şi (j≤m) execută
(3) dacă (a[i]>b[j]) atunci c[k] ←b[j]; j←j+1;
(4) altfel c[k] ←a[i]; i←i+1;
(5) k←k+1;
(6) sfârşit;
(7) pentru l←i,..., n execută
(8) c[k] ←a[l]; k←k+1;
(9) sfârşit;
(10) pentru l←j,..., m execută
(11) c[k] ←b[l]; k←k+1;
(12) sfârşit;
(13) pentru k←i,..., n+m execută
(14) afisează şirul c[k];

Fig. 4. Sortare Merge Sort-pseudocod

Algoritmi de sortare

- **Exemplificare algoritm Merge Sort:**
- Fiind date două șiruri ordonate crescător, respectiv șirul a[10] cu n=7 elemente numere întregi (1, 3, 5, 7, 8, 10, 11) și șirul b[10] cu m=5 elemente numere întregi (2, 4, 6, 9, 12). Șirul c[20] cu k=n+m elemente se obține prin parcurgerea următorilor pași:

- i←1; j←1; k←1;
- se verifică dacă i(1)≤n(7) și j(1)≤m(5); condiția este îndeplinită atunci se verifică dacă a[1](1)>b[1](2); condiția nu este îndeplinită; apoi se verifică dacă a[1](1)<b[1](2); condiția este îndeplinită, atunci c[1] ←a[1](1), i←2; k←2;
- se verifică dacă i(2)≤n(7) și j(1)≤m(5); condiția este îndeplinită, atunci se verifică dacă a[2](3)>b[1](2); condiția este îndeplinită și atunci c[2] ←b[1](2), j←2; k←3;
- se verifică dacă i(2)≤n(7) și j(2)≤m(5); condiția este îndeplinită, atunci se verifică dacă b[2](4)>a[2](3); condiția este îndeplinită și atunci c[3] ←a[2](3), i←3; k←4;
- se verifică dacă i(3)≤n(7) și j(2)≤m(5); condiția este îndeplinită, atunci se verifică dacă a[3](5)>b[2](4); condiția este îndeplinită și atunci c[4] ←b[2](4), j←3; k←5;
- se verifică dacă i(3)≤n(7) și j(3)≤m(5); condiția este îndeplinită, atunci se verifică dacă b[3](6)>a[3](5); condiția este îndeplinită și atunci c[5] ←a[3](5), i←4; k←6;

Algoritmi de sortare

- se verifică dacă i(4)≤n(7) şi j(3)≤m(5); condiţia este îndeplinită, atunci se verifică dacă a[4](7)>b[3](6); condiţia este îndeplinită şi atunci c[6] ←b[3](6), j←4; k←7;
- se verifică dacă i(4)≤n(7) şi j(4)≤m(5); condiţia este îndeplinită, atunci se verifică dacă b[4](9)>a[4](7); condiţia este îndeplinită şi atunci c[7] ←a[4](7), i←5; k←8;
- se verifică dacă i(5)≤n(7) şi j(4)≤m(5); condiţia este îndeplinită, atunci se verifică dacă a[5](8)>b[4](9); condiţia nu este îndeplinită şi atunci se verifică dacă a[5](8)<b[4](9); condiţia este îndeplinită şi c[8] ←a[5](8), i←6; k←9;
- se verifică dacă i(6)≤n(7) şi j(4)≤m(5); condiţia este îndeplinită, atunci se verifică dacă a[6](10)>b[4](9); condiţia este îndeplinită şi atunci c[9] ←b[4](9), j←5; k←10;
- se verifică dacă i(6)≤n(7) şi j(5)≤m(5); condiţia este îndeplinită, atunci se verifică dacă b[5](12)>a[6](10); condiţia este îndeplinită şi atunci c[10] ←a[6](10), i←7; k←11;
- se verifică dacă i(7)≤n(7) şi j(5)≤m(5); condiţia este îndeplinită, atunci se verifică dacă a[7](11)>b[5](12); condiţia nu este îndeplinită şi atunci se verifică dacă a[7](11)<b[5](12); condiţia este îndeplinită şi atunci c[11] ←a[7](11), i←8; k←12;
- a mai rămas un element în cel de-al doilea şir, respectiv b[5](12) care nu mai are cu ce element din primul şir să fie comparat, motiv pentru care se adaugă pur şi simplu la sfârşitul celorlalte elemente din şirul rezultat, respectiv c[12] ←b[5](12);
- se verifică dacă i(8)≤n(7) şi j(5)≤m(5); condiţia nu mai este îndeplinită, toate elementele celor două şiruri au fost epuizate, motiv pentru care algoritmul se încheie, şirul obţinu c[12] fiind şirul ordonat crescător [c[1] ←a[1](1), c[2] ←b[1](2), c[3] ←a[2](3), c[4] ←b[2](4), c[5] ←a[3](5), c[6] ←b[3](6), c[7] ←a[4](7), c[8] ←a[5](8), c[9] ←b[4](9), c[10] ←a[6](10), c[11] ←a[7](11), c[12] ←b[5](12)].

