# Problemas y Soluciones comunes 2

## Búsqueda

Habíamos visto las búsquedas del menor y el mayor, ahora vamos a ver la búsqueda de un elemento en particular, hay dos formas secuencial y binaria.

## Búsqueda secuencial o lineal

La búsqueda secuencial se puede aplicar en cualquier caso, el método es buscar posición por posición desde el principio hasta encontrar el elemento buscado. Por ejemplo, en un arreglo de 20 elementos enteros buscamos el número 20:

```
for(int n=0;n<20;++n) if (arreglo[n]==20) break;
```

En n estará la posición del elemento que vale 20 o si n vale 20 (que estaría fuera del arreglo) es que el elemento 20 no existe en ese arreglo.

Este algoritmo requiere una sola comparación si el elemento buscado es el primer elemento del conjunto, o N comparaciones, en el peor de los casos, siendo N el tamaño del arreglo.

Busqueda lineal en una matriz

Normalmente se aplica el método anterior por cada línea de la matriz, para ello se anidan dos bucles , el interno como el del ejemplo anterior y el externo que siga las lineas de la matriz, por ejemplo buscamos en una matriz de 10x20 de enteros el elemento 20:

```
for(int l=0;l<10;++l)
{
	for(int n=0;n<20;++n) if (arreglo[l][n]==20) break;
	if(n<20) break; //es necesario un segundo break para salir del
segundo bucle.
}
```

Si l>=10 es porque no se encontró el elemento.

Búsqueda binaria

Este método sólo puede aplicarse a conjuntos cuyos elementos están ordenados, ya sea en forma ascendente o descendente. También se lo conoce con el nombre de búsqueda por bisección o dicotómica. Supongamos que queremos buscar en un vector ordenado en forma ascendente,

El método consiste en partir sucesivamente por mitades el vector y preguntar si el valor buscado es el elemento ubicado en la mitad de la tabla. Si la respuesta es positiva se finaliza la búsqueda. Sino, el valor buscado

debe estar en una de las dos mitades del vector con lo cual se descartan todos los elementos de la otra mitad. Con este segmento del vector donde posiblemente está el elemento buscado, se repite el proceso descripto, partiéndolo por la mitad y preguntando si el valor buscado es igual al contenido del elemento de la mitad. Este proceso se repite hasta que se encuentre el valor buscado o hasta que se llegue al límite de consultas en cuyo caso, el valor buscado no está en el vector.

En los casos en que existan repeticiones en el vector del valor buscado, este algoritmo obtendrá uno de ellos aleatoriamente según los lugares que ocupen, los cuales necesariamente son consecutivos.

Visualmente se puede comparar a la búsqueda de una palabra en un diccionario, como sabemos que las palabras están ordenadas alfabéticamente, abrimos el diccionario a la mitad, si encontramos la palabra ya está, sino vemos si esta en la mitad inferior o en la mitad superior comparando con la palabra que está en el medio. Una vez definida que mitad es la que puede contener la palabra, volvemos a abrir en la mitad de esta mitad y repetimos el procedimiento hasta encontrar la palabra o que ya no haya más subdivisiones posibles.

**Descripción del Método:**

El primer paso consiste en encontrar aproximadamente, la posición media del vector y examinar el valor que contiene: A[medio]. Al comparar el dato buscado K con A[medio]

puede suceder :

a) K < A[medio]

En este caso, se considere la primera mitad de la tabla como la próxima tabla de búsqueda. En esta nueva tabla, se calcula nuevamente el elemento situado en la posición central y se repite todo el proceso presentándose nuevamente uno de lo; tres mencionados.

b) K = A[medio]

Finaliza la búsqueda y el elemento buscado se encontró, en la posición "medio".

c) K > A[medio]

Se considera la segunda mitad de la tabla y se procede como en el caso a).

El procedimiento se repetirá en forma Iterativa hasta hallar el elemento buscado en alguna de las sucesivas iteraciones, o hasta agotar la tabla

(cuando el intervalo de búsqueda queda vacío), en cuyo caso dicho elemento no pertenece al conjunto dado.

La posición media del vector se determina en cada paso como la parte entera de la

siguiente expresión: (valor Inferior del índice + valor final del índice) / 2.

