TEMA 7.
ALGORITMOS DE BÚSQUEDA, ORDENACIÓN Y MEZCLA

Grado en Ingeniería en Tecnologías Industriales

Programación

M.P. Sesmero, P. de Toledo, F.J. Ordoñez, J. Gómez-Romero, J.A. Iglesias, J.L. Mira

Universidad Carlos III de Madrid
www.uc3m.es

CONTENIDOS

Búsqueda, ordenación y mezcla de vectores

- Algoritmos para buscar y ordenar una lista de valores
  - La lista se representa como un vector

- **Búsqueda:**
  - En listas sin ordenar
  - En listas ordenadas

- **Ordenación:**
  - Algoritmo de la burbuja
  - Algoritmo de inserción
  - Algoritmo de selección

- **Mezcla:**
  - Mezclar dos listas ordenadas en una única lista

7.1. BÚSQUEDA

M.P. Sesmero, P. de Toledo, F.J. Ordoñez, J. Gómez-Romero, J.A. Iglesias, J.L. Mira

Algoritmos de búsqueda

- Búsqueda
  - encontrar qué posición ocupa un determinado valor en una lista
- Dos opciones:
  - Búsqueda en listas sin ordenar
  - Búsqueda en listas ordenadas
    - Más eficiente, aunque previamente hay que ordenar
- Listas sin ordenar
  - Búsqueda secuencial simple o búsqueda lineal
    - Examinar cada elemento empezando por el primero hasta que se encuentra el valor buscado o hasta que llega el final de la lista
- Listas ordenadas
  - Búsqueda secuencial modificada
  - Búsqueda binaria

Búsqueda secuencial (lista sin ordenar)

```
//leemos el dato a buscar
scanf("%i",&dato);

// busqueda secuencial
    i = 0 ;
    encontrado = 0 ; // marcador: toma valor cierto(1) o falso(0)
    while((i<N)&&(!encontrado)){
      if (a[i]==dato)
        encontrado = 1 ;
      else
        i++;
    }
 // escribimos los resultados
 if (encontrado){ //a[i]==dato
    printf ("%s", "el dato buscado esta en la posición");
    printf ("%d", i+1);
 }
 else {

    printf ("%s", "el dato buscado no esta en el vector");
 }
```

Búsqueda en listas ordenadas

- Búsqueda en listas ordenadas
  - Es más eficiente (pensar en diccionario)
  - Previamente hay que ordenarlas

- Veremos dos algoritmos
  - Búsqueda secuencial modificada
    - La búsqueda termina cuando se encuentra el elemento o cuando se pasa de la posición de la lista donde ese elemento debería estar
  - Búsqueda binaria
    - Examinar el elemento central de la lista. Si no es el buscado determinar si el elemento debería estar en la mitad superior o inferior de la lista
    - Repetir con la sublista apropiada hasta que se localiza el elemento o la sublista está vacía.

Búsqueda secuencial (lista ordenada)

```
//leemos el dato a buscar
scanf("%i",&dato);

// busqueda secuencial
    i = 0 ;
    encontrado = 0 ; // marcador: toma valor cierto(1) o falso(0)
    fin=0 ;          // marcador: toma valor cierto(1) o falso(0)
    while((!fin)&&(!encontrado)){
      if (a[i]==dato)
         encontrado = 1 ;
      else
        if ( (a[i]>dato) || (i==(N-1)) )
          // Si se pasa la posición en la que debería estar el dato o
          // se ha recorrido todo el vector termina la búsqueda.
          fin = 1 ;
        else
          i++;
    }
 // escribimos los resultados
 if (encontrado){ //a[i]==dato
    printf ("%s", "el dato buscado esta en la posición");
    printf ("%d", i+1);
 }
 else {
    printf ("%s", "el dato buscado no esta en el vector");
 }
```

Búsqueda binaria

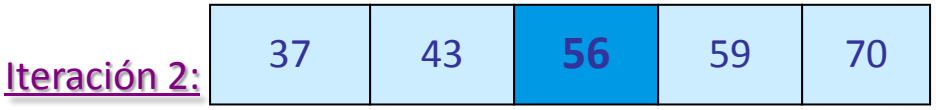
dato =37

¿7?

Búsqueda binaria

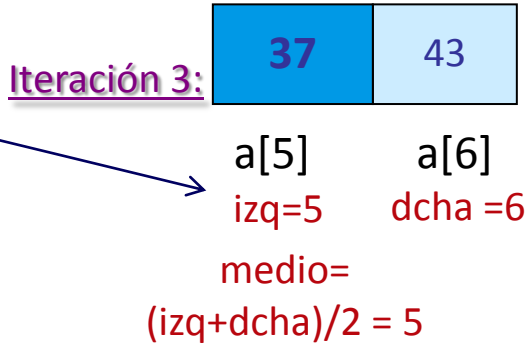
dato =37

El valor central (a[medio] = 28) es menor que el número buscado, se toma la mitad superior de la lista para la siguiente iteración.