Algoritmi de sortare

- **Sortarea prin selecţie (Selection Sort):**
- face parte din familia de tehnici de sortare care au la bază ideea selecţiei repetate de elemente;
- acest tip de sortare presupune selecţia repetată fie a valorii maxime, fie a valorii minime dintr-un tablou unidimensional;
- sortarea prin selecţia minimului sau a maximului este o metodă de ordonare prin selectarea unui element şi plasarea lui pe poziţia sa finală direct în vectorul considerat (fie acesta vectorul V[30] cu elementele $v_1, v_2, \ldots, v_n$);
- de exemplu, în caz de ordonare crescătoare a elementelor vectorului, pornind de la primul element se caută valoarea minimă din vector; aceasta se plasează pe prima poziţie din vector printr-o interschimbare realizată între elementul de pe prima poziţie şi elementul minim de pe poziţia *k*; apoi, se reia algoritmul pornind de la a doua poziţie şi se caută minimul între elementele $v_2, \ldots, v_n$; minimul găsit se interschimbă cu al doilea element din şir, dacă este cazul; procedeul se continuă până la ultimul element din vector.

Algoritmi de sortare

```
(1) pentru i←0, n-1 execută
(2) pivot←i;
(3) pentru j←i+1, n execută
(4)       dacă v[j]<v[pivot] atunci
(5)             pivot←j;
(6)             aux←v[i];
(7)             v[i] ←v[pivot];
(8)             v[pivot] ←aux;
(9) sfârşit;
```

Fig. 5. Sortare prin selecție minim - pseudocod.

Algoritmi de sortare

- **Exemplificare mod lucru algoritm de sortare prin selecţia repetată a minimului** a elementelor din vectorul V=[3, 2, 0, 1, 4], cu n=5 elemente.
- i←0, pivot←0, j←1; dacă v[1](2)<v[0](3) atunci pivot←1 şi se interschimbă v[0] cu v[1], vectorul V devine [2, 3, 0, 1, 4];

- i←0, pivot←0, j←2; dacă v[2](0)<v[0](2) atunci pivot←2 şi se interschimbă v[0] cu v[2], vectorul V devine [0, 3, 2, 1, 4];
- i←0, pivot←0, j←3; nu este îndeplinită condiţia v[3](1)<v[0](0);
- i←0, pivot←0, j←4; nu este îndeplinită condiţia v[4](4)<v[0](0);
- i←1, pivot←1, j←2; dacă v[2](2)<v[1](3) atunci pivot←2 şi se interschimbă v[2] cu v[1], vectorul V devine [0, 2, 3, 1, 4];
- i←1, pivot←1, j←3; dacă v[3](1)<v[1](2) atunci pivot←1 şi se interschimbă v[3] cu v[1], vectorul V devine [0, 1, 3, 2, 4];
- i←1, pivot←1, j←4; nu este îndeplinită condiţia dacă v[4](4)<v[1](1);
- i←2, pivot←2, j←3; dacă v[3](2)<v[2](3) atunci pivot←3 şi se interschimbă v[3] cu v[2], vectorul V devine [0, 1, 2, 3, 4] şi algoritmul se opreşte pentru că vectorul este deja ordonat crescător.

Algoritmi de sortare

- **Sortarea prin numărare (Count Sort):**
- constă în găsirea pentru fiecare element a_i $(i=1,\ldots,n)$ al unui vector $A=\{a_1,\ldots, a_n\}$ cu n elemente numere întregi, a numărului de elemente din vector mai mici decât elementul considerat a_i $(i=1,\ldots,n)$; dacă există k elemente mai mici decât a_i, atunci clar elementul a_i trebuie să se afle pe poziția $k+1$ în vectorul sortat.
- Se utilizează trei tipuri de vectori, și anume:
  - vectorul inițial $A=\{a_1,\ldots, a_n\}$ ce se dorește a se sorta;
  - vectorul $B=\{b_1,\ldots, b_n\}$ în care se reține pentru fiecare element a_i $(i=1,\ldots,n)$ al vectorului A, numărul de elemente mai mici decât a_i;
  - un vector auxiliar $C=\{c_1,\ldots, c_n\}$ (copia vectorului A) în care se salvează elementele vectorului inițial A.
- la finalul algoritmului, se rescriu în ordine crescătoare elementele vectorului A pe baza valorilor memorate în vectorii B și C.

