16 DE OCTUBRE DE 2015

ANÁLISIS DE EFICIENCIA DE ALGORITMOS

PRÁCTICA 1 DE ESTRUCTURA DE DATOS

Laura Calle Caraballo
Cristina María Garrido López
Germán González Almagro
Antonio Manuel Milán Jiménez

Contenido

Análisis de "ocurrencias"

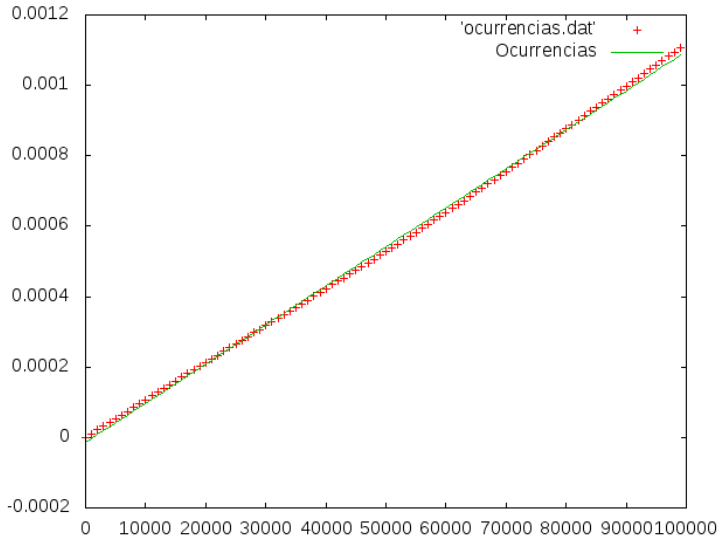
```
1  int contar_hasta( vector<string> & V, int ini, int fin, string & s) {
2    int cuantos = 0;
3    for (int i=ini; i< fin ; i++)
4      if (V[i]==s) {
5        cuantos ++;
6      }
7    return cuantos;
8  }
9
10 //Dentro del main
11
12
13 for (int fin = 10; fin < 100000 ; fin+= 1000){
14  string b="hidalgo";
15
16  start = clock();
17  for (int iteraciones = 0; iteraciones < 1000; iteraciones++)  // Numero de iteraciones se debe ajustar a cada caso
18      pos = contar_hasta(Q, 0,fin, b);
19  end= clock();
20  double dif = end-start;
21  cout << "ciclos " << dif << " seg " << dif/(double) (CLOCKS_PER_SEC * 1000.0) << " tama " << fin <<  endl;
22  }
```

Análisis teórico:

Vemos que el algoritmo es de $O(n)$ puesto que en la función "`contar_hasta`" tenemos un bucle que ejecuta una serie de instrucciones de $O(1)$, *n* veces, determinado por la variable "fin".

Análisis práctico:

Vemos que la gráfica crece de forma lineal, como esperábamos tras el análisis teórico.



$$T(n) = 1.1099 \times 10^{-8}\, n - 1.35849 \times 10^{-5}$$

Análisis de Ordenación por Selección

```
1  void OrdenaSeleccion(vector<string> & v, int fin){
2
3
4      int inicio, indice_elem_menor,i;
5      string elemento_menor,aux;
6
7
8      for(inicio=0; inicio < fin; inicio++){
9
10
11         elemento_menor = v[inicio];
12         indice_elem_menor = inicio;
13
14         for(i=inicio;i<fin;i++){
15
16             if(elemento_menor > v[i]){
17                 elemento_menor = v[i];
18                 indice_elem_menor = i;}
19         }
20
21         aux = v[inicio];
22         v[inicio] = elemento_menor;
23         v[indice_elem_menor] = aux;
24
25     }
26
27 }
```

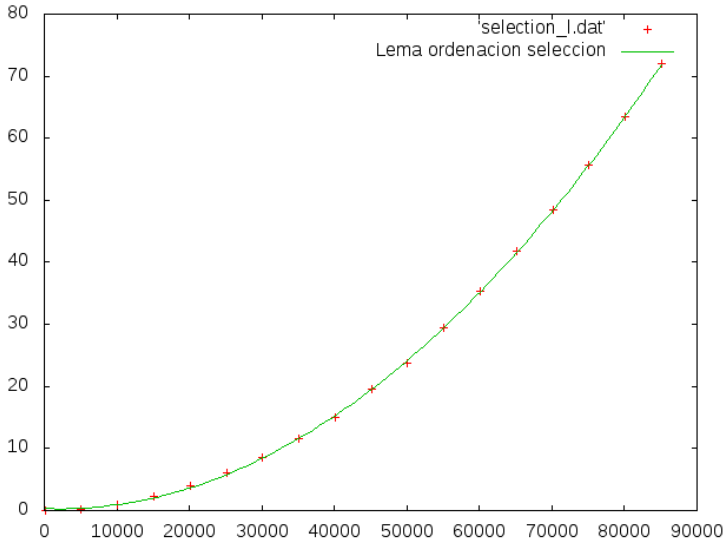
Análisis teórico:

