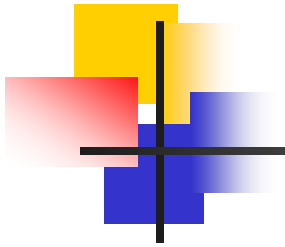


Algoritmos de ordenación y búsqueda

- Objetivos:
  - Tener una visión general de los métodos de ordenación y búsqueda más comunes.
  - Programar varios métodos de ordenación y búsqueda en lenguaje C.

Algoritmos de ordenación

- Se parte de un vector de elementos desordenados (números enteros, cadenas...)
- Se pretende ordenar los elementos del vector según algún criterio. Parámetros a tener en cuenta:
  - Tiempo utilizado por el algoritmo para realizar la ordenación.
  - Memoria del ordenador utilizada durante la ordenación.
- Algunos algoritmos de ordenación realizan del orden de N^2 operaciones. Si N es el número de elementos a ordenar. Son métodos directos, de pocas líneas de código, de tal forma que si N es pequeño, son lo suficientemente eficaces.
- Los algoritmos de ordenación más eficientes realizan del orden de $N \cdot \log_2 N$ operaciones.

Un método simple: Método de ordenación burbuja

- Consiste en recorrer sucesivamente el vector comparando los elementos consecutivos e intercambiándolos (si no están en orden).
- Vamos a ordenar números enteros, de izquierda a derecha.

10 2 41 56 3 1 57 23 45 22

- El resultado que buscamos es el siguiente:

1 2 3 10 22 23 41 45 56 57

Un método simple: Método de ordenación burbuja

- Realizamos una pasada por el vector, comparando los elementos del vector por parejas. Si no se cumple el orden los intercambiamos.
- Primera pasada:

10 2 41 56 3 1 57 23 45 22

- Intercambiamos 2-10 | 2 10 41 56 3 1 57 23 45 22 |
- Intercambiamos 3-56 | 2 10 41 3 56 1 57 23 45 22 |
- Etc...
- Los elementos mayores “suben” como burbujas.
- Fin de la primera pasada: | 2 10 41 3 1 56 23 23 45 22 57 |
- ¿Cuántas pasadas debemos dar?

Método burbuja

```
void OrdenarBurbuja(int n, int *pvector)
{
   int i, j, aux;

   for(j = 1; j < n; j++)
   {
       for(i = 0; i < n - j; i++)
       {
              if(pvector[i] > pvector[i+1])
              { // Si el elemento de la derecha es menor: se intercambian
                     aux = pvector[i]
                     pvector[i] = pvector[i+1];
                     pvector[i+1] = aux;
              }
       }
   }
}
```

Otro método

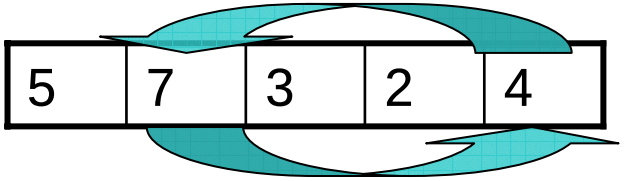
- Vamos a elaborar un método para ordenar. Partimos de este vector

5	7	3	2	4

→

2	3	4	5	7

- Por ejemplo, podemos pensar en obtener el máximo del vector y colocarlo al final, intercambiándolo por el número que se encuentre en esa posición.

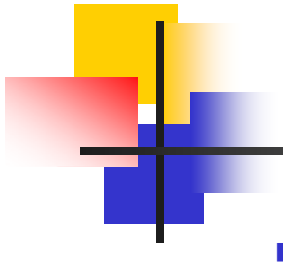


5	4	3	2	7

- El número más a la derecha ya se encuentra ordenado ¿Qué hacemos ahora?
- Volvemos a obtener el máximo del resto del vector y volvemos a intercambiar.