Algoritmi de sortare

```
(1)  pentru i←0, n-1 execută
(2) c[i]←a[i];
(3) pentru i←0,n-1 execută
(4) pentru j←i+1, n-1 execută
(5)       dacă a[i]<a[j] atunci b[j] ← b[j]+1;
(6)             altfel b[i] ← b[i]+1;
(7) pentru i←0, n-1 execută
(8)       a[b[i]] ← c[i];
(9) pentru i←0, n-1 execută
(10)      afişează a[i];
```

Fig. 6. Sortarea prin numărare-pseudocod

Algoritmi de sortare

- **Exemplificare algoritm sortare prin numărare:**
- fie vectorul *A*=[10, 20, 3, 6, 4]; pentru sortarea crecătoare a vectorului dat folosind **sortarea prin numărare**, paşii sunt următorii:

- se realizează o copie a vectorul *A* în vectorul *C*, deci *C*=[10, 20, 3, 6, 4];
- se determină elementele vectorului *B* astfel:
  - pentru *i*←1 se contorizează câte elemente mai mici decât a[1](10) sunt în vectorul *A*; numărul obţinut, respectiv valoarea 3 se trece în vectorul B, deci B=[3];
  - pentru i←2 se contorizează câte elemente mai mici decât a[2](20) sunt în vectorul A; numărul obţinut, respectiv valoarea 4 se adaugă în vectorul *B*, deci *B*=[3, 4];
  - pentru i←*3* se contorizează câte elemente mai mici decât a[3](3) sunt în vectorul *A*; numărul obţinut, respectiv valoarea 0 se adaugă în vectorul *B*, deci *B*=[3, 4, 0];
  - pentru *i*←4 se contorizează câte elemente mai mici decât a[4](6) sunt în vectorul *A*; numărul obţinut, respectiv valoarea 2 se adaugă în vectorul *B*, deci *B*=[3, 4, 0, 2];
  - pentru *i*←5 se contorizează câte elemente mai mici decât a[5](4) sunt în vectorul *A*; numărul obţinut, respectiv valoarea 1 se adaugă în vectorul *B*, deci *B*=[3, 4, 0, 2, 1];

Algoritmi de sortare

- se completează elementele vectorului *A* (folosind informațiile din vectorii *B* și *C*), respectiv *A[B[i]]=C[i]* (*i=0,…,n-1*), astfel:

- pentru *i*←0, A[B[0]]=C[0], A[3]=10, respectiv se adaugă în vectorul *A* valoarea 10 pe poziția 3, deci *A*=[_,_,_,10,_];
- pentru *i*←1, A[B[1]]=C[1], A[4]=20, respectiv se adaugă în vectorul *A* valoarea 20 pe poziția 4, deci *A*=[_,_,_,10,20];
- pentru *i*←2, A[B[2]]=C[2], A[0]=3, respectiv se adaugă în vectorul *A* valoarea 3 pe poziția 0, deci *A*=[3,_,_,10,20];
- pentru *i*←3, A[B[3]]=C[3], A[2]=6, respectiv se adaugă în vectorul *A* valoarea 6 pe poziția 2, deci *A*=[3,_,6,10,20];
- pentru *i*←4, A[B[4]]=C[4], A[1]=4, respectiv se adaugă în vectorul *A* valoarea 4 pe poziția 1, deci *A*=[3, 4 ,6, 10, 20];
- șirul A (A=[3, 4 ,6, 10, 20]) este ordonat crescător, algoritmul se încheie.

Algoritmi de sortare

- **Sortarea prin inserţie (Insertion Sort):**
- Fie vectorul $A=\{a_1,\ldots, a_n\}$; alg. presupune împărţirea elementelor vectorului iniţial în două liste, respectiv o listă sortată formată din primul element al vectorului şi o listă nesortată formată din restul elementelor vectorului (din cele *n-1* elemente rămase).

```
(1)  pentru i←1, n-1 execută
(2)       j←i-1; terminat←0; aux←x[i];
(3)      execută
(4)           dacă aux<x[j] atunci
(5)                          x[j+1] ←x[j]; j←j-1;
(6)           altfel terminat=1;
(7)       sfârşit;
(8)      cât timp j<>-1 and not(terminat);
(9)  x[j+1] ←aux;
(10) sfârşit;
```

Fig. 7. Sortarea prin inserţie - pseudocod

Algoritmi de sortare

- **Exemplificare sortare prin inserţie (Insertion Sort):**
- Fie vectorul $A=\{a_1,\ldots, a_n\}$; alg. presupune împărţirea elementelor vectorului iniţial în două liste, respectiv o listă sortată formată din primul element al vectorului şi o listă nesortată formată din restul elementelor vectorului (din cele *n-1* elemente rămase).

