Análisis de la eficiencia de algoritmos

Norberto Fernández de la Higuera
Javier Gálvez Obispo
Jose Antonio Álvarez Ocete
Yábir G. Benchakhtir

14 de marzo de 2018

Doble Grado en Ingeniería Informática y Matemáticas

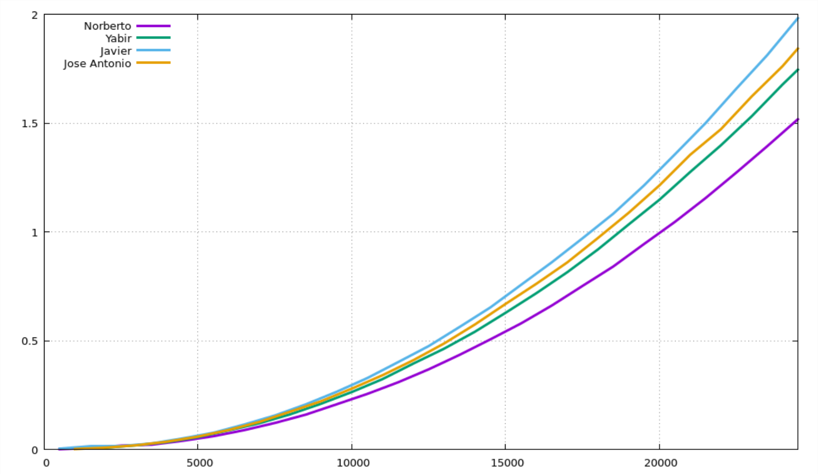
Algoritmos de ordenación $O(n^2)$

- Burbuja
- Selección
- Inserción

Algoritmo burbuja

El análisis teórico de este algoritmo nos dice que este algoritmo es de la forma:

$$\frac{a}{2}n^2 - \frac{3a}{2}n + a \in O(n^2)$$


Norberto
Yabir
Javier
Jose Antonio
2
1.5
1
0.5
0
0
5000
10000
15000
20000

Algoritmo de selección

```
static void seleccion_lims(int T[], int inicial, int
    final) {
        int i, j, indice_menor;
        int menor, aux;
        for (i = inicial; i < final - 1; i++) {
                indice_menor = i;
                menor = T[i];
                for (j = i; j < final; j++) {
                        if (T[j] < menor) {
                                indice_menor = j;
                                menor = T[j];
                        }
                }
                aux = T[i];
                T[i] = T[indice_menor];
                T[indice_menor] = aux;
        };
}
```

Algoritmo de selección

El análisis teórico de este algoritmo nos dice que este algoritmo es de la forma:

$$\begin{aligned}
\sum_{i=0}^{n-1}\left[a_2+\sum_{j=i}^{n} a_1\right] &= \sum_{i=0}^{n-1}\left[a_2+a_1\sum_{j=i}^{n} 1\right] \\
&= \sum_{i=0}^{n-1} a_2 + a_1 \sum_{i=0}^{n}\sum_{j=i}^{n-1} 1 \\
&= a_2\sum_{i=0}^{n-1} 1 + a_1\sum_{i=0}^{n}(n-i)
\end{aligned} \tag{1}$$

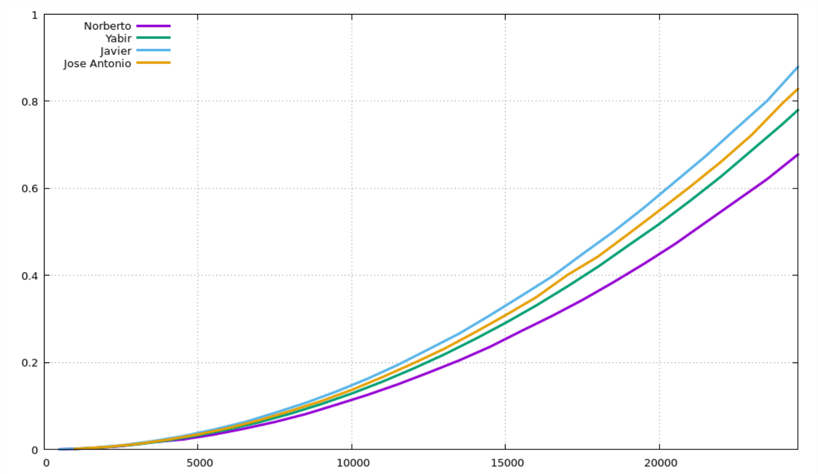
Algoritmo de selección

$$
\begin{aligned}
a\sum_{i=0}^{n-1}(n-i) &= a_1\left[\sum_{i=0}^{n} n - \sum_{j=i}^{n} i\right] \\
&= an^2 - a\frac{(n+1)n}{2} \\
&= an^2 - \frac{an^2}{2} - \frac{an}{2}
\end{aligned} \tag{2}
$$

