

PRÁCTICA 2

Divide y vencerás

Álvaro Maximino Linares Herrera

0	1	2	3	4	5	6	7	8	9
24	36	48	60	72	84	96	108	120	132

Notas:

El código del problema 1 es una implementación del pseudocódigo que usted nos dio en la práctica, archivo **divideyvenceras1.cpp**.

El código del problema 1, búsqueda terciaria, es una modificación del anterior, archivo **dv.cpp**.

El código del problema 2 es una implementación de un pseudocódigo de un libro de Internet, llamado Técnicas de Diseños de Algoritmos, sus autores son Rosa Guerequeta García y Antonio Vallecillo Moreno. Archivo **dv2.cpp**.

PROBLEMA 1º:

Nos enfrentamos a un problema de divide y vencerás, la búsqueda binaria.

En este problema, nos dan un vector de enteros relleno con una serie de números ordenados de forma creciente, de modo que si dividimos el vector por la mitad y comparamos lo que buscamos con el centro, desecharemos una parte u otra del vector y repetiremos este proceso hasta dar con el elemento buscado. Si el elemento buscado no coincide con alguno de los centros que calcularemos podremos decir que no se encuentra en el vector.

Código BúsquedaBinaria:

El algoritmo para la primera parte es la implementación a partir del pseudocódigo que usted nos proporciona en la práctica, es el siguiente:

```
bool BusquedaBinaria(int *vec, int tam, int dato){

    int centro, inf=0, sup=tam-1;
    while(inf<=sup){
        centro=((sup-inf)/2)+inf; //Division entera: se trunca la fraccion
        if(vec[centro]==dato){
            cout << "La posicion es: " << centro << endl;
            return true;
        }else if(vec[centro]>dato)
            sup=centro-1;
        else
            inf=centro+1;
    }
    return false;
}
```

Los parámetros que le pasamos a la función son:

- Un puntero de enteros: int *vec
- Un entero que represente el tamaño del vector: int tam
- Un entero que represente el dato que buscamos en el vector: int dato

El funcionamiento es bien sencillo, calculamos la posición central, si es el dato que buscamos terminamos, si no lo es comparamos si es mayor o menor, si es menor desechamos la parte de la derecha haciendo que sup sea igual que la posición central menos 1, y si es mayor desechamos la parte izquierda haciendo que inf sea la posición central más 1.

```

alvaro@alvaro-Inspiron-1545:~/Escritorio/practica 2$ g++ -o po divideyvenceras1.
cpp
alvaro@alvaro-Inspiron-1545:~/Escritorio/practica 2$ ./po 10
La posicion es: 1
36
pos: 0 es: 24
pos: 1 es: 36
pos: 2 es: 48
pos: 3 es: 60
pos: 4 es: 72
pos: 5 es: 84
pos: 6 es: 96
pos: 7 es: 108
pos: 8 es: 120
pos: 9 es: 132
alvaro@alvaro-Inspiron-1545:~/Escritorio/practica 2$ █

```

Este sería un ejemplo, le pasamos al programa el tamaño de nuestro vector y genera aleatoriamente el dato a buscar y como irá relleno cada posición.

Nos dice que la posición donde se encuentra el dato, 36, es 1, y como se puede comprobar es cierto. El código para mostrarlo esta en el main y es tan sencillo como:

```

if(BusquedaBinaria(vec,tam,dato)){
cout << dato<<endl;
    for(int i=0; i<tam; i++)
        cout <<"pos: " << i << " es: " << vec[i] << endl;
}

```

Código BúsquedaTerciaria:

También se nos pide la implementación de una segunda búsqueda que en lugar del centro (dividir entre dos) use un tercio del vector (dividir entre 3).