Ejemplo Dado el siguiente conjunto de 8 elementos ordenados en forma ascendente

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 12 | 15 | 17 | 22 | 27 | 33 | 38 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

El elemento a buscar es K =33

Determinamos primero la posición media del vector:

medio = (0 +7)/2 =3

El dato ubicado en dicha posición es A[3]=17

Como K=33 es mayor que 17, se eliminan los elementos A[0], A[2] y A[3] y se continúa la búsqueda con los elementos de la segunda mitad: A[4], A[5], A[6] y A[7].
Se determina el elemento central de esta nueva tabla, donde el valor del índice inicial es 4 y el final 7. Por lo tanto:

medio = (4+7) /2 **=** 5

Y el elemento central es ahora A[5]=27.

Como el valor K es mayor que A[5] se deben analizar los elementos que están a su derecha: A[6] y A[7].

Para hallar el elemento central de este nuevo conjunto, se debe considerar el valor del índice Inicial 6 y el final 7. Luego, la posición media está dada por:

medio **=** (6+7) /2 **=** 6 ==> A[medio] = A[6] =33

Como A[6] es igual a K, la búsqueda finaliza determinando que el elemento está en la posición 6.

Vimos que en cada paso se analiza un vector, para el cual se definen diferentes valores extremos del índice.

Por lo tanto para plantear el algoritmo se necesitan: una variable, para, guardar el valor Inicial del índice del vector que se está analizando; una segunda variable para el valor superior y una tercera, para guardar el valor de la posición **central.** En el primer paso, el valor inicial es 1 y el superior es N. Luego dependerán del vector que se considere para búsqueda. Para la resolución se ha planteado una función de búsqueda:

Este método permite consultar una tabla en forma eficiente ya que no es necesario recorrerla toda, elemento por elemento, para encontrar un determinado valor o para determinar que ese valor no está en la tabla.

Ejemplo de implementación de una función de búsqueda binaria para float cuyos parámetros son una arreglo de float a, el tamaño del arreglo t, y el dato a buscar d, obviamente la función devuelve la posición dentro el arreglo donde está d, sino encuentra ninguna devuelve -1:

```
int binaria(float a[],int t,float d)

{

   int medio, inicio, final;

   inicio=0;

   final=t;

   do{

      medio=(inicio+final)/2;

      if(d>a[medio])

         inicio=medio+1   //si es mayor esta del lado derecho

      else if(d<a[medio])  //sino es mayor probamos si es menor

               final=medio-1 //si es menor está del lado izquierdo

            else //sino es mayor, ni menor, es igual

               return medio;

   }

   while(inicio<final); //cuando inicio y final se igualen ya no queda donde
buscar

   return -1; //no se encontró el dato

}
```

Ordenamiento

Para la búsqueda binaria hay que tener los elementos ordenados, pero que pasa cuando no los tenemos, para eso existen varios algoritmos que se encargan de ordenar una serie de elementos.

Entre los métodos de ordenamiento conocidos, podemos hacer la siguiente clasificación, sobre la base de su eficiencia:

Ello indica que los métodos avanzados son mucho más eficientes. Pero antes pasar a algoritmos más rápidos estudiaremos los métodos directos, por lo siguiente:

Son más sencillos y obvios y nos permitirán analizar los principios de clasificación.
Los algoritmos son cortos y fáciles de entender.
Los métodos avanzados, sI bien requieren menor número de operaciones, basan en algoritmos complejos y, para N pequeños los métodos directos son más rápidos.

## MÉTODOS DIRECTOS DE CLASIFICACION

### 1) Selección Directa o Mínimos Sucesivos

El problema planteado es: Dado un vector A de N elementos, ordenar sus elementos en forma ascendente.

Este método consiste en colocar en la primera posición del vector el menor de los elementos del mismo; luego en la segunda posición, el menor de los N-1 elementos restantes; proseguir con los N-2 elementos restantes seleccionando el menor de para la tercera posición y, así sucesivamente hasta que solamente quede el mayor todos los elementos..

Para ello, en la primer pasada" debemos comparar el contenido del primer elemento con el valor del segundo elemento, y si este último resulta menor, intercambiar valores de tal manera que en la primera posición siempre quede el menor. Si repetir este proceso entre el primero y el tercero, luego entre el primero y el cuarto sucesivamente, hasta comparar el primer elemento con el último, tendremos seguridad que en la primera posición dejamos el menor de todos los elementos arreglo.

Consideremos ahora los N-1 elementos restantes, desde el segundo elementos hasta el último. Si realizamos el proceso de comparación dado para la primer pasada pero ahora entre el segundo elemento y todos los restantes, obtendremos en la segunda posición del vector, el menor entre todos los elementos analizados.

Para ordenar todo el vector, repetimos este proceso sucesivamente hasta compararr finalmente el penúltimo elemento contra el último, después de lo cual termina el proceso.