El nuevo a[medio] (56) es mayor que el valor buscado, por lo que se toma la sublista inferior

a[medio] = dato buscado (37), búsqueda termina



Búsqueda binaria

dato =7

Iteración 1:

a[0]
izq=0

a[**4**]
medio =
(izq+dcha)/2=4

a[9]
dcha = 9

1 - El valor central a[medio] *(28),* es mayor que el número buscado, se toma la mitad inferior de la lista para la siguiente iteración

Iteración 2:

a[0]
izq=0

a[1]
medio =
(izq+dcha)/2 = 1

a[3]
dcha =3

2- El nuevo a[medio] (*11*), es mayor que el valor buscado, por lo que se toma la sublista inferior

Iteración 3:

a[0]
izq=0 dcha =0
medio = (izq+dcha)/2 = 0

3 -El nuevo a[medio] (5) es menor que el valor buscado, por lo que se toma la sublista superior

dcha =-1
izda = 0

4 – izq > dcha, la búsqueda termina

Búsqueda binaria

```
# define N 10  //Tamaño del vector

int a[N]; //vector
//índices para marcar los extremos izdo y dcho de la zona de
//búsqueda considerada
int izq, dcha, medio, dato;//valor buscado
int encontrado; //toma valor cierto(1) o falso(0)
int posicion=-1; //Posición que ocupa el dato en el vector.
                 //Valor por defecto -1;
```

Búsqueda binaria

```
//leemos el dato a buscar
scanf("%i",&dato);

   //inicializamos los valores
   izq = 0;  //Límite inferior/izqda (actual) de búsqueda.
   dcha = N-1;  //Límite superior/drcha (actual) de búsqueda.
   encontrado = 0;
  medio=(izq+dcha)/2;
   //búsqueda. Se va comparando a[m] con el dato buscado.

   while ((izd<=dcha) && (!encontrado)){
        if (a[medio] == dato){
           encontrado=1;
           posicion=medio;
          }
        else{
           if (a[medio] < dato)
          //Si a[medio] es menor, el nuevo extremo izdo es m+1
             izq = medio + 1;
           else
           //Si a[medio] es mayor, el nuevo extremo dcho es m-1
             dcha = medio - 1;
          medio= (izq+dcha)/2;
        }
    }
```

Búsqueda binaria

```
// escribimos los resultados

if (encontrado)
 {
   printf ("Valor %d encontrado en índice %d" ,dato, posicion);
 }
 else
 {
   printf (El dato %d no esta en el vector", dato);
 }
```

7.2. ORDENACIÓN

M.P. Sesmero, P. de Toledo, F.J. Ordoñez, J. Gómez-Romero, J.A. Iglesias, J.L. Mira

Algoritmos de ordenación

- Algoritmos de ordenación básicos (http://www.sorting-algorithms.com/)
  - **Intercambio (Burbuja):**
    - Consiste en comparar pares de elementos adyacentes e **intercambiarlos** entre sí hasta que estén todos ordenados
  - **Selección directa:**
    - Consiste en **seleccionar** el elemento más pequeño del vector y ponerlo en primera posición; luego, entre los restantes, se busca el elemento más pequeño y se coloca en segundo lugar, y así sucesivamente hasta colocar el último elemento
  - **Inserción directa:**
    - Consiste es tener una sublista ordenada de elementos del vector e ir **insertando** el resto en el lugar adecuado para que la sublista no pierda el orden. La sublista ordenada se va haciendo cada vez mayor, de modo que al final la lista entera queda ordenada.

Algoritmo de intercambio o de la burbuja. Ejemplo.

Ejemplo : Para **cuatro** elementos, ordenar **de menor a mayor**

- Primera iteración:

  Comparar cada elemento (excepto el último) con el que tiene detrás
  - Si están desordenados, cambiarlos de sitio
    - If m(j) > m (j+1), cambiarlos de sitio
  - De esta manera se coloca el mayor al final
  - El último elemento está ordenado, en su posición definitiva
- Segunda iteración:

  Comparar cada elemento (excepto los dos últimos) con el que tiene a la derecha
  - Si están desordenados, cambiarlos de sitio
  - De esta manera se coloca el segundo mayor en la penúltima posición
  - Los dos últimos elementos están ordenados, en su posición definitiva
- Tercera :

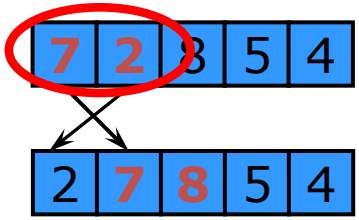
  Comparar cada elemento (excepto los tres últimos) con el que tiene a la derecha
  - Continuar hasta que toda la lista esté ordenada

Algoritmo de intercambio o de la burbuja.

