José Santos Salvador
Sistema operativo: Ubuntu 18
Hardware: Poratil HP 15 OMEN 2017(i5, 1tb, 8gb ram …)

EJERCICIO

SCRIPT:

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

@ i = $inicio
echo > $salida
while ( $i <= $fin )
  echo Ejecución tam = $i
  echo `./{$ejecutable} $i 1` >> $salida
  @ i += $incremento
end
```

CPP:

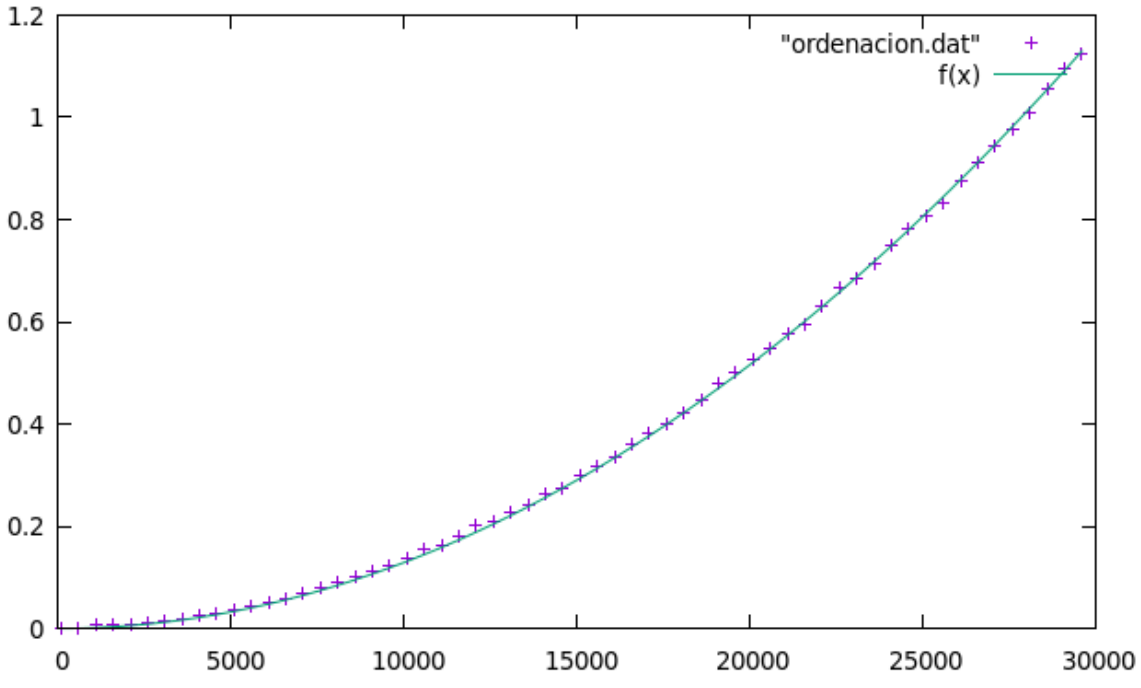
```
#include <iostream>
#include <ctime>    // Recursos para medir tiempos
#include <cstdlib>  // Para generación de números pseudoaleatorios

