Ejercicio 3: Problemas de precisión

Junto con este guión se le ha suministrado un fichero ejercicio_desc.cpp. En él se ha implementado un algoritmo. Se pide que:
• Explique qué hace este algoritmo.
• Calcule su eficiencia teórica.
• Calcule su eficiencia empírica.
Si visualiza la eficiencia empírica debería notar algo anormal. Explíquelo y proponga una solución. Compruebe que su solución es correcta. Una vez resuelto el problema realice la regresión para ajustar la curva teórica a la empírica.

1) Creamos el fichero 'ejercicio_desc.cpp' en el que incluimos tanto el programa completo como el algoritmo de ordenación de la burbuja.

2) Generamos el ejecutable mediante:

g++ ejercicio_desc.cpp –o ejercicio_desc

3) Modificamos los campos del archivo “ejecuciones_desc.csh” con los datos que nos han pedido

4) Para que el resultado de todas las ejecuciones se guarden en el archivo “ejercicio_desc.dat”, ejecutamos lo siguiente:
./ejercicio_desc.csh

5) Calculamos su eficiencia teórica

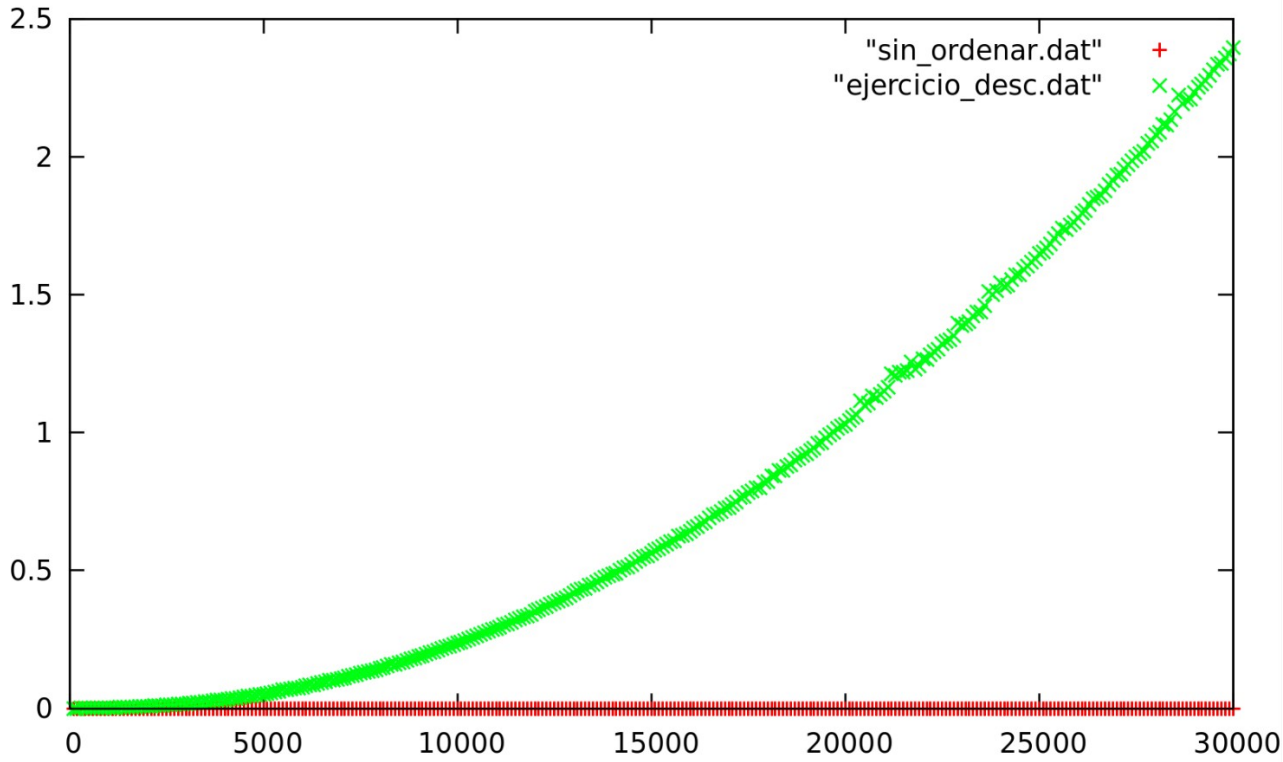
```
int operacion(int *v, int n, int x, int inf, int sup) {
  int med;
  bool enc=false;
  while ((inf<sup) && (!enc)) {
    med = (inf+sup)/2;
    if (v[med]==x)
      enc = true;
    else if (v[med] < x)
      inf = med+1;
    else
      sup = med-1;
  }
  if (enc)
    return med;
  else
    return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

Para calcular la eficiencia del algoritmo, se analizan n elementos. Tras la primera división, el número será como mucho n/2 ya que el vector se divide en dos, tras la siguiente división el número será n/4 y así sucesivamente. El código que hay dentro del while, tiene eficiencia O(1) ya que por la regla de la suma O(max(1,1,1)))=O(1)
Es por esto por lo que el algoritmo de búsqueda binaria tiene una eficiencia de orden logarítmica O(log n).

6) Explicación del algoritmo:

Dicho algoritmo recibe el nombre de búsqueda binaria. En el cual se quiere buscar que posición ocupa un valor en un vector. En el primer while, siempre que se cumpla que el inferior<superior y que no se ha encontrado el valor en el vector, asigna la variable med centro del vector, el cual comprueba si es el valor buscado. En otro caso, si el punto medio es más pequeño que el valor buscado, mueve el inf al punto medio+1 y análogamente ocurre si el punto medio es más grande que el valor buscado

Cabe destacar que el vector tiene que estar previamente ordenado, cosa que no ocurre en el ejemplo por lo que la eficiencia empírica obtenida en el primer intento resulta anormal.
Por esto mismo, hemos introducido el algoritmo de ordenación de burbuja del apartado anterior obteniendo como resultado con gnuplot:



Como podemos observar, el resultado de 'sin_ordenar.dat' no tiene sentido ya que previamente no había sido ordenado, no como ocurre en “ejercicio_desc.dat” en el que el vector está ordenado.

Características del ordenador:

Fabricante	Lenovo
Procesador	Intel Core i7 4710HQ
Sistema Operativo	Ubuntu
Versión SO	14.04
RAM	8192MB
CPU	64 bits