Algoritmo de selección

Finalmente

$$\frac{an^2}{2} - \frac{an}{2} + b(n-1) = \frac{a}{2}n^2 + \frac{b-a}{2}n - b \in O(n^2)$$

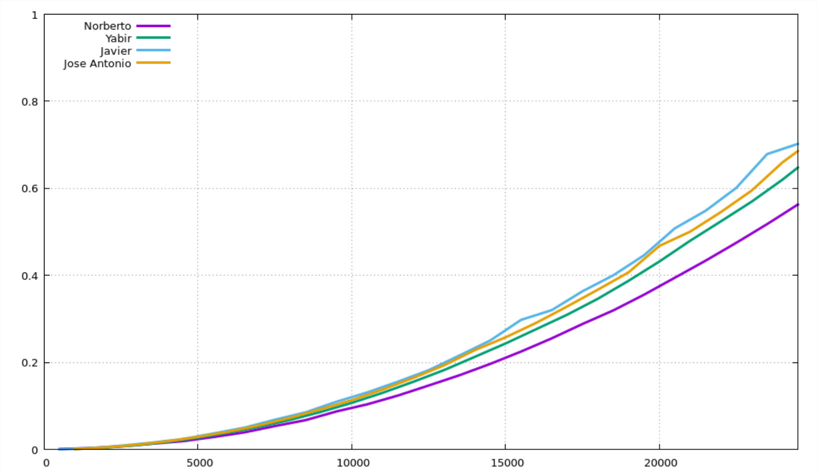

Norberto
Yabir
Javier
Jose Antonio
1
0.8
0.6
0.4
0.2
0
0
5000
10000
15000
20000

```
static void insercion_lims(int T[], int inicial, int
     final)
{
        int i, j;
        int aux;
        for (i = inicial + 1; i < final; i++) {
                j = i;
                while ((T[j] < T[j-1]) && (j > 0)) {
                        aux = T[j];
                        T[j] = T[j-1];
                        T[j-1] = aux;
                        j--;
                };
        };
}
```

Algoritmo de inserción

Aplicando un análisis semejante a los anteriores obtenemos:

$$\sum_{i=0}^{n}\sum_{j=i}^{n} a = ... = a \cdot (\frac{n^2}{2} + 2n) \in O(n^2)$$


Norberto
Yabir
Javier
Jose Antonio
1
0.8
0.6
0.4
0.2
0
0
5000
10000
15000
20000

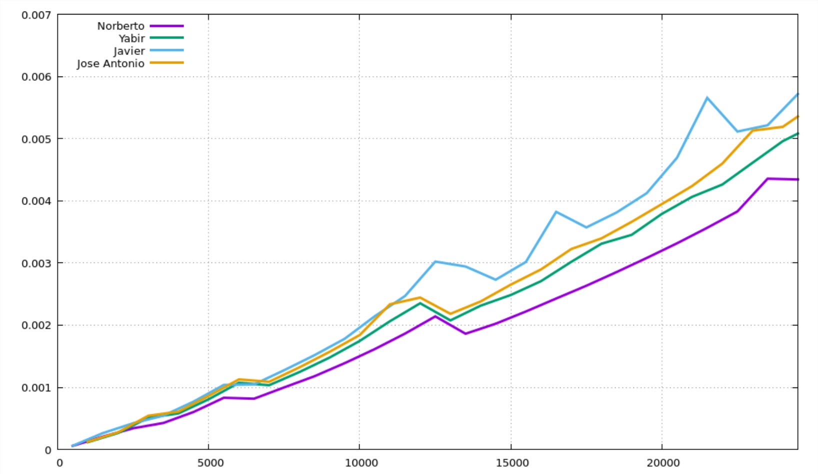
Algoritmos de ordenación $O(nlogn)$

- mergesort
- heapsort
- quicksort

Algoritmo mergesort

Como se ha visto en teoría la eficiencia de este algoritmo es:

$$c_1 n + c_2 n log_2(n) \in O(n log_2(n))$$



Norberto
Yabir
Javier
Jose Antonio
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
0
5000
10000
15000
20000