Ordenar un vector a[] de **cinco** elementos (N = 5) de **menor a mayor**

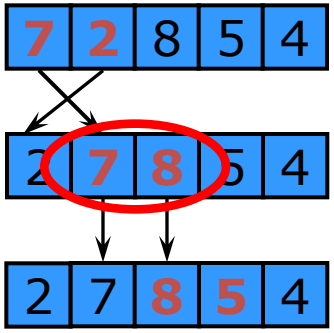
Algoritmo de intercambio o de la burbuja.

Ordenar un vector a[] de **cinco** elementos (N = 5) de **menor a mayor**



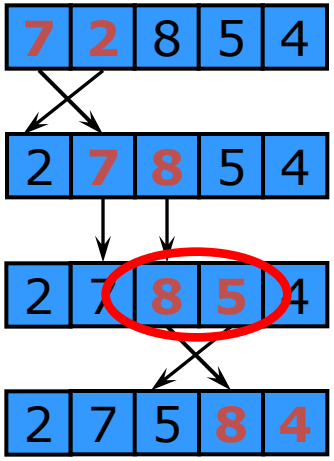
Algoritmo de intercambio o de la burbuja.

Ordenar un vector a[] de **cinco** elementos (N = 5) de **menor a mayor**



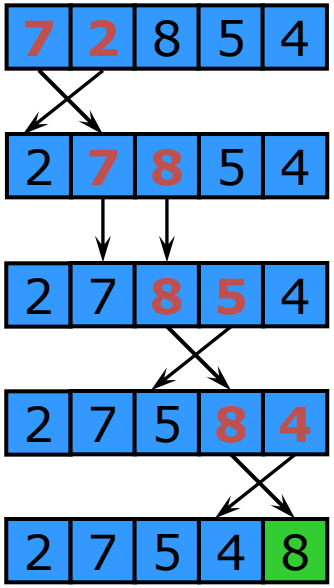
Algoritmo de intercambio o de la burbuja.

Ordenar un vector a[] de **cinco** elementos (N = 5) de **menor a mayor**



Algoritmo de intercambio o de la burbuja.

Ordenar un vector a[] de **cinco** elementos (N = 5) de **menor a mayor**

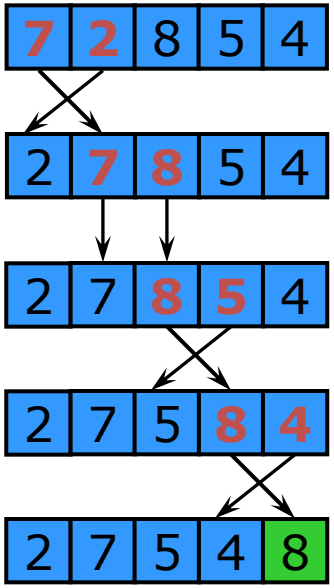


1ª iteración:
(i=0)
queda colocado el mayor

4 comparaciones
N-(i+1) comparaciones

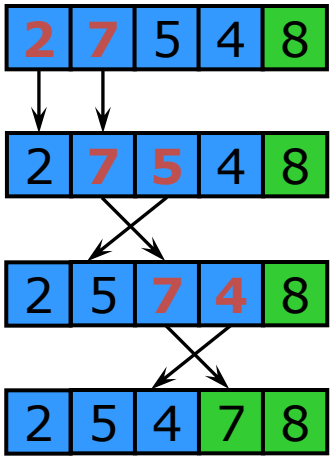
Algoritmo de intercambio o de la burbuja.

Ordenar un vector a[] de **cinco** elementos (N = 5) de **menor a mayor**



1ª iteración:
(i=0)
queda colocado el mayor

4 comparaciones
N-(i+1) comparaciones

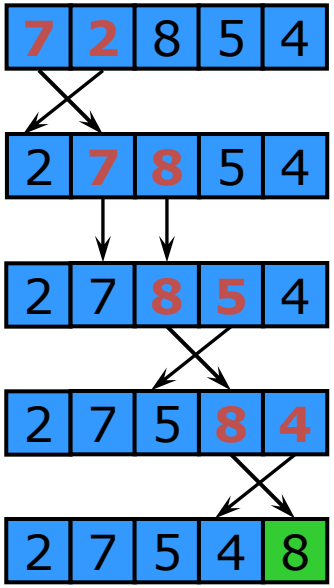


2ª iteración:
(i=1)
quedan colocados los dos mayores

3 comparaciones
N-(i+1) comparaciones

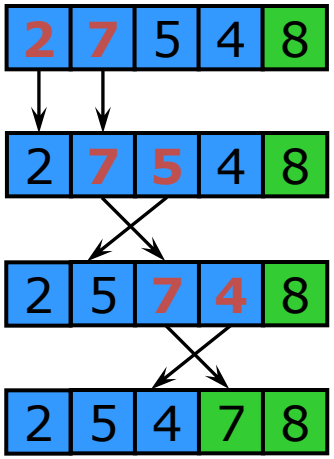
Algoritmo de intercambio o de la burbuja.

