1.Algoritmi de cautare

2.Algoritmi elementari de sortare

3.Liste

4.Algoritmi divide et impera

5.

**Algoritmi de cautare**

a). Cautare Secventiala (se prefera pentru tabele de dimensiuni mici)

static int cauta (String x) {

for (int i = 0; i < N; ++i)

if (x.equals(nume[i])) return telefon[i];

return 1;

}

se poate scrie ，si sub forma:

static int cauta (String x) {

int i = 0;

while (i < N && !x.equals(nume[i])) ++i;

if (i < N) return telefon[i];

else return 1;

}

O alta posibilitate este sa punem o santinela in capatul tabelului:

static int cauta (String x) {

int i = 0;

nume[N] = x; telefon[N] = 1;

while (! x.equals(nume[i])) ++i;

return tel[i];

}

Iata un program complet care utilizeaza o cautare liniara intr-o tabela

class Tabela {

final static int N = 6;

static String nume[] = new String[N+1];

static int telefon[] = new int[N+1];

static void initializare() {

nume[0] = "Paul"; telefon[0] = 2811;

nume[1] = "Robert"; telefon[1] = 4501;

nume[2] = "Laura"; telefon[2] = 2701;

nume[3] = "Ana"; telefon[3] = 2702;

nume[4] = "Tudor"; telefon[4] = 2805;

nume[5] = "Marius"; telefon[5] = 2806;

}

static int cauta(String x) {

for (int i = 0; i < N; ++i)

if (x.equals(nume[i]))

return tel[i];

return 1;

}

public static void main (String args[]) {

initializare();

if (args.length == 1)

System.out.println(cauta(args[0]));

}

}

b) Cautare binara

O alta tehnica de cautare in tabele este cautarea binara. Presupunem ca tabela de nume este sortata in ordine alfabetica. In loc de a cauta secvential, se compara cheia de cautare cu numele care se afla la mijlocul tabelei de nume. Daca aceasta este acelasi, se returneaza numarul de telefon din mijloc, altfel se reincepe cautarea in prima jumatate (sau a doua) daca numele cautat este mai mic (respective, mai mare) decat numele din mijlocul tabelei.

static int cautareBinara(String x) {

int i, s, d, cmp;

s = 0; d = N1;

do {

i = (s + d) / 2;

cmp = x.compareTo(nume[i]);

if (cmp == 0)

return telefon[i];

if (cmp < 0)

d = i 1;

else

s = i + 1;

} while (s <= d);

return 1;

}

Aceasta metode se numeste cautare prin interpolare. Timpul de cautare este O(log log N) , ceea ce inseamna cam 4 operatii pentru o tabela de 10.000 elemente si 5 operatii pentru 10^9 elemente in tabela. O implementare iterative a cautarii binare intr-un vector ordonat este:

int st, dr, m;

boolean gasit;

st=0;

dr=n-1;

gasit=false;

while((st < dr) && !gasit)

{

m=(st+dr)/2;

if(am]==x)

gasit=true;

else if(a[m] > x)

dr=m-1;

else st=m+1;

}

if(gasit) p=m; else p=-1;

Algoritmul poate fi descris, foarte elegant, recursiv.

**Algoritmi elementari de sortare**

a).Sortare prin selectie

class SortSelectie

{

final static int N = 10;

static int[] a = new int[N];

static void initializare()

{

int i;

for (i = 0; i < N; ++i)

a[i] = (int) (Math.random() \* 128);

}

static void afisare()

{

int i;

for (i = 0; i < N; ++i)

System.out.print (a[i] + " ");

System.out.println();

}

static void sortSelectie()

{

int min, t;

int i, j;

for (i = 0; i < N 1; ++i)

{

min = i;

for (j = i+1; j < N; ++j)

if (a[j] < a[min])

min = j;

t = a[min];

a[min] = a[i];

a[i] = t;

}

}

public static void main (String args[])

{

initializare();

afisare();

sortSelectie();

afisare(); } }

b).Sortare prin insertie

b.1)Insertie directa

protected void Page\_Load(object sender, EventArgs e)

{

InsertieDirecta();

}

private void InsertieDirecta()

{

initializare();

afisare();

string text = afisare();

Response.Write(text + "<br/>");

sortInsertie();

string text1 = afisare();

Response.Write(text1 + "<br/>");

}

static int N = 10;

static int[] a = new int[N];

public static void initializare()

{

Random rnd = new Random();

int i;

for (i = 0; i < N; ++i)

{

a[i] = rnd.Next(1, 30);

}

}

static string afisare()

{

StringBuilder sb = new StringBuilder();

sb.Append(" ");

int i;

for (i = 0; i < N; ++i)

{

sb.Append(a[i] + " ");

}

return sb.ToString();

}

static void sortInsertie()

{

int j, v;

for (int i = 1; i < N; ++i)

{

v = a[i]; j = i;

while (j > 0 && a[j - 1] > v)

{

a[j] = a[j - 1];

j--;

}

a[j] = v;

}

}

b.2) Insertie binara

#region InsertieBinara

private void InsertieBinara()

{

string sirb =afiseazaInsertieBinara();

Response.Write(sirb + "<br/>");

sorteaza();

string sortat= afiseazaInsertieBinara();

Response.Write(sortat + "<br/>");

