GRUPO AAD_5

Adrián Moreno Monterde
Antonio José Morano Moriña
Antonio Brimes Romero

1.1 Ordenación por inserción : Resumen

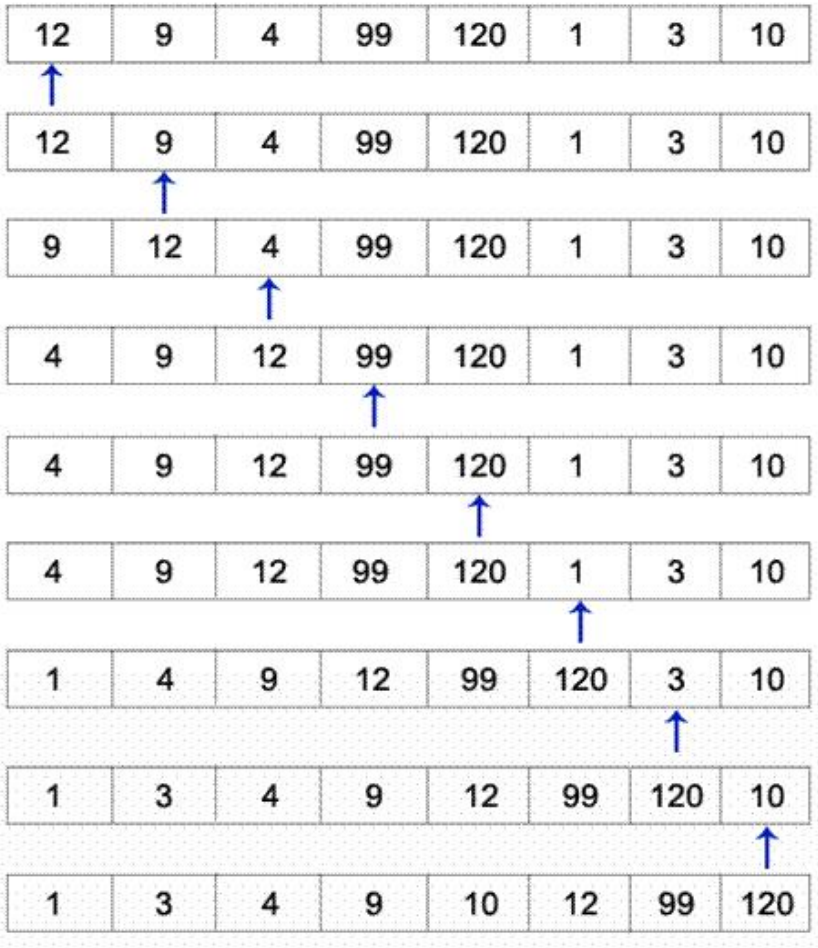
La ordenación por inserción consiste en tomar elemento a elemento e ir insertando cada elemento en su posición correcta de manera que se mantiene el orden de los elementos ya ordenados.

Los pasos para la ordenación son los siguientes:

1. Suponemos el primer elemento ordenado.
2. Desde el segundo hasta el último elemento, hacer:
    1. Suponer ordenados los (i-1) primeros elementos
    2. Tomar el elemento i
    3. Buscar su posición correcta
    4. Insertar dicho elemento, obteniendo i elementos ordenados

1.2. Algoritmo Inserción: Pseudocódigo

```
procedimiento inserción(T[1..n])
    para i← 2 hasta n hacer
        x ← T[i]
        j ← i-1
        mientras j>0 AND x < T[j] hacer
            T[j+1] ← T[j]
            j ← j-1
        fmientras
        T[j+1] ← x
    fpara
fprocedimiento
```



1.3. Análisis. Inserción

```
procedimiento inserción(T[1..n])
    para i← 2 hasta n hacer
        x ← T[i]
        j ← i-1
        mientras j>0 AND x < T[j]
hacer
            T[j+1] ← T[j]
            j ← j-1
        fmientras
        T[j+1] ← x
    fpara
fprocedimiento
```

- Caso mejor: cuando el vector está ordenado, sólo hace una comparación en cada paso. Por tanto T(n) = O(n).