Ordenar un vector a[] de **cinco** elementos (N = 5) de **menor a mayor**



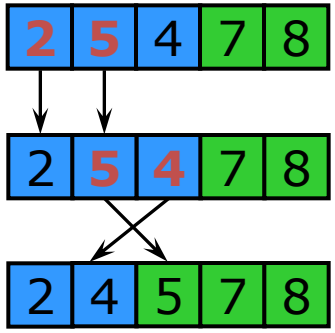
1ª iteración:
(i=0)
queda colocado el mayor

4 comparaciones
N-(i+1) comparaciones



2ª iteración:
(i=1)
quedan colocados los dos mayores

3 comparaciones
N-(i+1) comparaciones

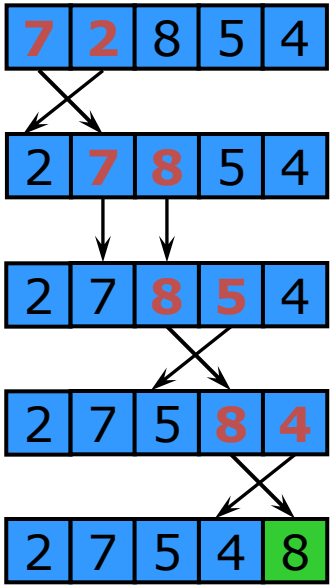


3ª iteración:
(i=2)
quedan colocados los tres mayores

2 comparaciones
N-(i+1) comparaciones

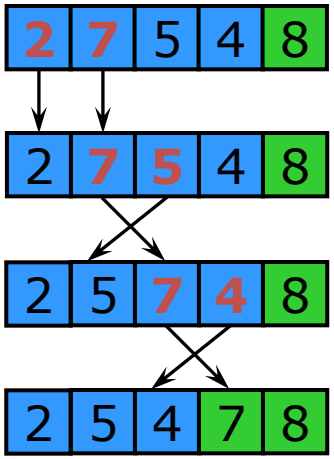
Algoritmo de intercambio o de la burbuja.

Ordenar un vector a[] de **cinco** elementos (N = 5) de **menor a mayor**



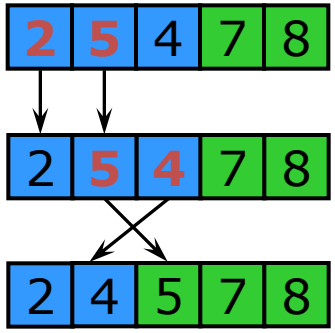
1ª iteración:
(i=1)
queda colocado el mayor en la última posición del array

4 comparaciones
(N-i) comparaciones



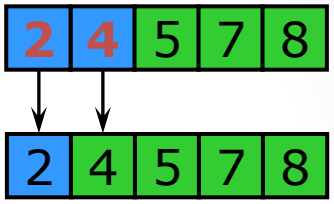
2ª iteración:
(i=2)
quedan colocados los dos mayores al final del array

3 comparaciones
(N-i) comparaciones



3ª iteración:
(i=3)
quedan colocados los tres mayores

2 comparaciones
(N-i) comparaciones



(fin)

4ª iteración:
(i=4)
quedan colocados todos

1 comparación
(N-i) comparaciones

Necesito 4 iteraciones = N-1 iteraciones

El número de comparaciones necesarias empieza en (N-1) y disminuye en cada iteración

Algoritmo de intercambio o de la burbuja

Generalizando, para un vector de N elementos:

- Son necesarias N-1 iteraciones, pues:
  - En cada iteración se coloca un elemento en su posición correcta.
  - Por tanto, en la iteración N-1 (la última) quedarán ordenados N-1 elementos.
  - Y el que queda estará necesariamente ordenado

- Son necesarias (N-i) comparaciones en la iteración i, pues:
  - En la primera iteración (i=1) hay que comparar cada elemento con el que tiene detrás. Por tanto, el último no es necesario compararlo (N-1 comparaciones).
  - En cada iteración se ordena un elemento más, por lo que se va reduciendo el número de comparaciones necesarias.
    - En la iteración 2 *(i=2),* hay que hacer N-2 comparaciones.
    - En la iteración 3 *(i=3)*, N-3 comparaciones, es decir: (*N-i)*
    - …

Algoritmo de intercambio o de la burbuja

```
// durante N-1 iteraciones
for(i=1; i<=N-1; i++)
{
   // Coge los N-i primeros elementos y los compara
   //  cada uno de ellos con el inmediatamente posterior.
   for(j=0; j < N-i; j++)
    {
   // si están desordenados se intercambian
   // usando una variable auxiliar.
      if(a[j]>a[j+1])
       {
          aux=a[j];
          a[j]=a[j+1];
          a[j+1]=aux;
        }
    }
}
```