Ejemplo: Dado el siguiente vector A de dimensión N=6

| 12 | 10 | 17 | 9 | 14 | 8 |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

La primer pasada al comparar A[0] con cada uno de los elementos del vector, resultan los siguientes intercambios:

con A[1] 10 12 17 9 14 8 (intercambia el 12 por el 10)

con A[2] 10 12 17 9 14 8 (sin cambios)

con A[3] 9 12 17 10 14 8 (intercambia el 10 por el 9)

con A[4] 9 12 17 10 14 8 (sin cambios)

con A[5] 8 12 17 10 14 9 (intercambia el 9 por el 8)

Obtenemos un vector que contiene en la primera posición el menor de todos sus valores. Ahora, partiendo de este último vector, debemos realizar el mismo proceso para dejar en A[1] el menor de todos los elementos que quedan. Al comparar A[1] con cada uno de los elementos restantes obtendremos el siguiente vector

| 8 | 9 | 17 | 12 | 14 | 10 |
|---|---|---|---|---|---|

En las sucesivas pasadas nos quedan los siguientes vectores

| 8 | 9 | 10 | 17 | 14 | 12 |
|---|---|---|---|---|---|
| 8 | 9 | 10 | 12 | 17 | 14 |
| 8 | 9 | 10 | 12 | 14 | 17 |

## 2) Intercambio Directo o Método de la Burbuja

Esto método se funda en clasificar pares de elementos contiguos, comenzando de atrás hacia adelante (de derecha a izquierda), de modo que al recorrer el vector la primera vez el elemento más pequeño ocupará la primera posición.

Luego, recorremos el vector desde su último elemento A[N], hasta el A[1] haciendo los intercambios necesarios -SIEMPRE ENTRE ELEMENTOS ADYACENTES

Luego recorremos desde A[N] hasta A[2], y así sucesivamente.

Con un poco de imaginación si colocamos el arreglo en posición vertical, poder asimilar a burbujas los elementos que ascienden una posición. De allí, el nombre método.

Veamos un ejemplo de función que se le entrega un arreglo desordenado de enteros y lo devuelve ordenado:

```
void burbuja(int a[], int t)

{

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

      for(int j=t;j>i;--j) if(a[j-1]>a[j]) intercambio(a[j-1],a[j]);
//intercambio es la de la actividad 4.5

}
```

### 3) Inserción Directa o Método de la Baraja

Supongamos que queremos clasificar en forma ascendente (de menor a mayor) e siguiente arreglo A(i) de 8 elementos:

| 44 | 55 | 12 | 42 | 94 | 18 | 6 | 67 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Este método consiste en tomar el elemento A[i] del arreglo y comenzar compararlo con los elementos que están a su izquierda, hasta encontrar un elemento menor que A[i]. Entonces, se Inserte A[i] en ese lugar.
Obviamente el método comienza a investigar el segundo elemento ( A[1] ) del arreglo.

En nuestro ejemplo los pasos sucesivos serian:

Elemento a Insertar Arreglo A[i]

Vector inicial 44 55 12 42 94 18 06 67

A[1] 44 55 12 42 94 18 06 67

A[2] 12 44 55 42 94 18 06 67

A[3] 12 42 44 55 94 18 06 67

A[4] 12 42 44 55 94 18 06 67

A[5] 12 18 42 44 55 94 06 67

A[6] 06 12 18 42 44 55 94 67

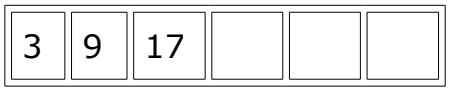
A[7] 06 12 18 42 44 55 67 94

Analicemos en el ejemplo el paso en que se insertó el elemento A[3]=42. Se fue comparando este elemento con los de su Izquierda hasta ubicarlo en la nueva posición A[1]. Los elementos 55 y 44, a su vez, fueron corridos un lugar a la derecha.

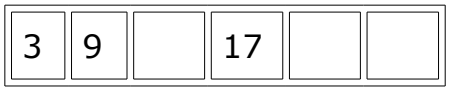
Inserción de un elemento en un vector

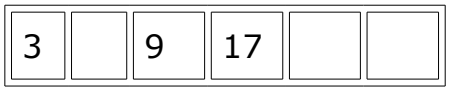
La inserción de un elemento en un vector no se da de manera automática, el problema es que para insertarlo primero hay que hacer espacio en dicho vector:

supongamos que tenemos un vector

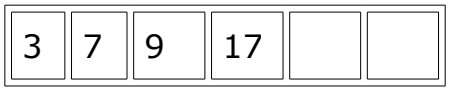



y queremos insertar el 7 para que quede en orden, para eso buscamos en que lugar le corresponde, n nuestro caso la posición 1, entonces debemos hacer lugar corriendo los otros números:




Luego que tenemos el espacio asignamos el 7




Como ven, se empieza a correr de atrás hacia adelante, ya que si lo

hiceramos al revés  y moviéramos primero el 9  escribiríamos encima del 17 y lo perderíamos.
Obviamente hay que disponer de espacio en el vector.