  Ej: 15 20 45 60 n = 4

- Caso Peor: Cuando el vector está ordenado inversamente. Por tanto T(n) = O(n2).

  Ej: 86 52 45 20 n = 4

- Caso medio: Los elementos aparecen de forma aleatoria. Por tanto T(n) = O(n2).

1.4. Traza. Inserción.

6 5 3 1 8 7 2 4

Se sitúa en la posición v[2], suponiendo ordenado v[1].

Compara desde v[2] hasta v[n], insertando el elemento en su lugar indicado.

Por ejemplo:

Como v[3] < v[2] y v[3] < v[1], V[3] pasaría a la primera posición, desplazando el resto a la derecha.

Siempre se compara la posición i, con i-1.

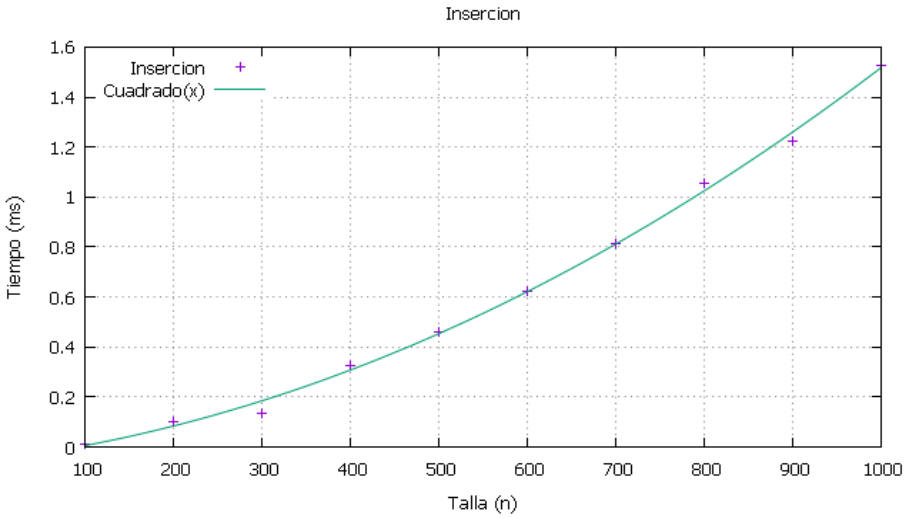
1.5. Codificación del algoritmo de ordenación: Inserción

```
void AlgoritmosOrdenacion::ordenaInsercion (int v[], int size){

    int x, j;

    for (int i=1; i<size; i++){
        x = v[i];
        j = i-1;
        while(j>=0 && x<v[j]){
            v[j+1] = v[j];
            j = j-1;
        }
        v[j+1] = x;
    }
}
```

1.6. Gráfica de coste: Inserción




```
Tiempos medios para el algoritmo de Ordenacion por Insercion

        Talla           Tiempo (ns)

        1000            1062.4
        2000            2211.2
        3000            5027.2
        4000            9058.1
        5000            13820
        6000            20627.6
        7000            34258.1
        8000            38119.5
        9000            45684.3
        10000           56869.2
```

2.1. Ordenación por Shell

Resumen del algoritmo:

Éste algoritmo supone una mejora del método de **Inserción**, es algo más rápido y se usa para vectores de mayor tamaño.

Va colocando cada elemento en su posición correcta desplazando el resto de elementos mayores que el seleccionado a la derecha.
A diferencia de Inserción, en el método Shell no comparamos cada elemento con su adyacente sino que se selecciona una distancia "**h**" (que inicialmente es la mitad de la talla) y se ordena comparando cada elemento con el que se encuentre a una distancia "**h**" de él.

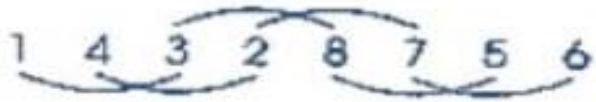
Ésta distancia irá disminuyendo hasta que al llegar a 1, todos los elementos deberían estar ordenados.