Algoritmo de intercambio o de la burbuja

¿Para ordenar de **mayor a menor?** cambio criterio de comparación:

```
// durante N-1 iteraciones
for(i=1; i<=N-1; i++)
{
   for(j=0; j < N; j++)
    {
       if(a[j]<a[j+1])
        {
            aux=a[j];
            a[j]=a[j+1];
            a[j+1]=aux;
         }
     }
}
```

Algoritmo de intercambio o de la burbuja

- Posible mejora del algoritmo:
  - Detectar cuando no se ha efectuado ninguna modificación en una iteración
  - Eso significa que ya está ordenado y no hace falta seguir

Algoritmo de selección directa

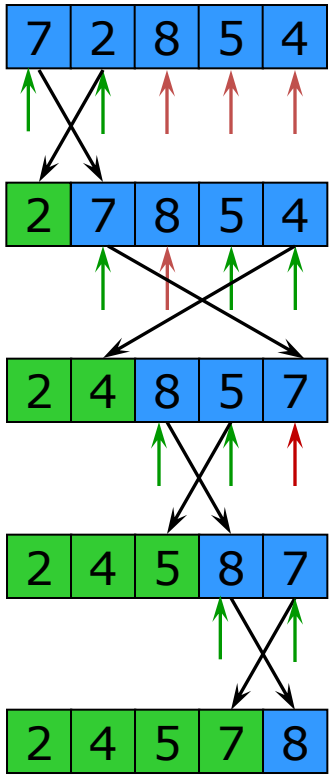
- Algoritmo: (http://www.youtube.com/watch?v=boOwArDShLU)
  - Seleccionar el elemento más pequeño del vector y ponerlo en primera posición; luego, entre los restantes, buscar el elemento más pequeño y colocarlo en segundo lugar, y así sucesivamente hasta colocar el último elemento
  - Tenemos una sublista ordenada que va aumentando

- Dado un vector de longitud N,
  - Buscar los elementos 0 hasta N-1 y elegir el menor
    - Cambiarlo con el elemento en posición 0
  - Buscar los elementos 1 hasta N-1 y elegir el menor
    - Cambiarlo con el elemento en posición 1
  - Buscar los elementos 2 hasta N-1 y elegir el menor
    - Cambiarlo con el elemento en posición 2
  - Buscar los elementos 3 hasta N-1 y elegir el menor
    - Cambiarlo con el elemento en posición 3
  - Continuar de esta forma hasta que no quede nada que buscar

Algoritmo de selección directa

Ejemplo con N=5

i indica el primer elemento de la lista desordenada



j desde 1 hasta N-1

j desde 2 hasta N-1

j desde 3 hasta N-1

j desde 4 hasta N-1

j indica los elementos que voy comparando

Primera iteración
Primer elemento de la lista desordenada ocupa la posición i=0; jmin=1;

Segunda iteración,
Primer elemento de la lista desordenada ocupa la posición i=1; jmin=4;

Tercera iteración,
Primer elemento de la lista desordenada ocupa la posición i=2; jmin=3;

Cuarta iteración i=3; jmin=4

FIN

Necesito 4 iteraciones
(N-1 iteraciones)

Algoritmo de selección directa

```
for(i=0;i<N-1;i++)
{ //en cada iteración inicializamos el mínimo al primero
  desordenado
  min = a[i];
  jmin = i;
  for(j=i+1; j<N; j++)
  {    //comparo cada elemento con el siguiente y
       //voy quedandome con el menor
       if (a[j] < min)
       {
              min = a[j];
              jmin = j;
       }
  }
  //pongo el primero desordenado a[i] donde estaba el menor
  a[jmin] = a[i];
  //y el menor min donde estaba el primero desordenado
  a[i] = min;
}
```

`min` almacena el menor valor de la iteración

`jmin` almacena el índice del menor valor de la iteración

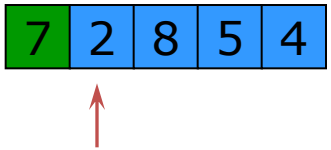
Algoritmo de inserción directa

- Se basa en tomar una sublista ordenada de elementos de la lista e ir insertando el resto en el lugar adecuado de la sublista
- La sublista ordenada se va haciendo cada vez mayor, de modo que al final la lista entera queda ordenada.
- La lista inicial es un único elemento, el primero
- Veremos la codificación del algoritmo usando búsqueda secuencial para colocar cada elemento
  - se podría optimizar el algoritmo usando búsqueda binaria
- Ejemplo: http://www.youtube.com/watch?v=gTxFxgvZmQs

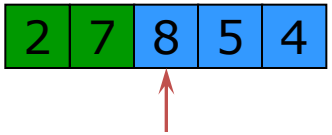
Algoritmo de inserción directa

Ejemplo con N=5

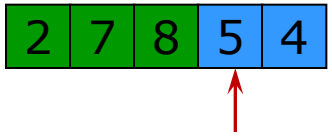
i indica el elemento a colocar



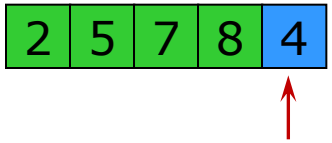
Primera iteración (i=1)
La sublista ordenada contiene un elemento
Insertamos el elemento lista[1]=2 en la posición correcta.