- De exemplu, fie vectorul iniţial $A=[4, 3, 1, 6, 5]$. Lista ordonată L_1 va conţine primul element din vectorul A ($L_1=(4)$), iar lista L_2 va conţine restul de 4 elemente rămase, respectiv $L_2=(3, 1, 6, 5)$. Sortarea crescătoare prin inserţie a elementelor vectorului A presupune parcurgerea următorilor paşi:
- se caută poziţia primului element din lista L_2 (3) în lista ordonată $L_1=(4)$ şi se deplasează cu o poziţie spre dreapta primul element, obţinându-se şirul $A=[3, 4, 1, 6, 5]$;
- se caută poziţia elementului 1 din lista L_2 în lista ordonată $L_1=(3, 4)$ şi se deplasează cu o poziţie spre dreapta elementele 3 şi 4, obţinându-se şirul $A=[1, 3, 4, 6, 5]$;
- pentru că elementul 6 din L_2 este mai mare strict decât elementul 4 din $L_1=(1, 3, 4)$, nu se vor realiza deplasări, vectorul rămânând neschimbat, respectiv $A=[1, 3, 4, 6, 5]$;
- se caută poziţia elementului 5 din lista L_2 în lista ordonată $L_1=(1, 3, 4, 6)$ şi se deplasează cu o poziţie spre dreapta elementul 6, obţinându-se şirul $A=[1, 3, 4, 5, 6]$.
- şirul obţinut este sortat crescător, algoritmul de sortare prin inserţie se încheie.

Algoritmi de sortare

- **Sortarea rapidă (Quick Sort):**
- În cadrul acestei sortări, fiind dat un șir cu *n* elemente numere întregi, se alege un element (așa numit pivot), iar restul elementelor din șir sunt împărțite în două subșiruri:
  - un subșir cu elementele mai mici decât elementul de partiție;
  - un subșir cu elementele mai mari sau egale cu acesta;
- Același proces este apoi aplicat recursiv la cele două subșiruri;
- Când un subșir are mai puțin de două elemente, nu are nevoie de nicio sortare, aceasta oprind recursivitatea.

Algoritmi de sortare

- Pentru rezolvare, s-a definit funcţia *int poz(int l_i, int l_d)*, care tratează porţiunea din vector cuprinsă între limita inferioară l_i şi limita superioară l_d.
- Această funcţie a fost concepută astfel:
  - iniţial indicele *i* primeşte valoarea l_i, iar *j* valoarea l_d;
  - se setează modul de lucru, respectiv modul de lucru 1, în care *i* rămâne constant, iar *j* scade cu o unitate;
  - atâta timp cât $i < j$, se execută:
    - dacă a[i]>a[j], atunci se inversează cele două elemente şi se schimbă modul de lucru, respectiv se trece în modul de lucru 2, în care *i* creşte cu o unitate, iar *j* rămâne constant;
    - *i* şi *j* se modifică în funcţie de modul de lucru în care se află programul;
  - funcţia returnează *i*, adică prima conponentă a[l_i] este poziţionată pe o poziţie *i* cuprinsă între l_i şi l_s, astfel încât toate elementele vectorului cuprinse între l_i şi *i-1* să fie mai mici sau egale decât a[i], iar toate elementele vectorului cuprinse între *i+1* şi l_d să fie mai mari sau egale decât a[i].
- De asemenea, s-a utilizat şi funcţia recursivă directă *void QS(int li, int ld)*, în care:
  - se apelează *poz(li, ld);*
  - se apelează QS pentru *li* şi *i-1;*
  - se apelează QS pentru *i+1, ld.*
- **Implementare recursivă Quick Sort, pag. 37.**

- Testarea aplicațiilor 9.2.1, 9.2.2 și 9.2.3, pag. 225-229;
- Aplicații propuse: aplicațiile din cadrul lucrării de laborator nr. 13, pag. 229;
- **Obs: se va utiliza cartea "*Elemente de proiectarea algoritmilor. Ghid teoretic și practic*", Șef lucr. dr. mat. Cărbureanu Mădălina, Editura Universității Petrol-Gaze din Ploiești, 2021.**

Să vă fie de folos și spor la lucru!