También se puede ordenar copiando un vector en otro pero al copiar ir insertándolo en el lugar que debería ir según el orden. En nuestro ejemplo anterior podía ser el segundo vector al cual se le está copiando el cuarto elemento que es el 7.

Por el problema del espacio al correr elementos, en el primer ejemplo donde inserta en el propio vector el elemento vacío es el que se saca, veamos de nuevo el ejemplo:

Elemento a Insertar Arreglo A[i]

Vector inicial 44 55 12 42 94 18 06 67

A[1] 44 55 12 42 94 18 06 67

A[2] 12 44 55 42 94 18 06 67

El siguiente valor a revisar es el A[3] que vale 42, que según el algoritmo le correspondería estar en el A[1] (en el segundo lugar, y el resto de ahí hacia adelante debe correrse, pero al parecer no hay lugar en el vector.

En realidad si hay, en el cuarto lugar A[3], va a estar vacío porque vamos a sacar de ahí el 42, entonces se empieza a correr (de atras para adelante) desde A[3] hasta A[1] que es donde se va a insertar el 42:

Empezamos con:

12 44 55 42 94 18 06 67  primero salvamos en una variable auxiliar el 42 Aux=42;

luego copiamos el 55 al lugar del 42 A[3]=A[2];

12 44 55 55 94 18 06 67

luego copiamos el 44 al lugar del 45 A[2]=A[1];

12 44 44 55 94 18 06 67

por último copiamos en el lugar que le corresponde el 42 A[1]=Aux;

12 42 44 55 94 18 06 67

## METODOS DE CLASIFICACIÓN AVANZADOS

## 1) Método de Shell

D. L. Shell propuso en 1959 un refinamiento del método de inserción directa que hemos visto, en el cual se produce un acercamiento de los elementos descolocados hacia su posición correcta en saltos de mayor longitud.

Se repite un mismo proceso con una distancia de comparación que, inicialmente es la mitad de la longitud del vector y que se va reduciendo a la mitad en cada repetición 4 hasta que dicha distancia vale 1.

Cada pasada termina al detectarse que no se ha producido ningún cambio de elementos en la distancia correspondiente.

Analicemos su modificación sobre un vector ejemplo A de 9 elementos.

Primero conforma grupos con todos los elementos que estén separados 4 posición es decir A[0] con A[4], A[1] con A[5], A[2] con A[6], A[3] con A[7], y A[4] con A[8].

Luego clasifica estos grupos con el método de Inserción directa. A este proceso lo llama "clasificación 4".

Luego agrupa a todos los elementos separados por 2 posiciones, o sea: A[0] con A[2], A[1] con A[3], A[2] con A[4] y así sucesivamente.

Y clasifica cada uno de estos grupos por inserción directa.

Por último agrupa los elementos distanciados por 1 posición, es decir, elementos adyacentes; con lo que obtiene un solo grupo, que es el mismo arreglo. Al aplicarle ahora inserción directa, queda completamente ordenado.

El último paso es el método de inserción directa propiamente dicho, solo que su trabajo será mucho más ágil, debido a los acomodamientos previos.

Veamos como funciona el método en el vector ejemplo:

| 3 | 2 | 4 | 6 | 7 | 3 | 5 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Primer paso: distancia de comparación 4

| 3 | 2 | 4 | 6 | 7 | 3 | 5 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| 3 | 2 | 4 | 1 | 1 | 3 | 5 | 6 | 7 |
| 3 | 2 | 4 | 1 | 1 | 3 | 5 | 6 | 7 |
| 1 | 2 | 4 | 1 | 3 | 3 | 5 | 6 | 7 |

Ordenados los pares de distancia 4

Segundo paso: distancia de comparación 2

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 1 | 3 | 3 | 5 | 6 | 7 |
| 1 | 1 | 3 | 2 | 4 | 3 | 5 | 6 | 7 |

Ordenados los pares de distancia 2

Tercer paso: distancia de comparación 1

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 2 | 4 | 3 | 5 | 6 | 7 |
| 1 | 1 | 2 | 3 | 3 | 4 | 5 | 6 | 7 |

