Lucrarea nr. 4 - Algoritmi de sortare şi ordonare

Breviar teoretic

Un algoritm de sortare este o metoda prin care se aranjează elementele unui tablou într-o ordine precisă. Cele mai folosite tipuri de date în cadrul acestor metode de sortare sunt tipul de ordin numeric şi tipul lexicografic. Eficienta metodelor de sortare este importantă deoarece se pune un mare accent pe optimizarea altor algoritmi (cum sunt algoritmii de căutare) care necesită sortarea tablourilor cu care lucrează, devenind astfel mai eficienti.
Dacă un algoritm realizează o sortare prin procedee complexe el poate avea o comportare avantajoasă, comportare mulțumitoare sau o comportare dezavantajoasa. Prin **complexitatea metodei (calculul)** se întelege efectuarea operațiilor de comparare a valorii unui termen oarecare din tabloul de marime *n*, cu celelelte *n* valori ale termenilor prezenți în cadrul aceluiaşi tablou. Pentru algoritmii de sortare eficienți funcția de comportare va fi O(*n logn*), iar cea mai dezavantajoasă comportare poate fi în cazul în care funcția de comportare este de O(n^2). Comportarea ideală pentru un algoritm de sortare este O(*n*).
Dintre cei mai cunoscuiti algoritmi de sortare putem enumera:

1. Heap sort
2. Sortare prin numarare
3. Sortare prin inserare
4. Metoda Shell
5. Metoda bulelor - **Bubble Sort**
6. Quicksort
7. Sortare prin selectie
8. Sortare prin interclasare
9. Bucket sort (bin sort)
10. Radix sort

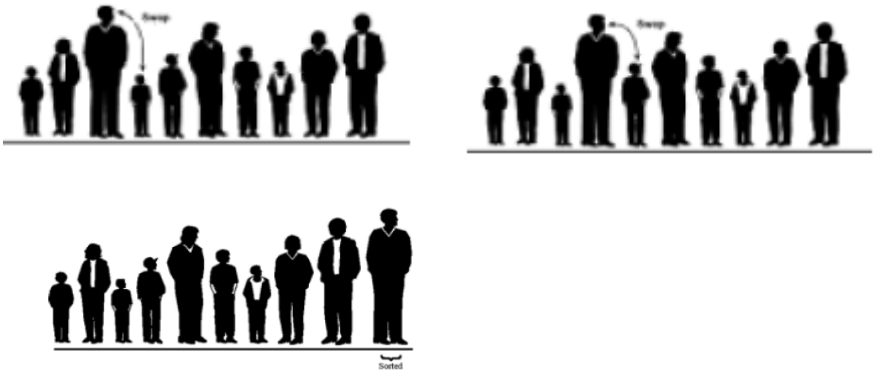
In cadrul acestei lucrari vor fi abordati cativa dintre acesti algoritmi, realizandu-se totodata si o scurta prezentare a principiului de sortare specific respectivului algoritm prezentat.

I. Algoritmul Bubble Sort

Regulile acestui algoritm sunt extrem de simple:

1. Sunt comparate 2 valori.
2. Dacă cea din stânga este mai mare decât cea din dreapta se inversează locul lor.
3. Se continuă algoritmul cu o pozitie la dreapta.

O reprezentare grafica a acestui algoritm, relativ la o mulțime de persone este exemplificată în imaginile următoare:



Se reia compararea de N-1 ori, în cazul în care şirul are N locații.

Aplicatie. Creați un proiect nou BubbleSort şi introduceți următoarele două clase:

```
class CreareSir
{     private double[] a;
      private int NrElmts;

public CreareSir(int max)
      { a = new double[max];
      NrElmts = 0;
      }

public void introduelement(double value)
      {     a[NrElmts] = value;
            NrElmts++;
      }

public void afiseaza()
      {for(int j=0; j<NrElmts; j++)
      System.out.print(a[j] + " ");
      System.out.println(""); }

public void bubbleSort()
      {     int out, in;
            for(out=NrElmts-1; out>1; out--)
            for(in=0; in<out; in++)
            if( a[in] > a[in+1] )
            inverseazaPozitii(in, in+1); }

private void inverseazaPozitii(int one, int two)
      {     double temp = a[one];
            a[one] = a[two];
            a[two] = temp; }
}

class BubbleSortProgram
{     public static void main(String[] args)
      {     int maxSize = 100;
            CreareSir vector = new
            CreareSir(maxSize);
            vector.introduelement(77);
```

```
            vector.introduelement(99);
            vector.introduelement(44);
            vector.introduelement(55);
            vector.introduelement(22);
            vector.introduelement(88);
            vector.introduelement(11);
            vector.introduelement(00);
            vector.introduelement(66);
            vector.introduelement(33);
        System.out.println("Sirul initial neordonat are urmatoarea forma");
        vector.afiseaza();
        vector.bubbleSort();
        System.out.println("Sirul ordonat are urmatoarea forma");
        vector.afiseaza(); }
}
```