Segunda iteración (i=2)
La sublista ordenada contiene dos elementos
Insertamos el elemento lista[2]=8 en la posición correcta.



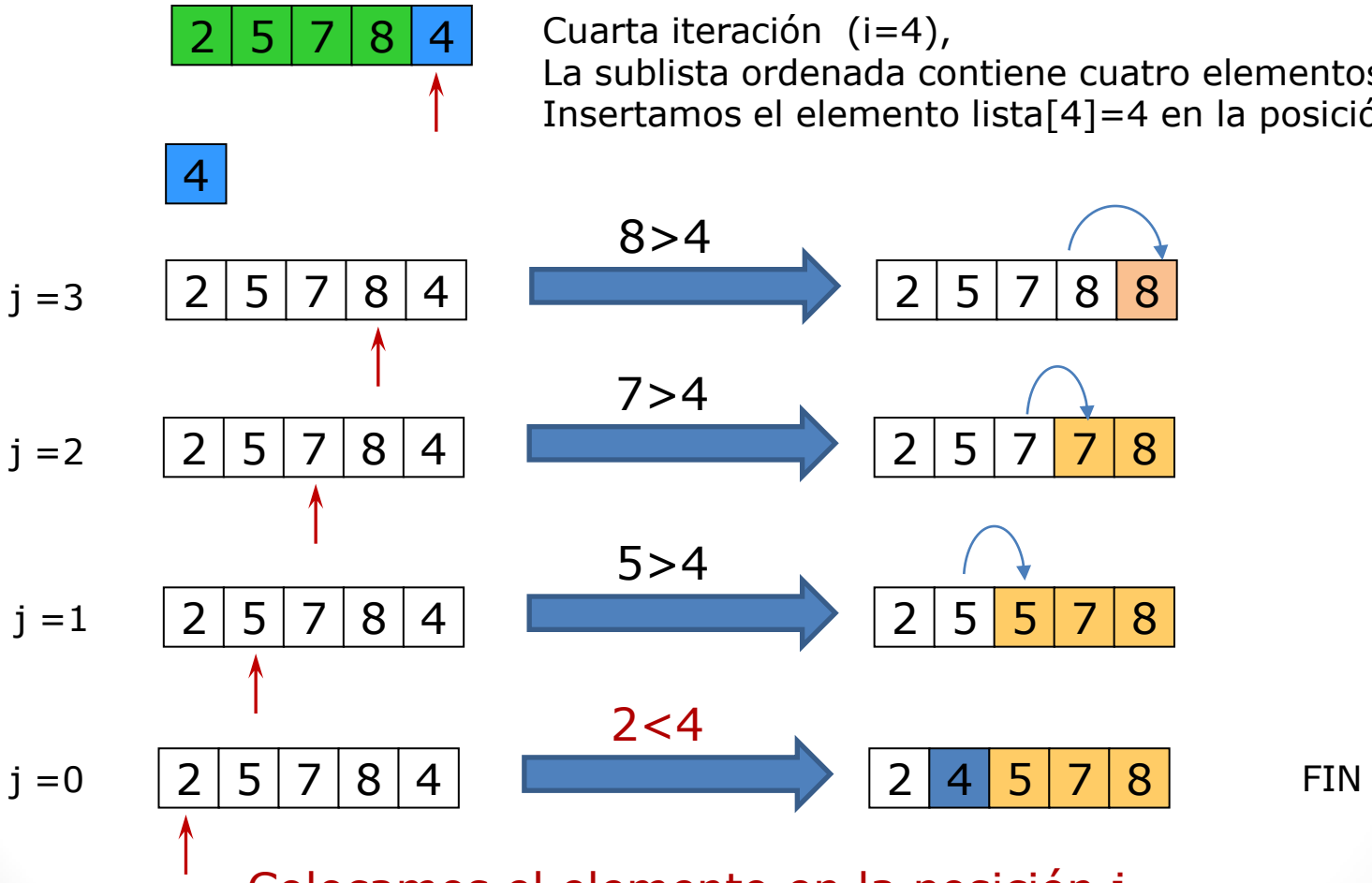
Tercera iteración (i=3)
La sublista ordenada contiene tres elementos
Insertamos el elemento lista[3]=5 en la posición correcta.



