Tema 9

Algorismes d'ordenació i recerca

1 Introducció

El propòsit general de les estructures de dades és el d'emmagatzemar informació. Ara, bé: imaginem que tenim un llistat de noms. Com serà més fàcil localitzar una persona? Si els noms estan ordenats o desordenats? La resposta és clara: si estan ordenats és molt més senzill.

Per tant, ara el problema radica en com ordenar eixa informació. Hi ha molts algorismes d'ordenació. Quin triar? Hi haurà alguns que seran més senzills d'implementar que altres però potser són menys eficients (tarden més temps). Segons els casos, convindrà un mètode o altre.

En este tema vorem uns pocs algorismes d'ordenació més o menys senzills i altres més avançats i complicats. Una vegada estan ja les dades ordenades, vorem algorismes per a buscar dades i per a inserir i esborrar noves dades de forma que el conjunt quede també ordenat.

Com podem imaginar, les dades que ordenarem les tindrem guardades en un vector. I les ordenarem de forma ascendent.

2 Algorismes d'ordenació

Imaginem que tenim en un full un llistat de 50 números que volem ordenar. Com ho farem? El més normal seria buscar el més menut, escriure'l en un altre full i tatxar-lo del primer full. Aleshores, amb els 49 números restants, faríem el mateix. I així fins que acabàrem la llista de números.

Però... a banda del cost (temps) de buscar el més menut 50 voltes, ¿com "tatxem" un element d'un vector? A més, el mètode anterior suposaria la utilització de 2 vectors: un desordenat i altre ordenat. Els mètodes d'ordenació que vorem fan servir el mateix vector per a fer la ordenació (no 2), ja que així reduïm el cost d'emmagatzematge del problema.

Hi ha algorismes que ens resolen estos problemes. Els més importants són:

- Algorismes d'ordenació
  - Senzills
    - **Intercanvi** (bombolla)
    - **Selecció directa** (recerca del menor)
    - **Inserció directa** (inserció ordenada)
  - Avançats
    - Shell Sort (increments decreixents)
    - Quick Sort (pivot)

Important:

Tots els algorismes que vorem dividixen el vector en 2 parts:

- ✓ una part desordenada (estarà a la dreta del vector)
- ✓ una part ordenada (estarà a l'esquerra del vector)

1) Inicialment partirem d'un vector tot desordenat (la part ordenada serà nul·la)

2) En un bucle anirem ordenant el vector. En cada iteració passarem un element de la part desordenada a l'ordenada.

3) Finalment, acabarem en el cas contrari: tot ordenat (i la part desordenada serà nul·la).

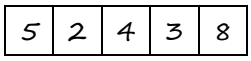
2.1 Ordenació per intercanvi (o mètode de la bombolla)

Es tracta d'anar ordenant els valors de 2 en 2, des del final fins al principi, de manera que el valor menor baixe fins la primera posició. Quan acabem eixe procés tindrem 1 element ordenat.

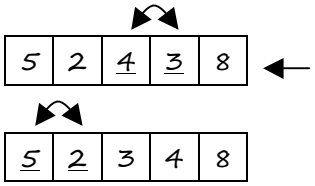
Per a tindre tot el vector ordenat hem de repetir el procés tantes vegades com elements (menys 1) tinga el vector.

Exemple gràfic:

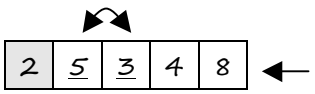
Vector original:



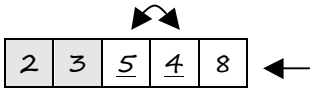
1r) Comencem comparant de 2 en 2 des del final cap al principi i intercanviem les parelles de números que no estiguen ordenades.



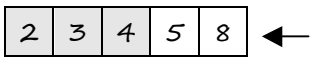
2n) Tornem a fer el procés



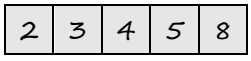
3r) Tornem a fer el procés



4t) Tornem a fer el procés, encara que ja estiga bé (són 5-1 voltes)



Vector final:



Algorisme

```
v[N]: float        // vector de N reals
ordenats: enter // quantitat d'elements que quedaran ordenats després de cada iteració
des: enter        // índex per a recórrer la part desordenada
aux: float        // variable auxiliar per a fer els intercanvis
PER ordenats=0 FINS N-1, INCR 1
        PER des=N-1 FINS ordenats+1, INCR -1
                SI v[des]<v[des-1]   ALESHORES
                        // intercanviem la parella desordenada
                        aux = v[des]
                        v[des] = v[des-1]
                        v[des-1] = aux
                FI_SI
        FI_PER
FI_PER
```

Nota: es podria millorar l'algorisme, ja que si en una de les passades no es fa cap intercanvi, no caldrà que tornem a fer més passades. Això es pot resoldre amb una variable booleana que faça d'interruptor.