Este algoritmo es de $O(n^2)$ puesto que tenemos un bucle de $O(n)$ ya que, aunque el cuerpo de este bucle es de $O(1)$, el número de ejecuciones depende de la variable "inicio" dada por el bucle externo. El cuerpo de este bucle es de $O(inicio)$, y se ejecutará n veces, con lo cual basta hacer $\sum_{inicio=0}^{n} \text{inicio}$ para comprobar el orden del algoritmo.

Análisis práctico:

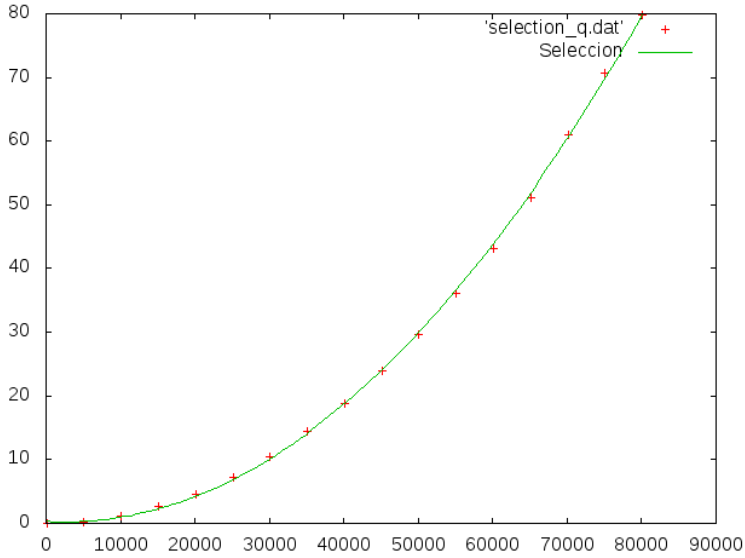
Vemos que la gráfica se corresponde con la función cuadrática que vimos en el análisis teórico. Observamos que el tiempo de ejecución crece exponencialmente en función del volumen de datos. Con cada grafica se muestra la fórmula que corresponde a la función representada.

Gráfica correspondiente a lema.txt



$$T(n) = 1.03896 \times 10^{-8}n^2 - 4.31997 \times 10^{-5}n + 0.322221$$

Gráfica correspondiente a quijote.txt



$$T(n) = 1.33812 \times 10^{-8}n^2 - 8.21155 \times 10^{-5}n + 0.530761$$

Análisis de Ordenación por Inserción

```
1  void insertion(vector<string> & T, int inicial, int final){
2      string aux;
3      bool placed=false;
4
5      for(int i=inicial;i<final;i++){
6          aux=T[i];
7          for (int j=i; j>inicial && !placed ; j--){
8              if(T[j-1]<=T[j])
9                  placed=true;
10             else {
11                 T[j]=T[j-1];
12                 T[j-1]=aux;
13             }
14         }
15     }
16 }
```

Análisis teórico:

Este algoritmo es de $O(n^2)$. Consta de un bucle interno de $O(i)$ con condición de parada, que en el peor caso no se dará. El bucle externo se ejecutará **n** veces, por lo que $\sum_{i=0}^{n} i$ da $O(n^2)$. Si se da la condición de parada del bucle interno, el conjunto de datos está casi ordenado y el orden del algoritmo es $O(n)$.

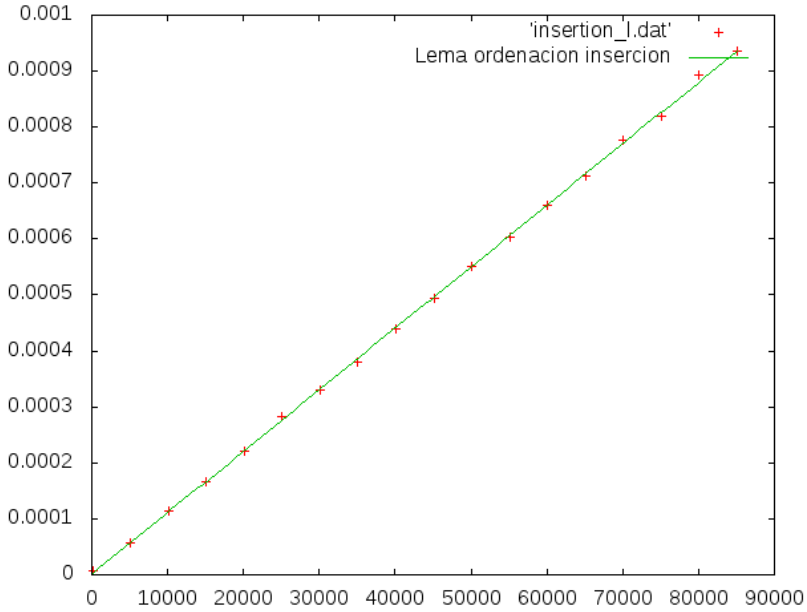
Análisis práctico:

En el caso de lema.txt la fórmula que corresponde con la gráfica no es cuadrática, puesto que para conjuntos de datos casi ordenados el algoritmo de ordenación por inserción es un algoritmo de $O(n)$.

La gráfica que corresponde a quijote.txt presenta una forma cuadrática, en la que el tiempo de ejecución crece exponencialmente en función del volumen de datos, al igual que el algoritmo de Selección.

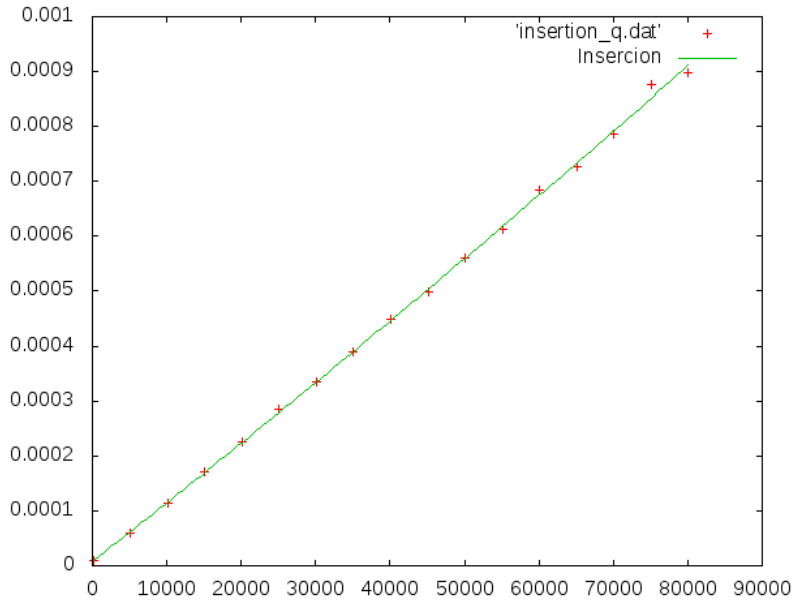
Con cada grafica se muestra la fórmula que corresponde a la función representada.

Gráfica correspondiente a lema.txt



$$T(n) = 1.09889 \times 10^{-8}\, n + 1.15258 \times 10^{-6}$$

Gráfica correspondiente a quijote.txt



$$T(n) = 8.40556 \times 10^{-15} n^2 + 1.06107 \times 10^{-8} n + 7.77579 \times 10^{-6}$$

Análisis de Ordenación Burbuja

```
1  void burbuja(vector<string> & T, int inicial, int final) {
2      int i, j;
3      string aux;
4
5      for (i = inicial; i < final - 1; i++)
6          for (j = final - 1; j > i; j--)
7              if (T[j] < T[j-1]) {
8                  aux = T[j];
9                  T[j]= T[j-1];
10                 T[j-1] = aux;
11             }
12
13 }
```

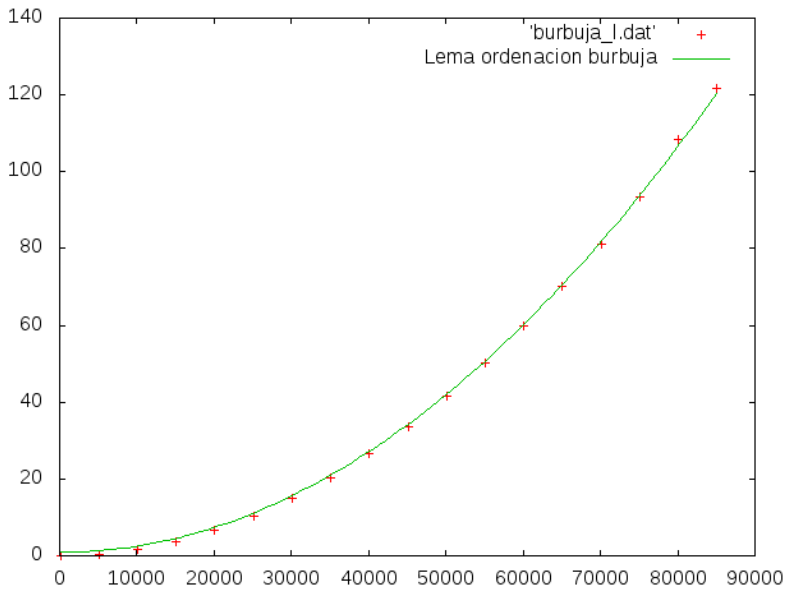
Análisis Teórico:

Consta de dos bucles anidados: el interno es de $O(i)$ ya que el cuerpo del bucle es de orden constante y el numero de iteraciones viene dado por la variable `i` del bucle más externo, el bucle externo es de $O(n^2)$ puesto que $\sum_{i=0}^{n} i$.