Numărul total de comparații realizate în cadrul programului, pentru vectorul cu 10 componente este: 9+8+7+6+5+4+3+2+1. În general, daca N este numărul de elemente al vectorului support pentru componente, atunci numărul total de comparatii este:
(N–1) + (N–2) + (N–3) + ... + 1 = N*(N–1)/2

Aproximând destul de bine, putem considera că algoritmul face un total de $N^2/2$ comparații (acel -1 nu are mare importanță dacă N este destul de mare).

II. Sortarea prin selecție

Explicatiile legate de acest algorim, vor fi prezentate cu referire directa la realizarea sortarii prin selectie pentru a aranja elementele dintr-un sir (array) in ordine crescatoare. Algoritmul functioneaza astfel:

- In cadrul colectiei de date care urmeaza a fi sortate (initial, intreaga structura), gaseste cel mai mic articol;
- Plaseaza acest articol in prima pozitie (a structurii nesortate), inlocuind orice element care se afla deja acolo si muta elementul inlocuit la pozitia celui mai mic articol;
- Sectiunea nesortata contine acum un element mai putin (articolul tocmai considerat a fi cel mai mic si prin urmare, mutat);
- Continua plasarea celui mai mic element nesortat in prima pozitie pana cand toate datele au fost sortate.

Aplicatie: Creați un proiect nou *SelectionSort* şi implementați următoarele clase.

```
class CreareSir
{private double[] a;
 private int NrElmts;

public CreareSir(int max)
            { a = new double[max];
            NrElmts = 0;
            }
```

```
public void introduelement(double value)
            {     a[NrElmts] = value;
            NrElmts++;
            }

public void afiseaza()
{     for(int j=0; j<NrElmts; j++)
      System.out.print(a[j] + " ");
      System.out.println("");
            }

public void selectionSort()
            {
            int out, in, min;
            for(out=0; out< NrElmts -1; out++)
            {     min = out;
            for(in=out+1; in< NrElmts; in++)
                  if(a[in] < a[min] )
                  min = in;
                  inverseazaPozitii(out, min); }
            }

private void inverseazaPozitii(int one, int two)
            {     double temp = a[one];
            a[one] = a[two];
            a[two] = temp;
            }
            }

class SelectionSortProgram
{     public static void main(String[] args)
{     int maxSize = 100;

      CreareSir vector = new CreareSir(maxSize);
            vector.introduelement(77);
            vector.introduelement(99);
            vector.introduelement(44);
            vector.introduelement(55);
            vector.introduelement(22);
            vector.introduelement(88);
            vector.introduelement(11);
            vector.introduelement(00);
            vector.introduelement(66);
            vector.introduelement(33);
            System.out.println("Sirul initial neordonat are urmatoarea
      forma");
            vector.afiseaza();
            vector.selectionSort();
            System.out.println("Sirul ordonat are urmatoarea forma");
            vector.afiseaza();
            }
            }
```

Chiar dacă această metodă utilizează pentru comparare acelaşi număr de proceduri ca şi sortarea bubble sort: N*(N–1)/2, avantajul acestei metode rezidă din faptul că numărul de schimbări de locuri, între

termenii şirului este mult mai mic decât în cazul anterior. Pentru cazul în care N este foarte mare, această din urmă metodă este net superioară din punct de vedere al reducerii timpului de execuție.

III. Sortarea prin inserare

Sortarea prin insertie este un algoritm caracteristic procedurilor de sortare care opereaza cu tablouri cu un numar mic de elemente.
Principial algoritmul opereaza astfel:
- tabloul este divizat in doua parti - o parte sortata si o parte nesortata
- initial, partea sortata contine doar primul element al tabloului, iar partea nesortata contine restul tabloului.
- cu fiecare pas, algoritmul ia primul element din partea nesortata si il insereaza in locul potrivit al partii sortate. Cand partea nesortata nu mai are nici un element, algoritmul se opreste.