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;
      }
    }
  }
}

void sintaxis()
{
  cerr << "Sintaxis:" << endl;
```

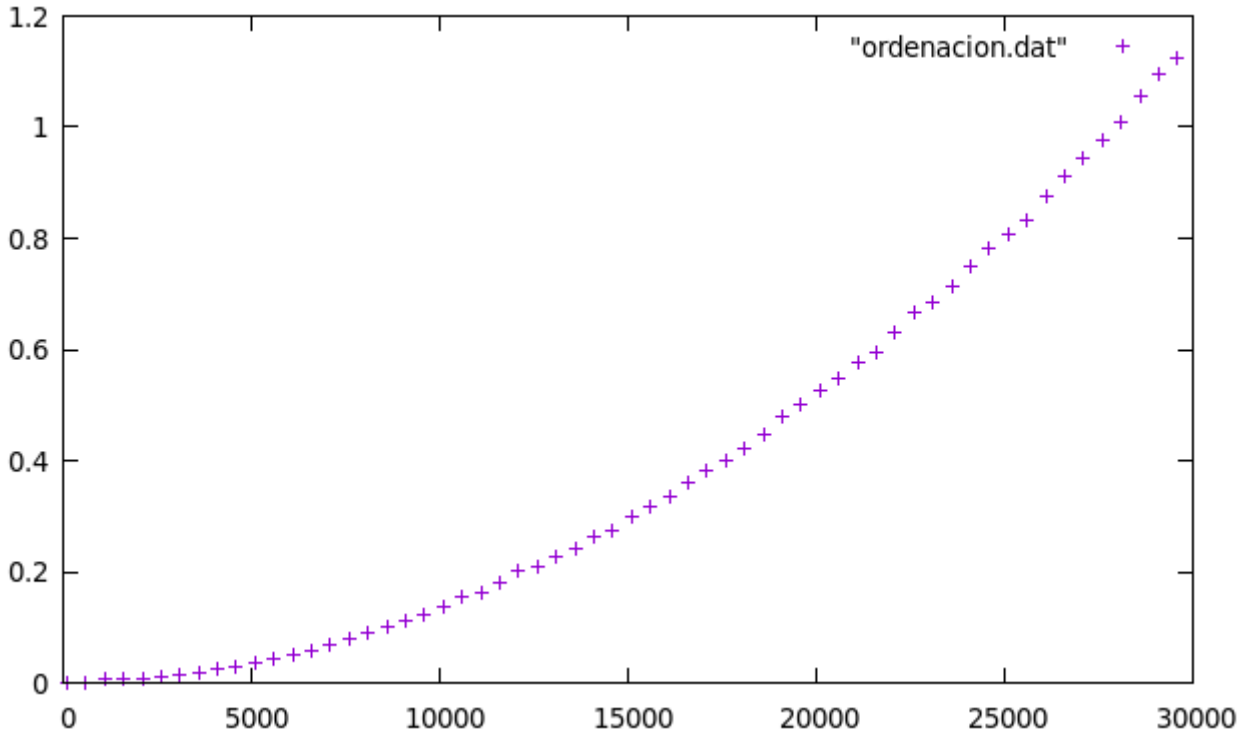
```
  cerr << "  TAM: Tamaño del vector (>0)" << endl;
  cerr << "  VMAX: Valor máximo (>0)" << endl;
  cerr << "Se genera un vector de tamaño TAM con elementos aleatorios en [0,VMAX[" << endl;
  exit(EXIT_FAILURE);
}

int main(int argc, char * argv[])
{
  // Lectura de parámetros
  if (argc!=3)
    sintaxis();
  int tam=atoi(argv[1]);     // Tamaño del vector
  int vmax=atoi(argv[2]);    // Valor máximo
  if (tam<=0 || vmax<=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() % vmax;    // Generar aleatorio [0,vmax[