Algoritmo heapsort

```
    static void reajustar(int T[], int num_elem, int
         k) {
        int j;
        int v;
        v = T[k];
        bool esAPO = false;
        while ((k < num_elem/2) && !esAPO) {
                j = k + k + 1;
                if ((j < (num_elem - 1)) && (T[j] <
                    T[j+1]))
                        j++;
                if (v >= T[j])
                        esAPO = true;
                T[k] = T[j];
                k = j;
        }
        T[k] = v;
}
```

Acotamos el interior del bucle por una constance c_1 y $t = n - k$

$$R(n-k) = R(\frac{n-k}{2}) + c_1 \leftrightarrow R(t) = R(\frac{t}{2}) + c_1$$

Aplicando el cambio de variable $t = 2^m \leftrightarrow log_2 t = m$:

$$R(2^m) = R(2^{m-1}) + c_1$$

Denotando $R(2^m) = X_m$ obtenemos una ecuación en recurrencia:

$$X_m = X_{m-1} + c_1$$

Cuya solución es:

$$X_m = c_2 + c_1 * m$$

Donde c_2 es otra constance. Deshaciendo el cambio de variable obtenemos finalmente:

$$R(2^m) = R(t) = c_2 + c_1 * log_2(t) \in O(log(n))$$

A partir de la eficiencia de la función **Reajustar** es relativamente sencillo obtener la del algoritmo completo:

$$\sum_{i=0}^{n/2} R(i) + \sum_{i=1}^{n-1}(R(i) + c_3)$$

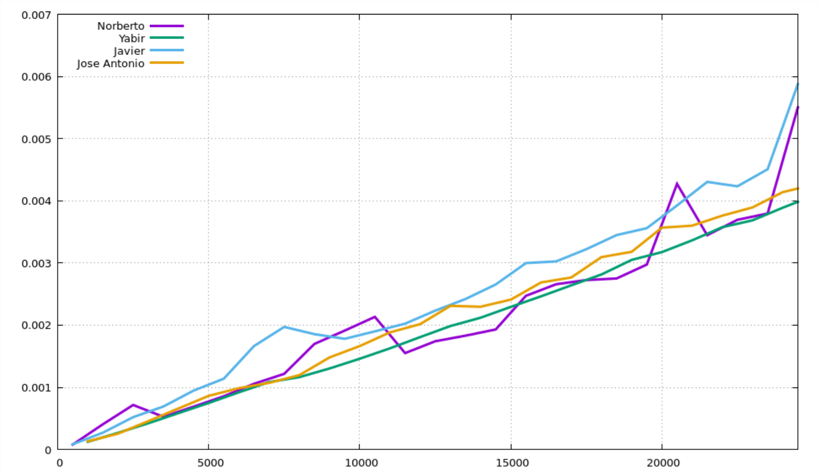
Como el logarítmo es una función creciente, acotamos los valores interiores de los bucles por el mayor valor alcanzado:

$$\sum_{i=0}^{n/2} R(i) + \sum_{i=1}^{n-1}(R(i) + c_3) \leq \sum_{i=0}^{n/2} R(n/2) + \sum_{i=1}^{n-1}(R(n-1) + c_3) =$$

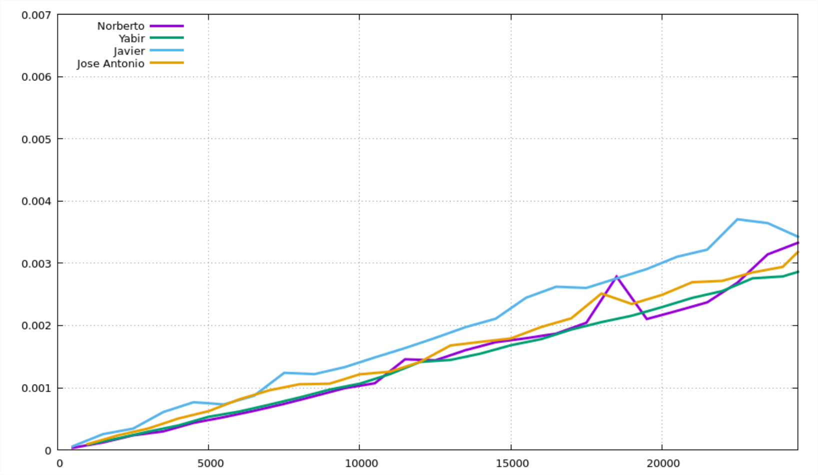
$$= (n/2) \cdot R(n/2+1) + (n-1) \cdot R(n-1) + c_3(n-1) =$$

$$= (n/2) \cdot R(n/2+1) + (n-1) \cdot R(n-1) + c_3(n-1)$$

Como el orden de eficiencia de **Reajustar** es de $O(log(n))$, podemos concluir que el algoritmo tiene una eficiencia de $O(nlog(n))$