El código de este problema es parecido al anterior, solo que en lugar del centro, usaremos un tercio, el código es el siguiente:

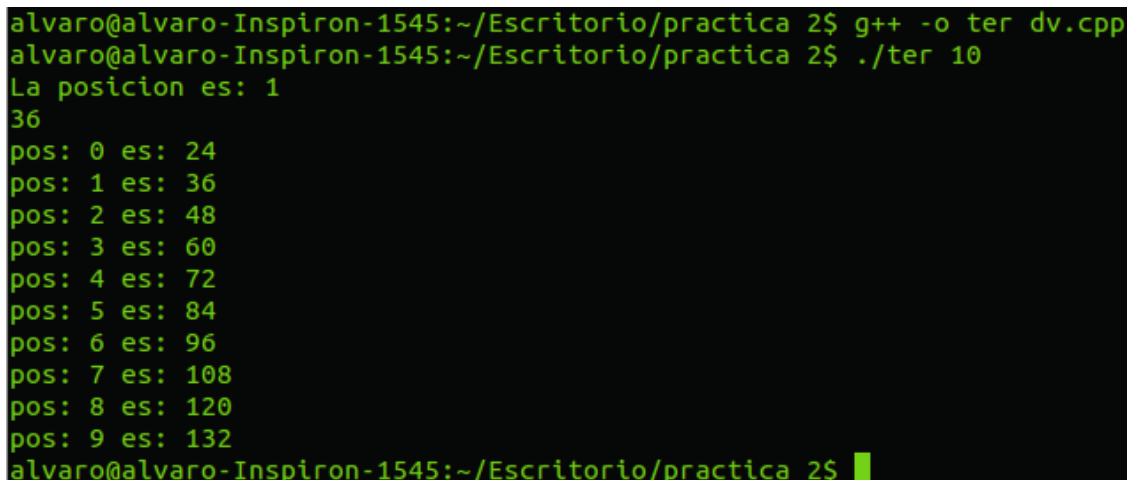
```

bool BusquedaUnTercio(int *vec, int tam, int dato){

    int tercio, inf=0, sup=tam-1;
    while(inf<=sup){
        tercio=((sup-inf)/3)+inf; //Para que divida en un tercio
        if(vec[tercio]==dato){
            cout << "La posicion es: " << tercio << endl;
            return true;
        }else if(vec[tercio]>dato)
            sup=tercio-1;
        else
            inf=tercio+1;
    }
    return false;
}

```

Como se puede apreciar en el código, lo único que cambiamos es la forma de calcular el valor a comparar, en lugar de ser el medio del vector es el tercio. El código está basado en el pseudocódigo de la práctica 2.



```

alvaro@alvaro-Inspiron-1545:~/Escritorio/practica 2$ g++ -o ter dv.cpp
alvaro@alvaro-Inspiron-1545:~/Escritorio/practica 2$ ./ter 10
La posicion es: 1
36
pos: 0 es: 24
pos: 1 es: 36
pos: 2 es: 48
pos: 3 es: 60
pos: 4 es: 72
pos: 5 es: 84
pos: 6 es: 96
pos: 7 es: 108
pos: 8 es: 120
pos: 9 es: 132
alvaro@alvaro-Inspiron-1545:~/Escritorio/practica 2$

```

Este sería un ejemplo de la búsqueda terciaria. Como vemos el dato también es 36, esto se debe a que `rand()` no es tan aleatorio como nos pensamos, sino que siempre empieza en unos determinados números.

El funcionamiento es parecido al de la búsqueda binaria, solo que, como ya mencione antes, en lugar de un medio usamos un tercio, de modo que si buscamos por la izquierda es mucho más pequeño el pedazo que nos quedamos(1/3), mientras que si es por la derecha es más grande (2/3).

¿Cuál es más eficiente?

El primero será de orden de eficiencia $O(n)$ y el segundo también, pero al ser el segundo dividido entre 3 en lugar de ser dividido entre 2 este es más lento que el primero, por lo cual el primero es más eficiente.

PROBLEMA 2º:

En este problema se nos plantea calcular la mediana de dos vectores de mismo tamaño, n , y ordenados de forma no decreciente, para ordenarlos usaremos en el main la función Burbuja que ya se uso en la práctica “cero” para mostrar como calcular la eficiencia. Hay que implementar una función que calcule el elemento de la posición n de la unión ordenada de los dos vectores, en mi caso se llamarán $v1$ y $v2$.