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

  ordenar(v,tam); // de esta forma forzamos el peor caso

  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
}
```

DATOS

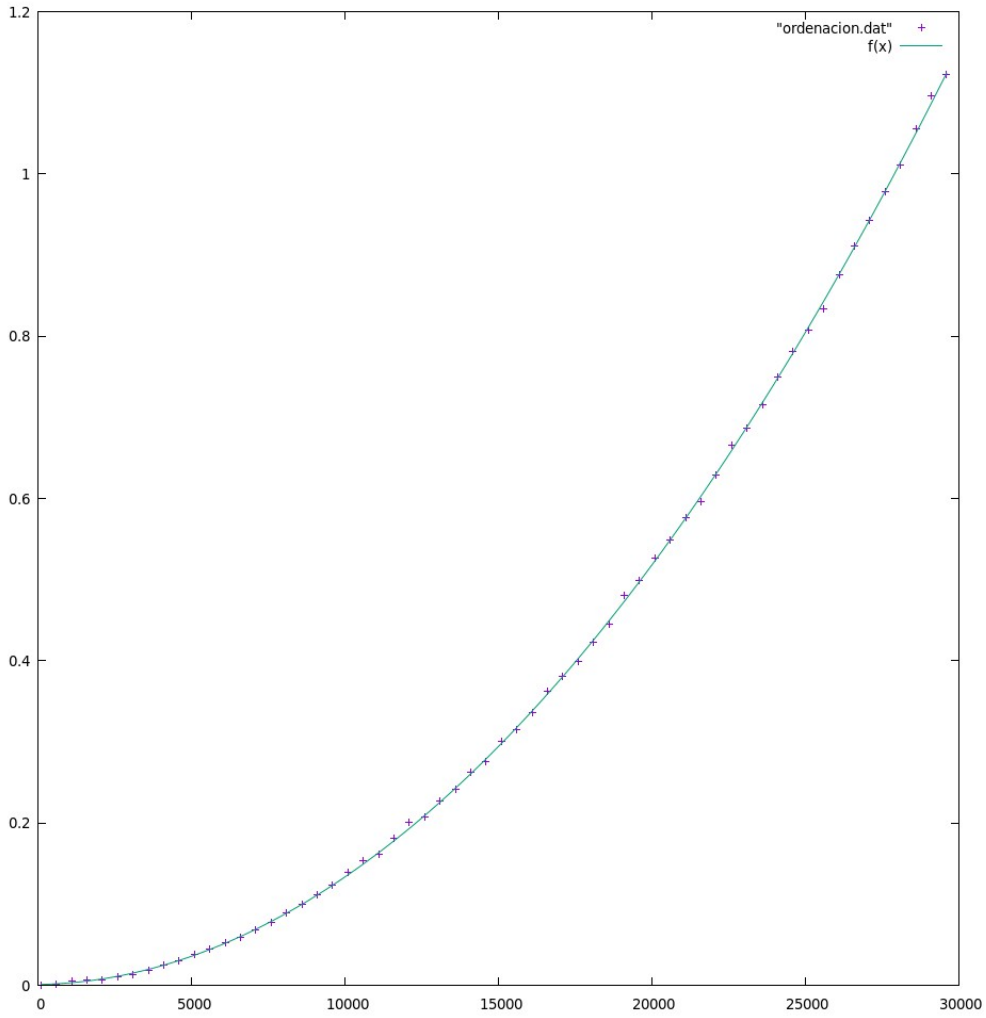
```
100 5.2e-05
600 0.00169
1100 0.005739
1600 0.006731
2100 0.007429
2600 0.009836
3100 0.01306
3600 0.01778
4100 0.025204
4600 0.02801
5100 0.036553
5600 0.042788
6100 0.049466
6600 0.059438
7100 0.070014
7600 0.076408
8100 0.083326
8600 0.099043
9100 0.106274
9600 0.126564
```

10100 0.131966
10600 0.14918
11100 0.172089
11600 0.20429
12100 0.197207
12600 0.246648
13100 0.248869
13600 0.264388
14100 0.294494
14600 0.325278
15100 0.322418
15600 0.370612
16100 0.422496
16600 0.396295
17100 0.414917
17600 0.47933
18100 0.475671
18600 0.480619
19100 0.516724
19600 0.537576
20100 0.590528
20600 0.590703
21100 0.680494
21600 0.691005
22100 0.777984
22600 0.680951
23100 0.701685
23600 0.745958