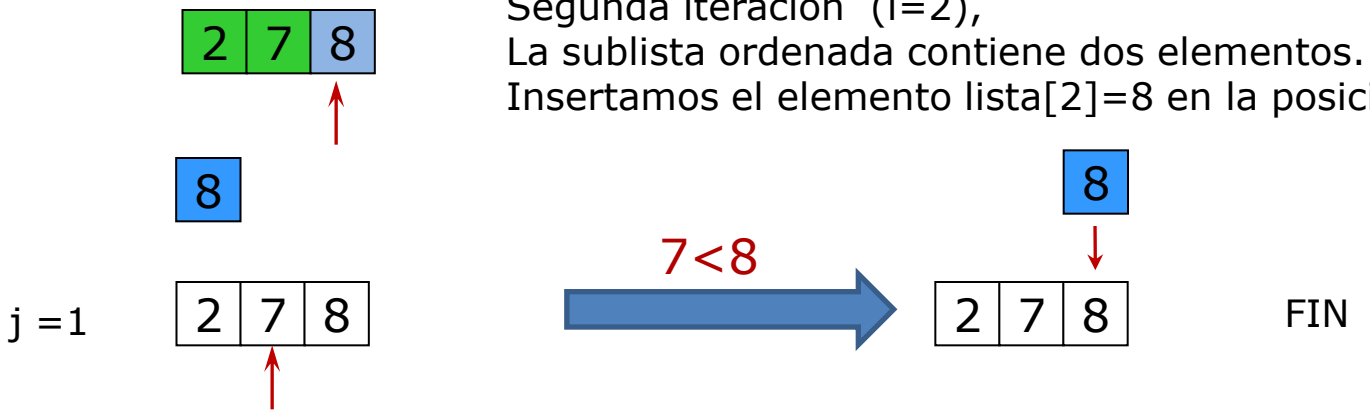
Cuarta iteración (i=4),
La sublista ordenada contiene cuatro elementos.
Insertamos el elemento lista[4]=4 en la posición correcta.

FIN

Algoritmo de inserción directa. Continuación



Algoritmo de inserción directa. Cont



Segunda iteración (i=2),
La sublista ordenada contiene dos elementos.
Insertamos el elemento lista[2]=8 en la posición correcta.

Algoritmo de inserción directa

```
for (i=1; i<N; i++){
  //N-1 iteraciones
  dato=a[i]; //Dato a colocar en la sublista ordenada de i
  elementos.
  j=i-1;      //Índice del último elemento de la sublista.
  while ( (j>=0) && (a[j]>e) ){
      a[j+1]=a[j];
      j= j-1;
  }
  a[j+1]=e; //Se coloca el elemento en la posición correcta.
}
```

Comparación de algoritmos de ordenación

- ¿Cuánto tardo en ordenar un vector de N elementos?
  - La eficiencia de un algoritmo se mide en términos del número de comparaciones y movimientos de datos necesarios para llegar a la solución
    - Puede ser diferente para distintas situaciones de partida: casi ordenado, desordenado del todo, ordenado a la inversa
      - Hay algoritmos que 'saben aprovechar' si la lista ya está ordenada y otros no
  - Los algoritmos sencillos (burbuja, inserción, selección) no son adecuados para muchos elementos
    - Para pocos elementos tiene menor trascendencia que para muchos
  - Para números mayores hay otros algoritmos más eficientes: shell, heap, merge, quicksort

Otros algoritmos: Shell

- Shell = Inserción por incrementos decrecientes
  - Modificación del método de inserción en la que no se comparan elementos contiguos sino separados un incremento que va disminuyendo en cada iteración
  - En la ultima iteración el intervalo del salto es 1
  - En la primera iteración el intervalo es la mitad del intervalo entre el primero y el último
  - En las siguientes, cada vez la mitad

  - Las cartas se van acercando rápidamente a su posición definitiva
  - En la última iteración, están casi en su sitio

  - Un ejemplo, con cartas
    - http://www.youtube.com/watch?v=QTtHQVRiD04

Source: http://latecladeescape.com [link]

Otros algoritmos: Quicksort

- Quick Sort (http://www.youtube.com/watch?v=cNB5JCG3vts)
  - Elegir un valor pivote, normalmente el primero de la lista
  - Dividir la lista en dos mitades (divide y vencerás)
    - Por un lado los inferiores y por otro los superiores
    - Colocar el pivote en medio
    - Previamente comprobar que el pivote es mayor que al menos dos elementos
      - Si el tamaño del grupo es dos, basta compararlos e intercambiarlos si es necesario
  - Repetir el proceso recursivamente para todos los grupos mayores de dos elementos

