EJERCICIO 3
Sistema operativo: Ubuntu 18.04.1 LTS
Compilador usado: ii g++ 4:7.3.0-3ubuntu2 amd64 GNU C++ compiler

a) Explique qué hace este algoritmo
El algoritmo busca un valor (x) en el vector, si lo encuentra devuelve la posición en donde está ese valor.
Si no lo encuentra, devuelve -1.

Para buscar el valor, se calcula la mitad del vector y se comprueba si el valor que está en la mitad es mayor o menor que el valor buscado, si es menor se vuelve a calcular la mitad del vector pero solo de la parte de la derecha, si es mayor que el valor buscado, se calcula la mitad pero de la parte de la izquierda.

b) Calcule su eficiencia empírica
Primero lo he compilado:
$ g++ ejercicio_desc.cpp -o ejercicio_desc

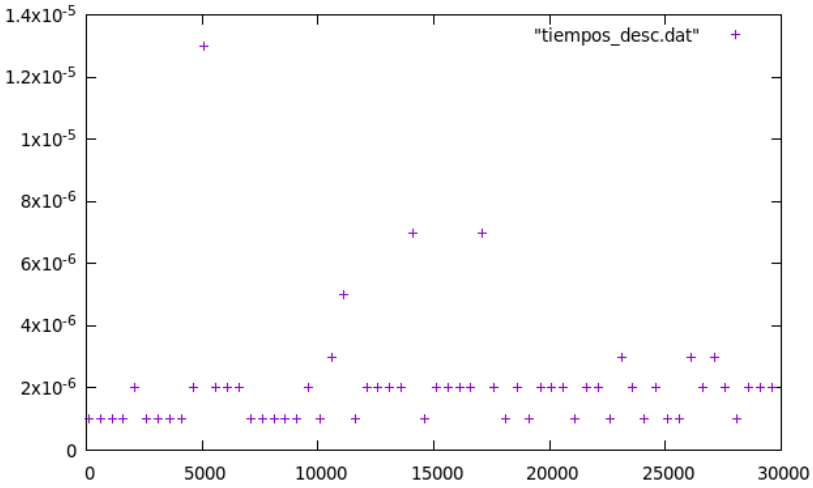
Luego he creado “ejecuciones_desc.csh”:
```
#!/bin/csh
@ inicio = 100
@ fin = 30000
@ incremento = 500
set ejecutable = ejercicio_desc
set salida = tiempos_desc.dat

@ i = $inicio
echo > $salida
while ( $i <= $fin )
  echo Ejecución tam = $i
  echo `./{$ejecutable} $i` >> $salida
  @ i += $incremento
end
```

Y he almacenado los datos en “*tiempos_desc.dat”:*
```
100 1e-06
600 1e-06
1100 1e-06
1600 1e-06
2100 2e-06
2600 1e-06
3100 1e-06
3600 1e-06
4100 1e-06
4600 2e-06
5100 1.3e-05
5600 2e-06
6100 2e-06
6600 2e-06
7100 1e-06
7600 1e-06
```

8100 1e-06
8600 1e-06
9100 1e-06
9600 2e-06
10100 1e-06
10600 3e-06
11100 5e-06
11600 1e-06
12100 2e-06
12600 2e-06
13100 2e-06
13600 2e-06
14100 7e-06
14600 1e-06
15100 2e-06
15600 2e-06
16100 2e-06
16600 2e-06
17100 7e-06
17600 2e-06
18100 1e-06
18600 2e-06
19100 1e-06
19600 2e-06
20100 2e-06
20600 2e-06
21100 1e-06
21600 2e-06
22100 2e-06
22600 1e-06
23100 3e-06
23600 2e-06
24100 1e-06
24600 2e-06
25100 1e-06
25600 1e-06
26100 3e-06
26600 2e-06
27100 3e-06
27600 2e-06
28100 1e-06
28600 2e-06
29100 2e-06
29600 2e-06

Luego he creado la gráfica con gnuplot y he obtenido:



Por lo que he notado algo anormal: los puntos no están del todo alineados. He llegado a la conclusión de que para que el algoritmo funcione, el vector debe estar ordenado. Por lo que he incluido la función ordenar del ejercicio 1. El nuevo fichero se llama “ejercicio_desc2.cpp”:

```
#include <iostream>
#include <ctime>    // Recursos para medir tiempos
#include <cstdlib>  // Para generación de números pseudoaleatorios

using namespace std;

void ordenar (int *v, int n){
  for (int i=0; i<n-1; i++)
    for (int j=0; j<n-i-1; j++){
      if (v[j] > v[j+1]){
        int aux = v[j];
        v[j] = v[j+1];
        v[j+1] = aux;
      }
    }
}

int operacion(int *v, int n, int x, int inf, int sup) {
  int med;
  bool enc=false;
  while ((inf<sup) && (!enc)) {
    med = (inf+sup)/2;
    if (v[med]==x)
      enc = true;
    else if (v[med] < x)
      inf = med+1;
    else
      sup = med-1;
  }
  if (enc)
    return med;
  else
    return -1;
}
```

```
void sintaxis()
{
  cerr << "Sintaxis:" << endl;
  cerr << "  TAM: Tamaño del vector (>0)" << endl;
  cerr << "Se genera un vector de tamaño TAM con elementos aleatorios" << endl;
  exit(EXIT_FAILURE);
}

int main(int argc, char * argv[])
{
  // Lectura de parámetros
  if (argc!=2)
    sintaxis();
  int tam=atoi(argv[1]);     // Tamaño del vector
  if (tam<=0)
    sintaxis();

  // Generación del vector aleatorio
  int *v=new int[tam];       // Reserva de memoria
  srand(time(0));            // Inicialización del generador de números pseudoaleatorios
  for (int i=0; i<tam; i++)  // Recorrer vector
    v[i] = rand() % tam;

  ordenar(v,tam);

  clock_t tini;    // Anotamos el tiempo de inicio
  tini=clock();

  // Algoritmo a evaluar
  operacion(v,tam,tam+1,0,tam-1);

  clock_t tfin;    // Anotamos el tiempo de finalización
  tfin=clock();

  // Mostramos resultados
  cout << tam << "\t" << (tfin-tini)/(double)CLOCKS_PER_SEC << endl;

  delete [] v;     // Liberamos memoria dinámica
}
```