24100 0.771823
24600 0.852842
25100 0.916062
25600 0.87648
26100 0.933128
26600 0.991966
27100 0.991347
27600 1.02695
28100 1.06927
28600 1.1011
29100 1.13297
29600 1.16885




"ordenacion.dat"
1.2
1
0.8
0.6
0.4
0.2
0
0
5000
10000
15000
20000
25000
30000

EJERCICIO 2


"ordenacion.dat"
f(x)
1.2
1
0.8
0.6
0.4
0.2
0
0
5000
10000
15000
20000
25000
30000

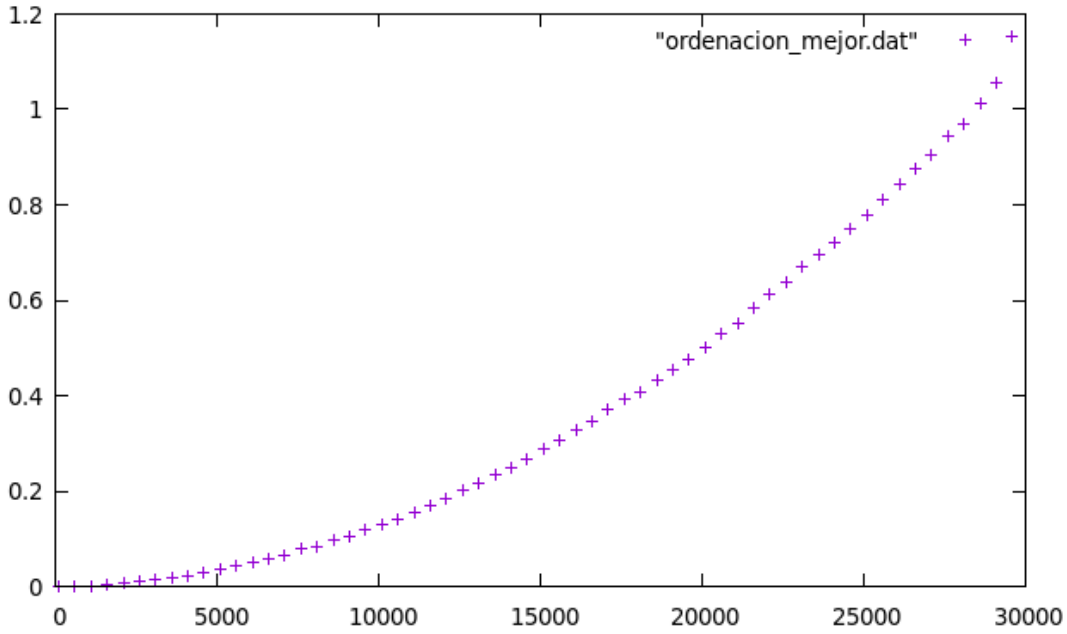
EJERCICIO 3

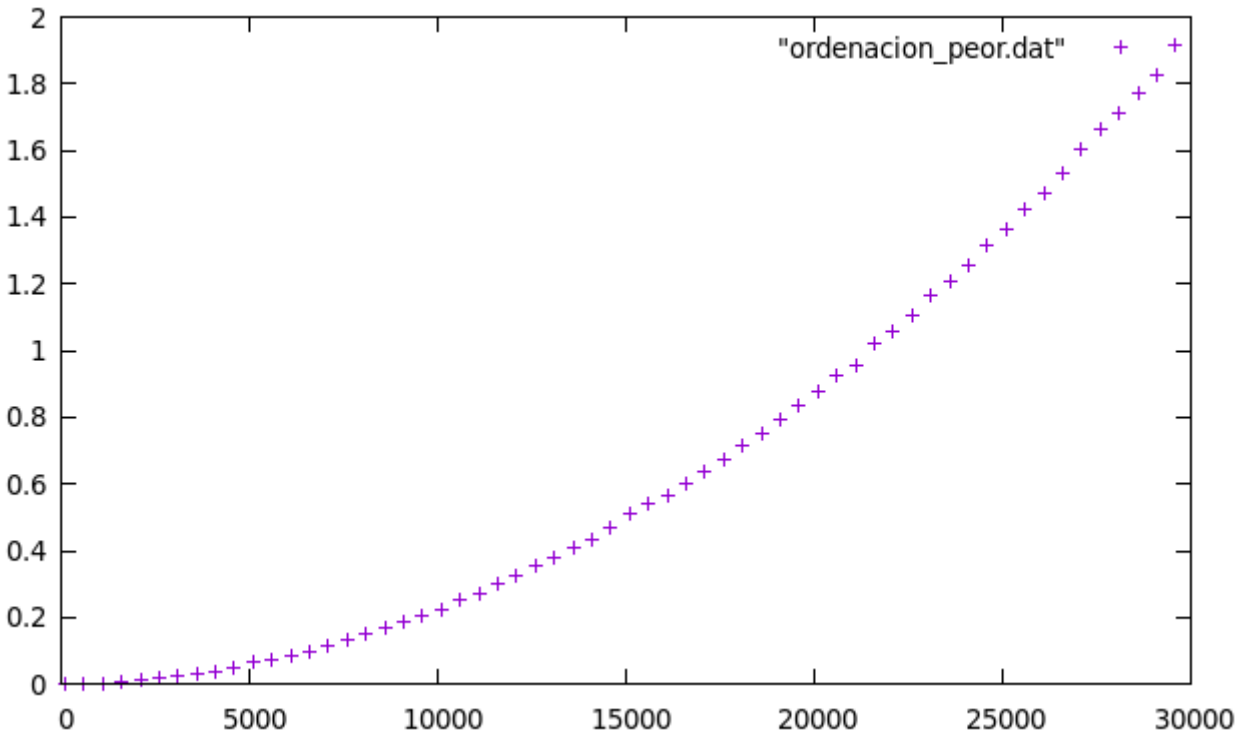
Es un algoritmo de búsqueda ordenada de un elemento, divide el vector en dos y empieza desde la mitad a buscar, si x es mayor que el valor del centro pues sigue a la derecha y si es menor sigue por la izquierda ya que se considera que está ordenado.

Tiene una eficacia teórica de O(log n).

EJERCICIO 4

Los archivos de este ejercicios irán adjuntos al zip de la practica





EJERCICIO 5

La eficiencia teórica sería de O(n), y la eficiencia empírica se muestra en la siguiente gráfica.