2.2 Ordenació per selecció directa (o recerca del menor)

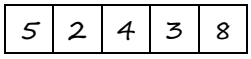
Es tracta de buscar el menor de tots els elements i substituir-lo pel primer.

Després, buscar el menor des del 2n element fins l'últim i substituir-lo pel 2n.

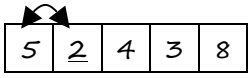
I així, successivament fins estar tot el vector ordenat.

Exemple gràfic:

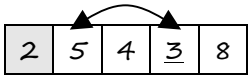
Vector original:



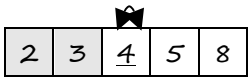
1r) Busquem el menor de tots i el substituïm pel primer



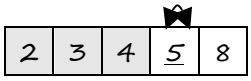
2n) Busquem el menor dels restants i el substituïm pel segon



3r) Busquem el menor dels restants i el substituïm pel tercer



4t) Busquem el menor dels restants i els substituïm pel quart



Vector final:

Algorisme:

```
v[N]: float        // vector de N reals
PER i=0 FINS N-2, INCR 1
        // Busquem el menor des de v[i] fins v[N-1]
        pos_min = i
        PER j = i+1 FINS N-1, INCR 1
                SI v[j]<v[pos_min] ALESHORES
                        pos_min = j
                FI_SI
        FI_PER
        // Intercanviem el mínim per v[i]
        aux = v[i]
        v[i] = v[pos_min]
        v[pos_min] = aux
FI_PER
```

2.3 Ordenació per inserció directa (o ordenada)

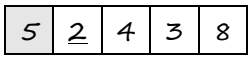
És el mètode que utilitzen els jugadors de cartes per a ordenar-se-les. La 1a carta està ja ordenada. Si la 2a és menor, la posem davant. Ja en tenim 2 ordenades. Agafem la 3a i la posarem al lloc que li toque en la part de cartes ordenades, desplaçant les cartes que calguen cap a la dreta. I així successivament fins estar totes ordenades.

Exemple gràfic

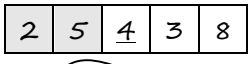
Vector original:

Partim del fet que el 1r element "ja està ordenat".

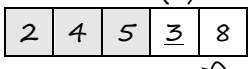
1r) Posem el 2n on toca: desplacem el 5 a la dreta i fiquem el 2:



2n) Posem el 3r on toca: desplacem el 5 a la dreta i fiquem el 4:



3r) Posem el 4t on toca: desplacem el 5 i 4 a la dreta i fiquem el 3:

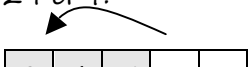


4t) Posem el 5é on toca: no desplacem perquè el 8 va al final de la part ordenada (es queda on està).

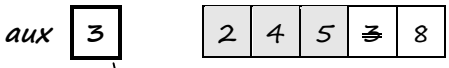
Vector final:

(*) Detallem el procés de posar un element on toca: el 3 entre el 2 i el 4.

Pretenem posar el 3 al lloc que li toque en la part ordenada (entre el 2 i el 4). Vegem els passos que cal fer:



a) Guardem el 3 en una variable auxiliar:



b) Recorrem la part ordenada des del final mentre desplacem cap a la dreta aquells elements que siguen majors que 3:

- Com el 5 és major que el 3, el copiem a la dreta:

- Com el 4 és major que el 3, el copiem a la dreta:

- Com el 2 NO és major que el 3, eixim del bucle.

c) Copiem el 3 (que tenim guardat en aux) on estava l'últim element desplaçat (el 4).

Algorisme

```
PER i=1 FINS N-1, INCR 1
        aux=v[i];
        j=i-1
        MENTRE ((j>=0) && (aux < v[j]))
                v[j+1]=v[j];
                j--;
        FI_MENTRE
        v[j+1] = aux;
FI_PER
```

Exercicis

1. Fes la funció *ordenarBombolla* que implemente l'algorisme de la bombolla. És a dir, se li passa com a paràmetre un vector de reals i l'ha de tornar ordenat.

2. Fes el programa *ordenar* que demane 10 números, els guarde en un vector, els ordene utilitzant la funció anterior i, finalment, els mostre ordenats.