Análisis Práctico:

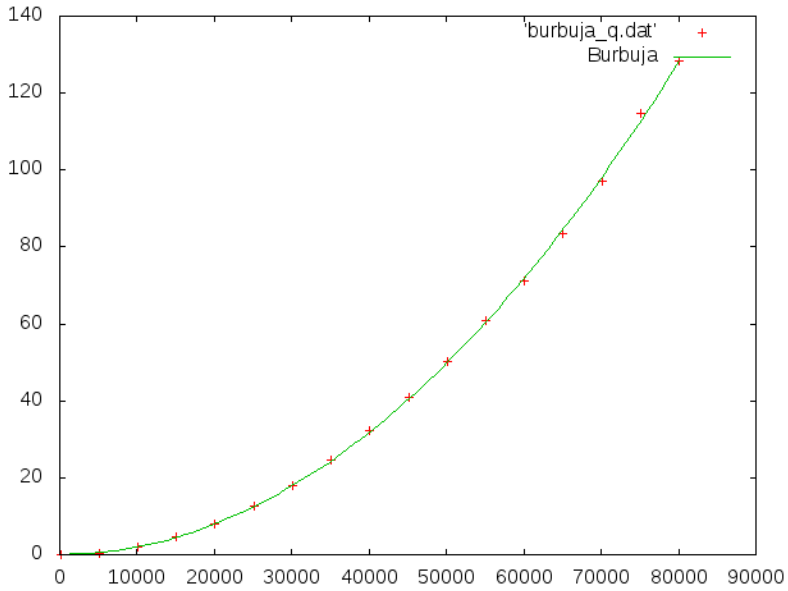
Como podemos ver la gráfica resultante se ajusta a un polinomio de segundo grado con las constantes que se indican.

Gráfica correspondiente a lema.txt



$$T(n) = 1.66495 \times 10^{-8} n^2 + 1.04438 \times 10^{-8}\, n + 7.77579 \times 10^{-6}$$

Gráfica correspondiente a quijote.txt

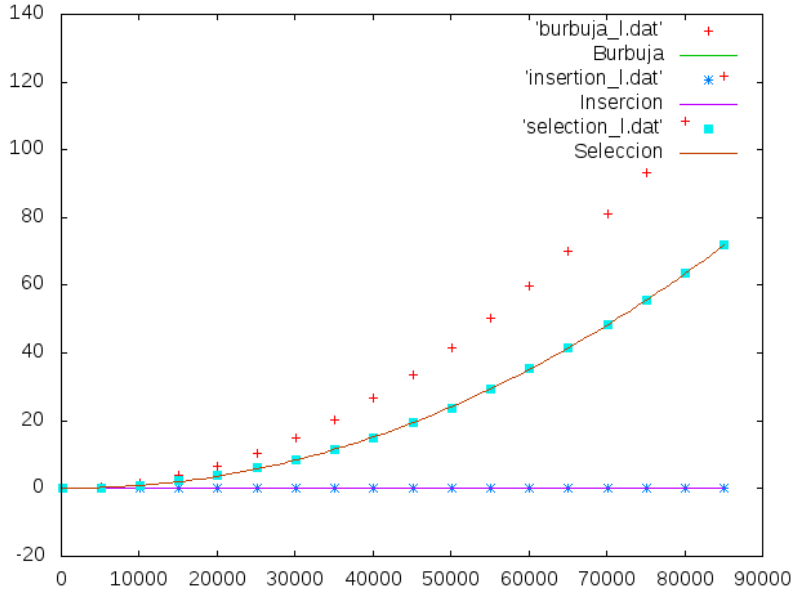


$$T(n) = 2.01497 \times 10^{-8} n^2 - 1.48142 \times 10^{-5} n + 0.147566$$

Comparativa de eficiencia de algoritmos

Pese a que los tres algoritmos son de $O(n^2)$ podemos observar que son mucho más eficientes inserción y selección que burbuja. En el caso en el que el conjunto de datos está casi ordenado podemos comprobar la gran diferencia entre un algoritmo de $O(n^2)$ frente a uno de $O(n)$

Gráfica correspondiente a lema.txt



Gráfica correspondiente a quijote.txt