5	4	3	2	7

2	4	3	5	7



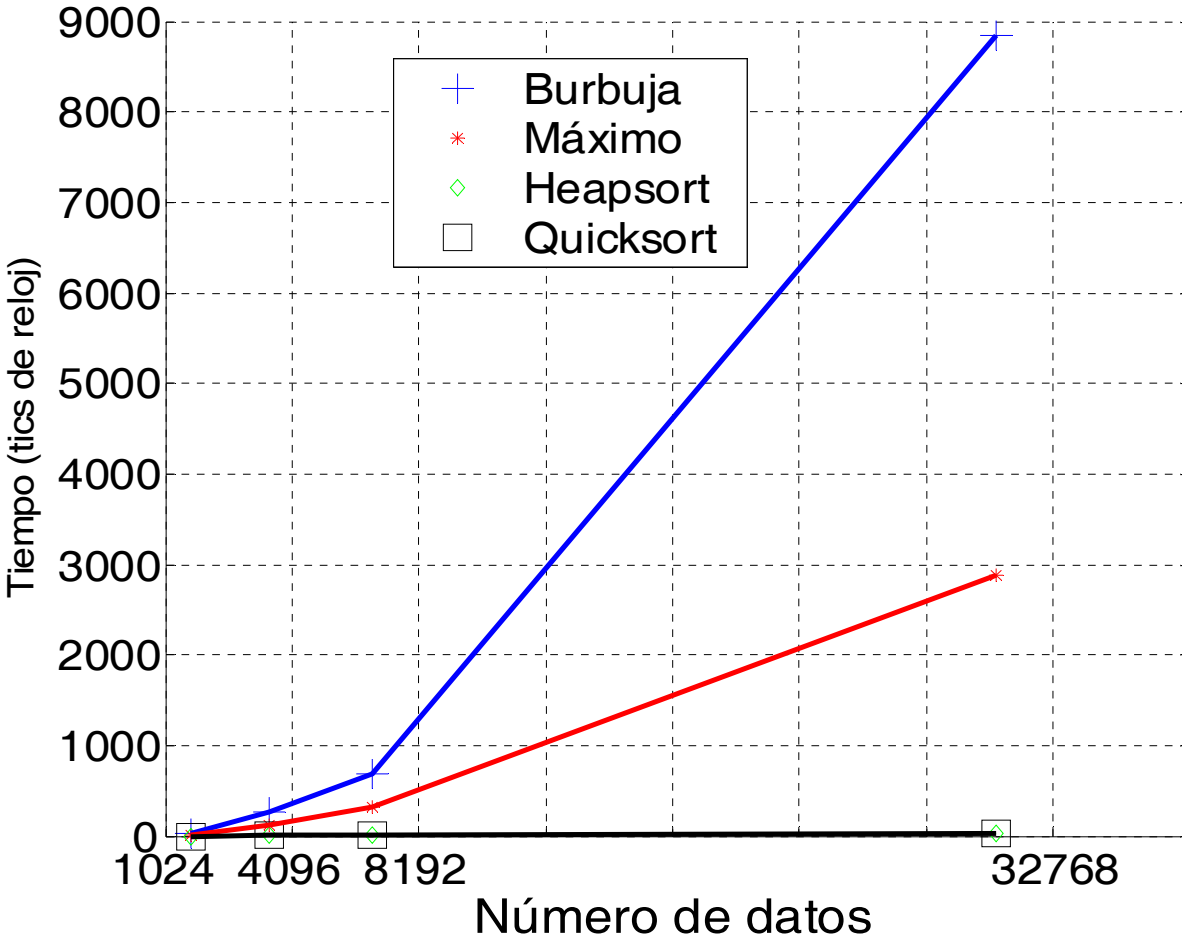
Otro método

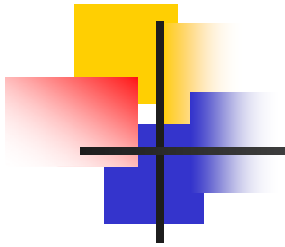
- Ahora vamos a pensar en como codificar el algoritmo.
- ¿Cuántas pasadas deberemos dar para ordenar el vector?
- Vamos a hacer una función que nos devuelva el índice del máximo valor en el vector hasta cierto elemento.
- A continuación deberemos poner el máximo en el elemento más a la derecha y continuar.

Método

```
void OrdenarMaximo(int *v, int n)
{
       int i;
       int ind, temp;

       for(i=0; i < n-1; i++) //n-1 pasadas
       {
                ind = Maximo(v, n-1-i);

                temp = v[n-1-i];
                v[n-1-i] = v[ind];
                v[ind]=temp;
       }
}

int Maximo(int *v, int n)
{
    int i, indMax;
    int max=-32768; //Mínimo entero

    for(i=0; i <= n;i++)
    {
             if(v[i]>max)
             {
                    max = v[i];
                    indMax = i;
             }

    }
      return indMax;
  }
```

Comparación de Métodos





Algoritmos de búsqueda

- Queremos conocer si se encuentra un dato en el vector.
- Pretendemos evitar la comparación con todos los datos del vector.
- Búsqueda binaria: Para este algoritmo es necesario que el vector se encuentre ordenado.
  - Consiste en situarse en el dato central del vector.
  - Si el dato a buscar es menor, rechazamos los elementos de la derecha.
  - Si el dato a buscar es mayor, rechazamos los elementos de la izquierda.
  - El proceso continua hasta que encontramos el dato.

Búsqueda binaria

```
…
inicio = 0;
final = n-1;
while(inicio <= final)
{
  medio = (inicio+final)/2;
  if(dato == v[medio])
    return (medio);
  else if (dato > v[medio])
      inicio = medio+1;
  else
      final = medio-1;
}
return -1;
…
```