Norberto
Yabir
Javier
Jose Antonio
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
0
5000
10000
15000
20000

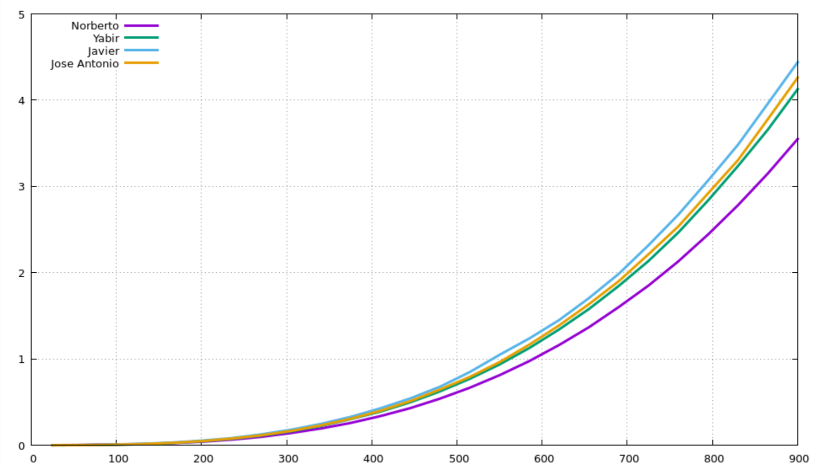
Algoritmo quicksort



Algoritmo Floyd

```
void Floyd(int **M, int dim) {
  for (int k = 0; k < dim; k++)
  for (int i = 0; i < dim;i++)
  for (int j = 0; j < dim;j++){
    int sum = M[i][k] + M[k][j];
    M[i][j] = (M[i][j] > sum) ? sum : M[i][j];
  }
}
```

$$\sum_{k=0}^{n}\sum_{i=0}^{n}\sum_{j=0}^{n} a = an^3 \in O(n^3)$$


Norberto
Yabir
Javier
Jose Antonio
0
1
2
3
4
5
0
100
200
300
400
500
600
700
800
900

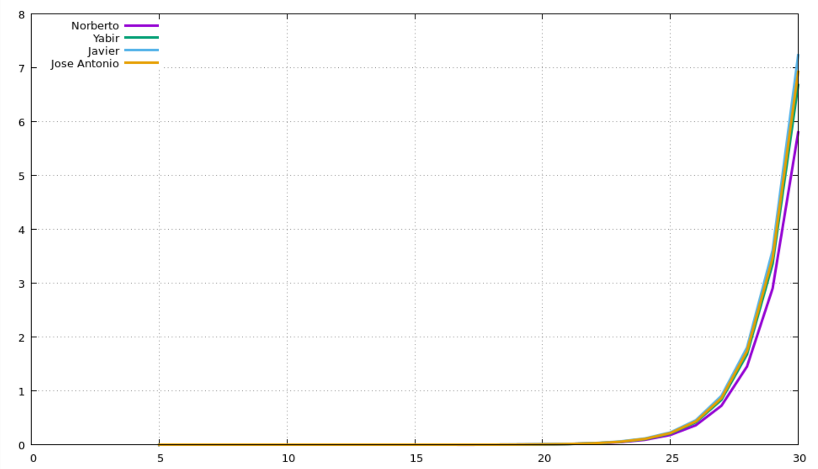
Algoritmo Hanoi

```
void hanoi (int M, int i, int j) {
        if (M > 0) {
                hanoi(M-1, i, 6-i-j);
                hanoi (M-1, 6-i-j, j);
        }
}
```

$$Hanoi(n) = 2 \cdot Hanoi(n-1)$$

$$X_{n+1} = 2 \cdot X_n$$

$$X_n = C \cdot 2^n \in O(2^n)$$



Norberto
Yabir
Javier
Jose Antonio
0
1
2
3
4
5
6
7
8
0
5
10
15
20
25
30