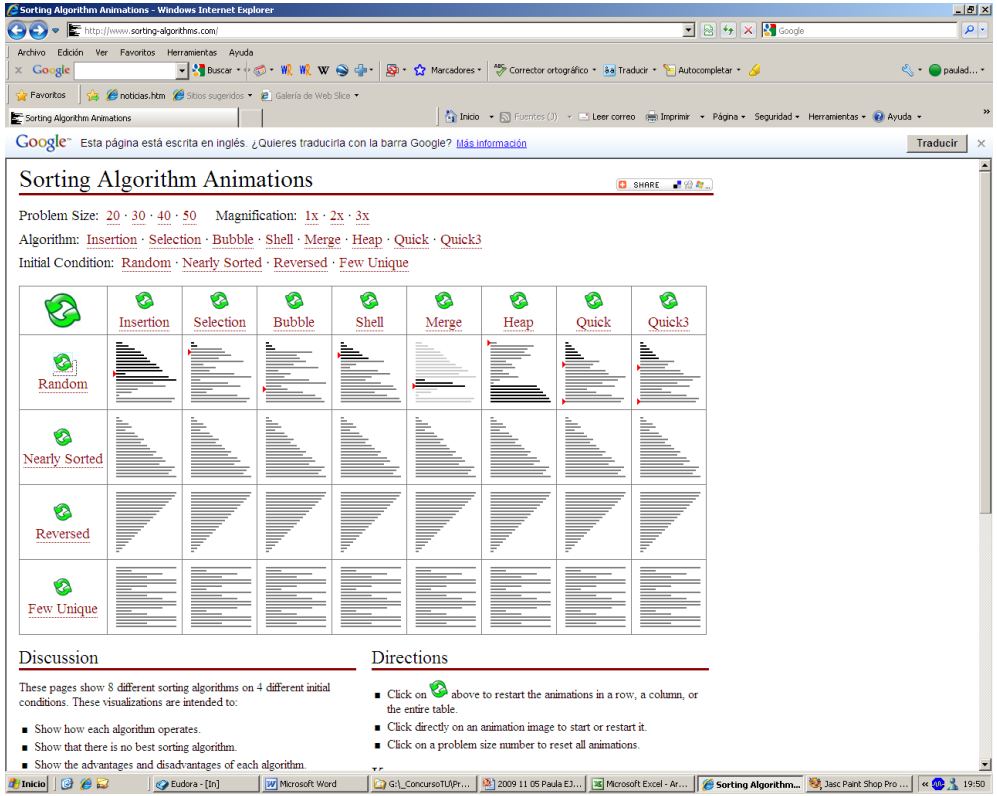
Otros algoritmos: Quicksort. Un ejemplo

12	6	15	24	7	31	21	3	19	8
6	7	3	8	**12**	**15**	24	31	21	19
3	**6**	**7**	8	**12**	**15**	**24**	31	21	19
3	**6**	**7**	**8**	**12**	**15**	21	19	**24**	31
3	**6**	**7**	**8**	**12**	**15**	**19**	**21**	**24**	**31**

- Animación en wikipedia
  - » http://en.wikipedia.org/wiki/Quicksort
- ¿Se reduce mucho el número de comparaciones y cambios respecto a método de la burbuja?

Comparación de algoritmos de ordenación

- Veámoslo en funcionamiento:
  - Visualización para diferentes algoritmos y casos
  - http://www.sorting-algorithms.com/



Source: http://www.sorting-algorithms.com/ [link]

7.3. MEZCLA

M.P. Sesmero, P. de Toledo, F.J. Ordoñez, J. Gómez-Romero, J.A. Iglesias, J.L. Mira

7.3. Mezcla

- Mezclar = combinar secuencias ordenadas en una secuencia única, intercalando los elementos ya ordenados en cada secuencia de modo que la secuencia final se mantenga ordenada
  - Es utilizado por algunos algoritmos de ordenación que se basan en subdividir la lista en partes más pequeñas que se ordenan y luego se mezclan.
    - ordenación por fusión (merge)
  - Es necesario si la cantidad de datos a ordenar es tan grande que no cabe simultáneamente en la memoria
    - se divide en sublistas, se ordenan y luego se mezclan

http://www.youtube.com/watch?v=EeQ8pwjQxTM

7.3. Mezcla

```
const int T1=100, T2=50; // tamaños de los vectores
int vector1[T1],vector2[T2],vector3[T1+T2];
//dos listas ordenadas(a y b)y una lista a la que copiarlas.

//índices para recorrer cada lista, y un auxiliar:
int i1,i2,i, k;
i1=0; i2=0; i=0;

while((i1<T1)&&(i2<T2)){
   if (vector1[i1]<vector2[i2]){
      vector3[i]= vector1[i1];
      i1 =i1+1;
    }
   else{
      vector3[i]= vector2[i2];
      i2=i2+1;
    }
   i=i+1;
}
…
```

7.3. Mezcla

```
...
//Terminamos de copiar la sublista mas larga
if (i1<T1)
// El vector1 no se ha copiado entero, copio lo que falta
   for (k=i1; k<T1; k++){
      vector3[i] = vector1[k];
      i=i+1;
   }
else
//El vector2 no se ha copiado entero, copio lo que falta
   for (k=i2; k<T2; k++){
         vector3[i] = vector2[k];
         i=i+1;
   }
```

TEMA 7.
ALGORITMOS DE BÚSQUEDA, ORDENACIÓN Y MEZCLA

Grado en Ingeniería en Tecnologías Industriales

Programación

M.P. Sesmero, P. de Toledo, F.J. Ordoñez, J. Gómez-Romero, J.A. Iglesias, J.L. Mira

Universidad Carlos III de Madrid
www.uc3m.es