}

static int[] b = { 3, 8, 5, 4, 9, 1, 6, 4 };

private string afiseazaInsertieBinara()

{

StringBuilder sb1 = new StringBuilder();

int j;

for (j = 0; j < b.Length; j++)

{

sb1.Append(b[j] + " ");

}

return sb1.ToString();

}

static int pozitiaCautBin(int p, int u, int x)

{

int i = u + 1;

while (p <= u)

{

i = (p + u) / 2;

if (x > b[i])

{

p = i + 1;

}

else if (x < b[i])

{

u = i - 1;

}

else { return i; }

}

return p;

}

static void deplasare(int k, int i)

{

if (i != k)

{

int x = b[k];

for (int j = k; j >= i + 1; j--)

{

b[j] = b[j - 1];

}

b[i] = x;

}

}

static void sorteaza()

{

int N = b.Length, i;

for (int k = 1; k <= N - 1; k++)

{

i = pozitiaCautBin(0, k - 1, a[k]);

deplasare(k, i);

}

}

#endregion

c).Sortare prin Interschimbare

#region SortarePrinInterschimare

private void SortareInterschimbare()

{

Response.Write("Insertie prin Interschimbare:");

initializare();

Response.Write(afisare() + "<br/>");

interschimbare(a);

Response.Write(afisare());

}

static int N = 10;

static int[] a = new int[N];

public static void initializare()

{

Random rnd = new Random();

int i;

for (i = 0; i < N; ++i)

{

a[i] = rnd.Next(1, 30);

}

}

static string afisare()

{

StringBuilder sb = new StringBuilder();

sb.Append(" ");

int i;

for (i = 0; i < N; ++i)

{

sb.Append(a[i] + " ");

}

return sb.ToString();

}

static void interschimbare(int[] a)

{

int x, i, n = a.Length;

bool schimb = true;

while (schimb)

{

schimb = false;

for (i = 0; i < n - 1; i++)

{

if (a[i] > a[i + 1])

{

x = a[i];

a[i] = a[i + 1];

a[i + 1] = x;

schimb = true;

}

}

}

} #endregion

d)Sortare prin micsorarea incrementului – Shell

#region SortareShell

private void SortareShell\_PrinMicsorareaIncrementului()

{

foreach (int j in h)

{

Response.Write(j + " ");

}

Response.Write("<br/>");

shell(h, 4);

foreach (int j in h)

{

Response.Write(j + " ");

}

}

static int[] h = { 9, 5, 3, 1 };

void shell(int[] a, int n)

{

int m, x, i, j, k;

for (m = 0; m < 4; m++)

{

k = m;

/\* sortare elemente aflate la distanta k in tablul a[] \*/

for (i = k; i < n; i++)

{

x = a[i];

for (j = i - k; (j >= 0) && (a[j] > x); j -= k)

a[j + k] = a[j];

a[j + k] = x;

}

}

}

#endregion

e)Sortare prin partitionare – quicksort

#region QuickSort

private void QuickSortCall()

{

int[] numbers = { 3, 8, 7, 5, 2, 1, 9, 6, 4 };

int len = numbers.Length;

StringBuilder sb = new StringBuilder();

Console.WriteLine("QuickSort By Recursive Method");

QuickSort\_Recursive(numbers, 0, len - 1);

for (int i = 0; i < 9; i++)

{

sb.Append(numbers[i] + " ");

}

string text = sb.ToString();

Response.Write(text);

}

static public int Partition(int[] numbers, int left, int right)

{

int pivot = numbers[left];

while (true)

{

while (numbers[left] < pivot)

left++;

while (numbers[right] > pivot)

right--;

if (left < right)

{

int temp = numbers[right];

numbers[right] = numbers[left];

numbers[left] = temp;

}

else

{

return right;

}

}

}

static public void QuickSort\_Recursive(int[] arr, int left, int right)

{

// For Recusrion

if (left < right)

{

int pivot = Partition(arr, left, right);

if (pivot > 1)

QuickSort\_Recursive(arr, left, pivot - 1);

if (pivot + 1 < right)

QuickSort\_Recursive(arr, pivot + 1, right);

}

}

#endregion

**Bactracking iterativ**

Structura general˘a a algoritmului este:

for(k=1;k*<*=n;k++) x[k]=gol; init，iarizarea vectorului solut，ie

k=1; pozit，ionare pe prima component˘a

while (k*>*0) cˆat timp exist˘a componente de analizat

if (k==n+1) dac˘a x este solut，ie

*{*afis(x); –k;*}* atunci: afi，sare solut，ie ，si pas stˆanga

else altfel:

*{*

if(x[k]*<*max[k]) dac˘a exist˘a elemente de ˆıncercat

if(posibil(1+x[k])) dac˘a 1+x[k] este valid˘a

++x[k++]; atunci m˘aresc ，si fac pas dreapta

else ++x[k]; altfel m˘aresc ，si r˘amˆan pe pozitie

else x[k–]=gol; altfel golesc ，si fac pas dreapta

*}*

Vectorul solut，ie *x* cont，ine indicii din mult，imile *A*1*,A*2*, ...,An*. Mai precis,

*x*[*k*] = *i* ˆınseamn˘a c˘a pe pozit，ia *k* din solut，ia *x* se afl˘a *ai* din *Ak*