Aplicatie: Creați un nou proiect *InsertionSort* şi implementați următoarele clase:

```
class CreareSir
{private double[] a;
 private int NrElmts;

public CreareSir(int max)
 { a = new double[max];
   NrElmts = 0;
  }

 public void introduelement(double value)
  {   a[NrElmts] = value;
      NrElmts++;
  }

public void afiseaza()
  {   for(int j=0; j<NrElmts; j++)
      System.out.print(a[j] + " ");
      System.out.println("");
  }

public void insertionSort()
 {
   int in, out;
   for(out=1; out<NrElmts; out++)
   {
      double temp = a[out];
      in = out;
         while(in>0 && a[in-1] >= temp)
          {
             a[in] = a[in-1];
             --in;
          }
      a[in] = temp;
    }
  }
private void inverseazaPozitii(int one, int two)
   {  double temp = a[one];
      a[one] = a[two];
```

```
      a[two] = temp;
   }
}

class InsertionSortProgram
{     public static void main(String[] args)
{     int maxSize = 100;
      CreareSir vector = new CreareSir(maxSize);
      vector.introduelement(77);
      vector.introduelement(99);
      vector.introduelement(44);
      vector.introduelement(55);
      vector.introduelement(22);
      vector.introduelement(88);
      vector.introduelement(11);
      vector.introduelement(00);
      vector.introduelement(66);
      vector.introduelement(33);
      System.out.println("Sirul initial neordonat are urmatoarea forma");
      vector.afiseaza();
      vector.insertionSort();
      System.out.println("Sirul ordonat are urmatoarea forma");
      vector.afiseaza();
     }
     }
```

Aplicatie: Creați un proiect nou *SortarePersoane* şi implementați următoarele clase.

```
class Persoana
{     private String Prenume;
      private String Nume;
      private int varsta;

public Persoana(String last, String first, int a)
            { Prenume = last;
            Nume = first;
            varsta = a;
            }

public void afisarePersoana()
{System.out.print(" Pren. persoanei: " + Prenume);
System.out.print(" ,Numele persoanei: " + Nume);
System.out.println(", Varsta: " + varsta);
      }

public String preiaPrenume()
      { return Prenume; }
}

class CreareLista

{private Persoana[] a;
private int nElems;
public CreareLista(int max)
      {a = new Persoana[max];
```

```
      nElems = 0;
      }

public void inserarePersoana(String last, String first, int varsta)
      { a[nElems] = new Persoana(last, first, varsta);
        nElems++;
      }

public void afisare()
      {
      for(int j=0; j<nElems; j++)
      a[j].afisarePersoana();
      System.out.println("");
      }
      public void Sortare()
      {     int in, out;
      for(out=1; out<nElems; out++)
            {
            Persoana temp = a[out];
            in = out;
while(in>0 && a[in-1].preiaPrenume().compareTo(temp.preiaPrenume())>0)
{     a[in] = a[in-1];
                  --in;
                  }
            a[in] = temp;
            }
      }
}

class SortarePersoane
{     public static void main(String[] args)
      {
      int maxSize = 100;
      CreareLista arr = new
      CreareLista(maxSize);

arr.inserarePersoana("Ionescu", "Alin", 24);
arr.inserarePersoana("Ionescu", "Aladin", 59);
arr.inserarePersoana("Silvestru", "Vasile", 37);
arr.inserarePersoana("Anrdeescu", "Ivan", 37);
arr.inserarePersoana("Stamate", "Dan", 43);
arr.inserarePersoana("Popescu", "Ion", 21);
arr.inserarePersoana("Popescu", "Ionut", 29);
arr.inserarePersoana("Popescu", "Liviu", 72);
arr.inserarePersoana("Anghelescu", "Lucia", 22);
arr.inserarePersoana("Constantin", "Lavinia", 18);

System.out.println("Inainte de sortare:");
      arr.afisare();
      arr.Sortare();
      System.out.println("Dupa sortare:");
      arr.afisare();
      }
}
```

Probleme propuse

Pentru proiectul BubbleSort

1. Modificați programul *BubbleSort* astfel încât să puteți calcula media aritmetică a elementelor şirului.

Pentru proiectul SelectionSort

1. Modificați programul *SelectionSort* astfel încât să puteți calcula media aritmetica a primei jumatați a şirului ordonat, apoi media aritmetică a celei de-a doua jumătăți a şirului ordonat.

Pentru proiectul InsertionSort

1. Comentați acest program.
2. Ce rol are fiecare metodă introdusă?
3. Ce avantaje are această metodă de sortare comparativ cu celelalte?
4. Modificați programul astfel încât din folosirea ca argument a unei litere să puteți realiza sortarea prin una din cele 3 metode. Afişarea va fi diferită pentru fiecare din metodele de sortare utilizate.

Pentru proiectul SortarePersoane

1. Ce metodă de sortare foloseşte acest program?
2. Modificați programul utilizând toate celelalte metode de sortare.