Ordenados los pares de distancia 1

## 2**) Método QuickSort**

El **ordenamiento rápido** (**quicksort** en inglés) es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar *n* elementos en un tiempo proporcional a *n* log *n*. Esta es la técnica de ordenamiento más rápida conocida. Fue desarrollada por C. Antony R. Hoare en 1960. El algoritmo original es **recursivo**, pero se utilizan versiones iterativas para mejorar su rendimiento (los algoritmos recursivos son en general más lentos que los iterativos, y consumen más recursos)

Este método consiste en:

Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.
Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados. Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.
En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño.
En el peor caso, el pivote termina en un extremo de la lista. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas.

La mayoría de optimizaciones que se aplican al algoritmo se centran en la elección del pivote.

Veamos como funciona con un ejemplo

En el siguiente ejemplo se marcan el pivote y los índices i y j con las letras p, I y J respectivamente.

Comenzamos con la lista completa. El elemento divisor será el 4:

| 5 | 3 | 7 | 6 | 2 | 1 | 4 |
|---|---|---|---|---|---|---|
| | | | | | | p |

Comparamos con el 5 por la izquierda y el 1 por la derecha.

| 5 | 3 | 7 | 6 | 2 | 1 | 4 |
|---|---|---|---|---|---|---|
| I | | | | | J | p |

5 es mayor que cuatro y 1 es menor. Intercambiamos:

| 1 | 3 | 7 | 6 | 2 | 5 | 4 |
|---|---|---|---|---|---|---|
| I | | | | | J | p |

Avanzamos por la izquierda y la derecha:

| 1 | 3 | 7 | 6 | 2 | 5 | 4 |
|---|---|---|---|---|---|---|
| | I | | | J | | p |

3 es menor que 4: avanzamos por la izquierda. 2 es menor que 4: nos mantenemos ahí.

| 1 | 3 | 7 | 6 | 2 | 5 | 4 |
|---|---|---|---|---|---|---|
| | I | | | J | | p |

7 es mayor que 4 y 2 es menor: intercambiamos.

| 1 | 3 | 2 | 6 | 7 | 5 | 4 |
|---|---|---|---|---|---|---|
| | | I | | J | | p |

Avanzamos por ambos lados:

| 1 | 3 | 2 | 6 | 7 | 5 | 4 |
|---|---|---|---|---|---|---|
| | | | I | J | | p |

En este momento termina el ciclo principal, porque los índices se cruzaron. Ahora intercambiamos lista[i] con lista[t] (t es la última posición del vector)

| 1 | 3 | 2 | 4 | 7 | 5 | 6 |
|---|---|---|---|---|---|---|
| | | | p | | | |

Aplicamos recursivamente a la sublista de la izquierda (índices 0 - 2). Tenemos lo siguiente:

| 1 | 3 | 2 |
|---|---|---|

1 es menor que 2: avanzamos por la izquierda. 3 es mayor: avanzamos por la derecha. Como se intercambiaron los índices termina el ciclo. Se intercambia lista[i] con lista[t]:

| 1 | 2 | 3 |
|---|---|---|

El mismo procedimiento se aplicará a la otra sublista. Al finalizar y unir todas las sublistas queda la lista inicial ordenada en forma ascendente.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|

Ejemplo en C:

```
int colocar(int *v, int b, int t)
{
   int i;
   int pivote, valor_pivote;
   int temp;

   pivote = b;
   valor_pivote = v[pivote];
   for (i=b+1; i<=t; i++){
      if (v[i] < valor_pivote){
             pivote++;
            temp=v[i];
            v[i]=v[pivote];
            v[pivote]=temp;

      }
   }
   temp=v[b];
   v[b]=v[pivote];
   v[pivote]=temp;
   return pivote;
}


void Quicksort(int v[], int b, int t)
{
    int pivote;
    if(b < t){
      pivote=colocar(v, b, t);
      Quicksort(v, b, pivote-1);
      Quicksort(v, pivote+1, t);
    }
}
```

Nota: Los tres parámetros de la llamada inicial a Quicksort serán el arreglo, 0 y numero_elementos -1, es decir, si es un array de 6 elementos (int array[6]) la llamada será Quicksort(array, 0, 5);

# Bibliografía

Apuntes de Fundamentos de Programación, Unidad 8 Clasificación y Búsqueda, de la Carrera de Licenciatura en informática de la Universidad Autónoma de Entre Ríos (UADER).

Wikipedia.