Luego he creado su correspondiente “ejecuciones_desc2.csh”:

```
#!/bin/csh
@ inicio = 100
@ fin = 30000
@ incremento = 500
set ejecutable = ejercicio_desc2
set salida = tiempos_desc2.dat

@ i = $inicio
echo > $salida
while ( $i <= $fin )
```

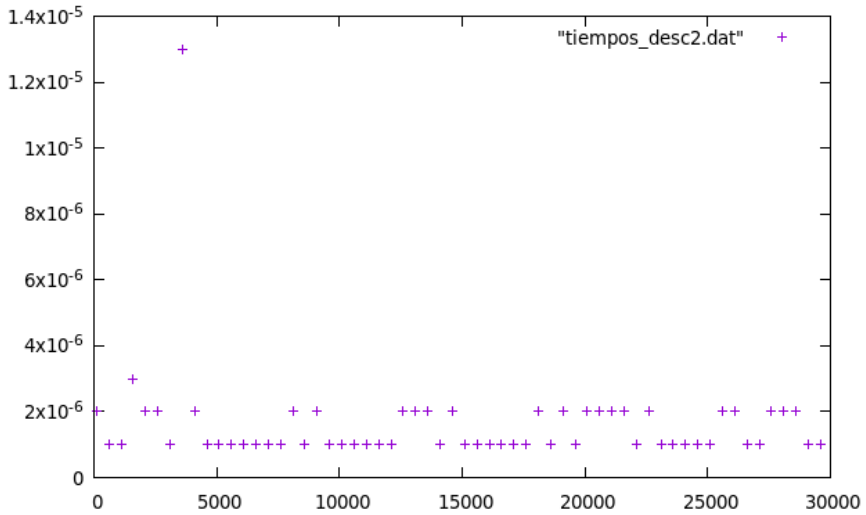
```
  echo Ejecución tam = $i
  echo `./{$ejecutable} $i` >> $salida
  @ i += $incremento
end
```

Lo he ejecutado y he calculado los datos:

100 2e-06
600 1e-06
1100 1e-06
1600 3e-06
2100 2e-06
2600 2e-06
3100 1e-06
3600 1.3e-05
4100 2e-06
4600 1e-06
5100 1e-06
5600 1e-06
6100 1e-06
6600 1e-06
7100 1e-06
7600 1e-06
8100 2e-06
8600 1e-06
9100 2e-06
9600 1e-06
10100 1e-06
10600 1e-06
11100 1e-06
11600 1e-06
12100 1e-06
12600 2e-06
13100 2e-06
13600 2e-06
14100 1e-06
14600 2e-06
15100 1e-06
15600 1e-06
16100 1e-06
16600 1e-06
17100 1e-06
17600 1e-06
18100 2e-06
18600 1e-06
19100 2e-06
19600 1e-06
20100 2e-06
20600 2e-06
21100 2e-06
21600 2e-06
22100 1e-06
22600 2e-06

23100 1e-06
23600 1e-06
24100 1e-06
24600 1e-06
25100 1e-06
25600 2e-06
26100 2e-06
26600 1e-06
27100 1e-06
27600 2e-06
28100 2e-06
28600 2e-06
29100 1e-06
29600 1e-06

Finalmente, he obtenido esta gráfica, que es una función constante:



Ahora, para calcular la regresión, he usado una función constante:
gnuplot> f(x) = a
gnuplot> fit f(x) "tiempos_desc2.dat" via a
iter chisq delta/lim lambda a
0 3.0198751147e-10 0.00e+00 6.50e-11 6.504576e-11
1 1.4844127587e-10 -1.03e+05 6.50e-12 1.573772e-06
2 1.4840000000e-10 -2.78e+01 6.50e-13 1.599996e-06
3 1.4840000000e-10 -7.72e-07 6.50e-14 1.600000e-06
iter chisq delta/lim lambda a

After 3 iterations the fit converged.
final sum of squares of residuals : 1.484e-10
rel. change during last iteration : -7.72364e-12

degrees of freedom (FIT_NDF) : 59
rms of residuals (FIT_STDFIT) = sqrt(WSSR/ndf) : 1.58596e-06
variance of residuals (reduced chisquare) = WSSR/ndf : 2.51525e-12

Final set of parameters Asymptotic Standard Error

========== ==========

a = 1.6e-06 +/- 2.047e-07 (12.8%)

gnuplot> plot "tiempos_desc2.dat", f(x)