A parte de la función Burbuja, también haré uso de las funciones Mínimo y Máximo, las cuales reciben dos números enteros y devuelven el mínimo o el máximo.

La función Mediana esta “dividida” en tres partes, la primera es un sencillo if que será el caso base en el que $n=1$, después irá el caso de que el tamaño de los vectores sea 2, y luego el caso “más” general en el cual se harán llamadas recursivas a la función Mediana.

El código es una adaptación de un pseudocódigo de Internet en el cual me he basado, es el siguiente:

Código:

```
1. int Mediana(int *v1, int inicio_v1, int tamaño_v1, int *v2, int inicio_v2, int tamaño_v2){
2.     int posv1, posv2, numeles;
//Caso base
3.     if((inicio_v1>=tamaño_v1)&&(inicio_v2>=tamaño_v2))
4.         return Minimo(v1[tamaño_v1],v2[tamaño_v2]);
//En caso de que sean vectores de dos elementos
5.     numeles=tamaño_v1-inicio_v1+1;
6.     if(numeles==2){
7.         if(v1[tamaño_v1]<v2[inicio_v2])
8.             return v1[tamaño_v1];
9.         else if(v2[tamaño_v2]<v1[inicio_v1])
10.            return v2[tamaño_v2];
11.        else
12.            return Maximo(v1[inicio_v1],v2[inicio_v2]);
13.    }
//Caso general
```

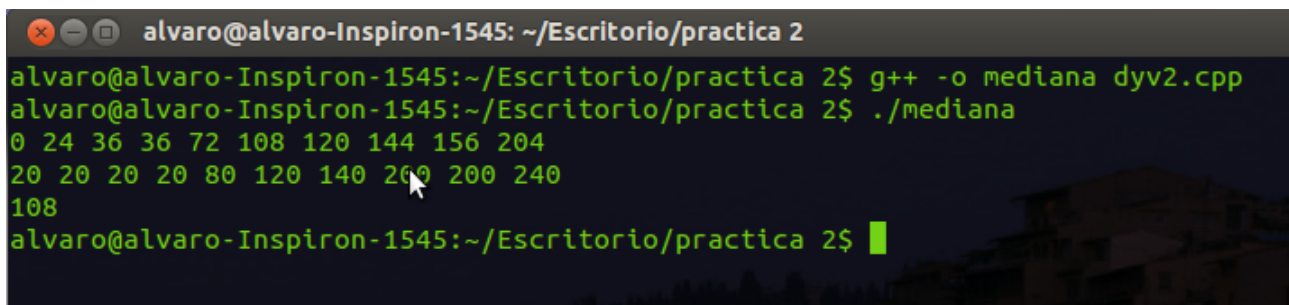
```

14.  numeles=(numeles-1)/2;
15.  posv1=inicio_v1+numeles;
16.  posv2=inicio_v2+numeles;
17.
18.  if(v1[posv1]<v2[posv2])
19.      return v1[posv1];
20.  else if(v1[posv1]>v2[posv2])
21.      return Mediana(v1,tamano_v1-
numeles,tamano_v1,v2,inicio_v2,inicio_v2+numeles);
22.  else
23.      return Mediana(v1,inicio_v1,inicio_v1+numeles,v2,tamano_v2-
numeles,tamano_v2);
24.  }

```

He numerado las lineas para facilitar su comprensión, ya que algunas al ser largas se bajan al siguiente renglón.

Un ejemplo de ejecucion sería:



```

alvaro@alvaro-Inspiron-1545: ~/Escritorio/practica 2
alvaro@alvaro-Inspiron-1545:~/Escritorio/practica 2$ g++ -o mediana dyv2.cpp
alvaro@alvaro-Inspiron-1545:~/Escritorio/practica 2$ ./mediana
0 24 36 36 72 108 120 144 156 204
20 20 20 20 80 120 140 200 200 240
108
alvaro@alvaro-Inspiron-1545:~/Escritorio/practica 2$

```

Como vemos tenemos dos vectores, de tamaño n=10, como vemos 108 sería el elemento central de la reunión ordenada de v1 y v2.