3. Fes la funció *ordenarSeleccioDirecta* que implemente l'algorisme corresponent.

4. Fes la funció *ordenarInsercioDirecta* que implemente l'algorisme corresponent.

5. Modifica el programa ordenar perquè ordene el vector amb els 3 mètodes vistos.

3 Algorismes de recerca

Els algorismes de recerca s'utilitzen per a saber si un valor determinat es troba entre els elements del vector, i en quina posició. Hi ha bàsicament 2 formes de buscar:

- **Recerca seqüencial o lineal**: buscar de principi a fi.
- **Recerca binària**: si el vector està ordenat, es busca "com en un diccionari".

3.1 Recerca seqüencial o lineal

És la tècnica més senzilla però la menys eficient.

Comencem comparant l'element a buscar en la 1a casella del vector. Si és eixe element, estem d'enhorabona. Si no, compararem l'element a buscar amb la 2a casella del vector. I així successivament fins trobar-lo.

```
trobat = -1   // Val -1 mentre no es troba; o la posició del vector on està.
i = 0         // Índex del vector.
// Recorrem el vector mentre no arribem al final ni trobem l'element.
MENTRE (i<N) AND (trobat == -1)
        SI v[i] == element ALESHORES
                trobat = i      // Guardem la posició on està l'element.
        FI_SI
        i++
FI MENTRE
// Podem haver eixit del bucle per 2 motius: fi de bucle o element trobat.
SI trobat == -1 ALESHORES
        escriu("L'element ", element, " no està al vector)
SI_NO
        escriu("L'element", element, " està a la posició ", trobat)
FI_SI
```

3.2 Recerca binària

Podem utilitzar esta tècnica **si el vector està ordenat**.

L'algorisme és un poc més complex que el de recerca lineal però és més ràpid.

Com busquem una paraula en un diccionari? El primer pas serà obrir-lo per la meitat. Si la paraula que busquem és menor, ens quedarem amb la meitat esquerra i la tornarem a obrir per la meitat. Però si és major, ens quedem amb la part dreta i l'obrim per la meitat. I així successivament fins trobar la paraula que busquem.

Eixa forma de buscar una paraula és una recerca binària. És a dir, consistix a comparar l'element a buscar amb l'element central del vector, on poden passar 3 casos:

a) Que siga igual → ja hem trobat l'element i acaba l'algorisme.
b) Que siga major al que busquem → buscarem en la part esquerra.
c) Que siga menor al que busquem → buscarem en la part dreta.

```
inferior = 0          // índex inferior de la part a buscar a cada iteració
superior = N-1        // índex superior de la part a buscar a cada iteració
mig = (inferior + superior) / 2      // índex de l'element a comparar
trobat = -1  // Val -1 mentre no es troba; o la posició del vector on està.
// Fem el vector més menut mentre pugam i mentre no trobem l'element.
MENTRE (inferior<superior) AND (trobat == -1)
        SI v[mig] == element ALESHORES
                trobat = mig // Guardem la posició on està l'element.
        SI_NO
                SI v[mig]>element ALESHORES
                        superior = mig – 1          // Meitat esquerra
                SI_NO
                        inferior = mig + 1          // Meitat dreta
                FI_SI
        FI_SI
        mig = (inferior + superior) / 2
FI MENTRE
// Podem haver eixit del bucle per 2 motius: fi de bucle o element trobat.
SI trobat == -1 ALESHORES
        escriu("L'element ", element, " no està al vector)
SI_NO
        escriu("L'element", element, " està a la posició ", trobat)
FI_SI
```

Exercicis

6. Funció *recercaLineal* que li passes un vector d'enters i un enter i retorna la posició del vector on està l'element. Si no està, retorna un -1.

7. Funció *recercaBinariaIterativa*, que faça una recerca binària de forma iterativa.

8. Funció *recercaBinariaRecursiva*, que faça una recerca binària de forma recursiva.

9. Programa *recerca* que òmpliga un vector de 10 enters amb números aleatoris de l'1 al 15. El programa ha de demanar un número per teclat i dir si no està en el vector o si sí que està i en quina posició. Cal fer-ho amb les 3 funcions anteriors.

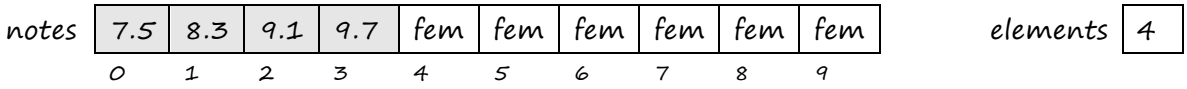
   Per a omplir el vector de números aleatoris, utilitza este codi:

```
import java.io.*;
import java.util.Random;

public class Main {

    public static void main(String[] args) {
        int v[] = new int[10];    // Creem el vector de 10 elements
        vectorAleatori(v, 15);    // L'omplim amb valors aleatoris de l'1 al 15
        imprimir(v);              // Mostrem el vector per comprovar resultats
        // Demanar element
        // Dir si està (i on) per recerca lineal
        // Ordenar vector (useu funció ja feta anteriorment)
        // Dir si està (i on) per recerca binària iterativa
        // Dir si està (i on) per recerca binària recursiva
    }

    public static void vectorAleatori(int v[], int maxim) {
        Random rnd = new Random();
        for (int i = 0; i < v.length; i++) {
                v[i] = (int)(rnd.nextDouble() * maxim) + 1;
        }
    }
}
```

4 Algorismes d'inserció i supressió ordenada

Si volem permetre que en el vector es puguen esborrar i inserir nous elements:

- ✓ Cal una variable que diga quants elements significatius té el vector
- ✓ Els elements estaran tots a la part esquerra del vector.




```
float notes[]=new float[10]
notes[0]=7.5; notes[1]=8.3; notes[2]=9.1; notes[3]=9.7;
int elements;
elements = 4;
```

4.1 Algorisme de supressió ordenada

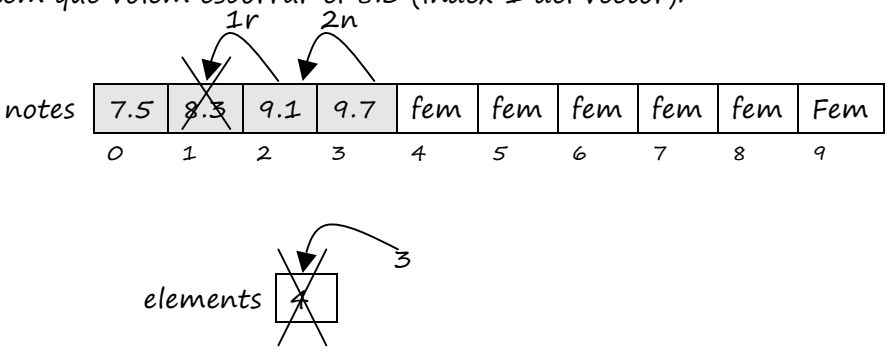
Suposem que volem esborrar un element d'un vector. Primer haurem de buscar en quina posició està l'element a esborrar i després procedir a l'esborrat.

Algorisme

- ✓ Localitzar l'element a esborrar (algorismes de recerca)
- ✓ Desplaçar els elements de la dreta una posició cap a l'esquerra.
- ✓ Decrementar la variable *elements* en 1 unitat

Exemple

Suposem que volem esborrar el 8.3 (índex 1 del vector):



4.2 Algorisme d'inserció ordenada

Per a inserir un nou element i que el vector continue estant ordenat, el més fàcil seria inserir l'element al final (primera posició lliure) i tornar a aplicar un mecanisme d'ordenació però este mètode és poc eficient. Per tant, li farem espai al nou element i l'inserirem.

Algorisme

- ✓ Si la taula ja esta plena, no podem continuar (no es pot inserir)
- ✓ Si la taula està buida, col·loquem l'element a la primera posició del vector.
- ✓ Si ja té elements (i ordenats, clar):
  - Aplicarem la tècnica de l'algorisme d'ordenació per inserció directa per a posar l'element on toca.
  - Incrementarem la variable *elements* en 1 unitat

Exemple

Veieu el detall del procés de posar un element on toca vist en l'algorisme d'ordenació per inserció directa.

Exercicis

10. Fes la funció *inserirEnVectorOrdenat*, que li passes el vector (d'enters), la quantitat d'elements significatius que té i l'element a inserir. El programa deu modificar el vector introduint el nou element al lloc que li pertoca i ha de retornar la nova quantitat d'elements significatius. Cal controlar que l'element càpiga al vector.

11. Fes la funció *esborrarDeVectorOrdenat*, que li passes el vector (d'enters), la quantitat d'elements significatius que té i l'índex de la posició a esborrar. El programa deu modificar el vector eliminant l'element corresponent i ha de retornar la nova quantitat d'elements significatius. Cal controlar que l'índex de l'element a esborrar continga un valor significatiu.