2.2. Pseudocódigo: Shell

```
procedimiento shell(T[1...n){
   para h= n/2 hasta 0 hacer (decremento h = h/2)
      para i=h hasta n hacer
         j ← i
         V ← T[i]
         mientras j>=h and v < T[j-h] hacer
            T[j] = T[j-h];
            j = j - h;
         fmientras
         T[j]=v;
      fpara
   fpara
fprocedimiento
```

2.3. Ejemplo de ejecución / Traza: Shell

Salto: 4
Intercambios: 8 ↔ 1
7 ↔ 4
5 ↔ 3



Salto: 2
Intercambios: 4 ↔ 2
8 ↔ 5
7 ↔ 6

Salto: 1
Intercambios: 8 ↔ 7

Ordenado

2.4. Codificación: Shell

```
template <typename elem>
void shell_sort(vector<elem>& T){
   int n= T.size();
   for(int h=n/2; h>0; h/=2){
      for(int i=h; i<n; i++){
         int j=i;
         elem v = T[i];
         while(j>=h and v<T[j-h]){
            T[j]=T[j-h];
            j-=h;
         }
         T[j]=v;
      }
   }
}
```

2.5. Eficiencia: Shell

Análisis de eficiencia:

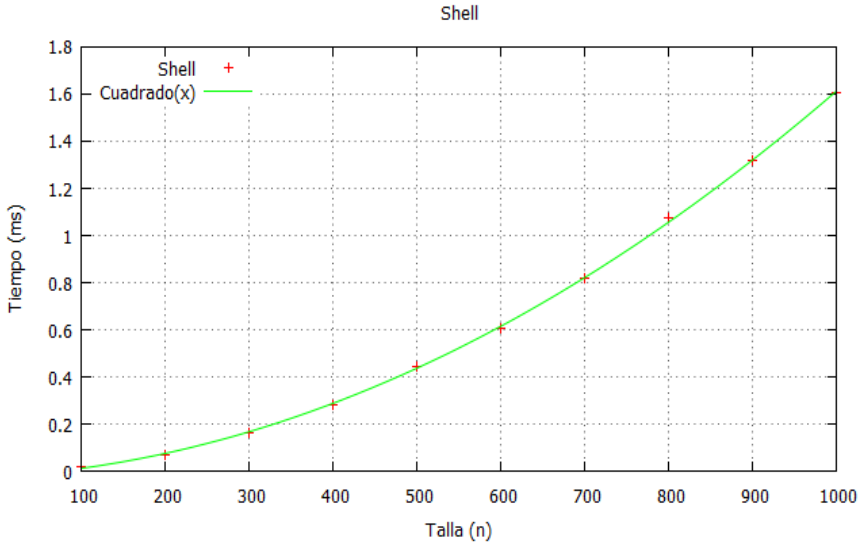
Aunque es fácil desarrollar un sentido intuitivo de cómo funciona este algoritmo, es muy difícil analizar su tiempo de ejecución. Dependiendo de la elección de la secuencia de espacios, Shellsort tiene un tiempo de ejecución en el **Caso Peor** de $\mathbf{O(n^2)}$.

-Usando los incrementos de Shell que comienzan con 1/2 del tamaño del vector y se dividen por 2 cada vez, $\mathbf{O(n^{3/2})}$
-Usando los incrementos de Hibbard de $2^k - 1$, $\mathbf{O(n^{4/3})}$
-Usando los incrementos de Sedgewick , $\mathbf{O(nlog^2n)}$

Posibles mejoras:

Este algoritmo ya de por sí es una mejora del método de ordenación por Inserción, no obstante, como hemos podido comprobar en el estudio de su complejidad, se puede reducir hasta $\mathbf{O(nlog^2n)}$ si usamos los incrementos de Sedgewick.

2.6. Gráfica: Shell



Tiempos medios para el algoritmo de Ordenacion Shell

Talla	Tiempo (ns)
1000	105.2
2000	233.5
3000	381.9
4000	522.7
5000	683
6000	836.8
7000	981.9
8000	1166.5
9000	1314.2
10000	1487

3.1. Ordenación rápida (Quicksort).

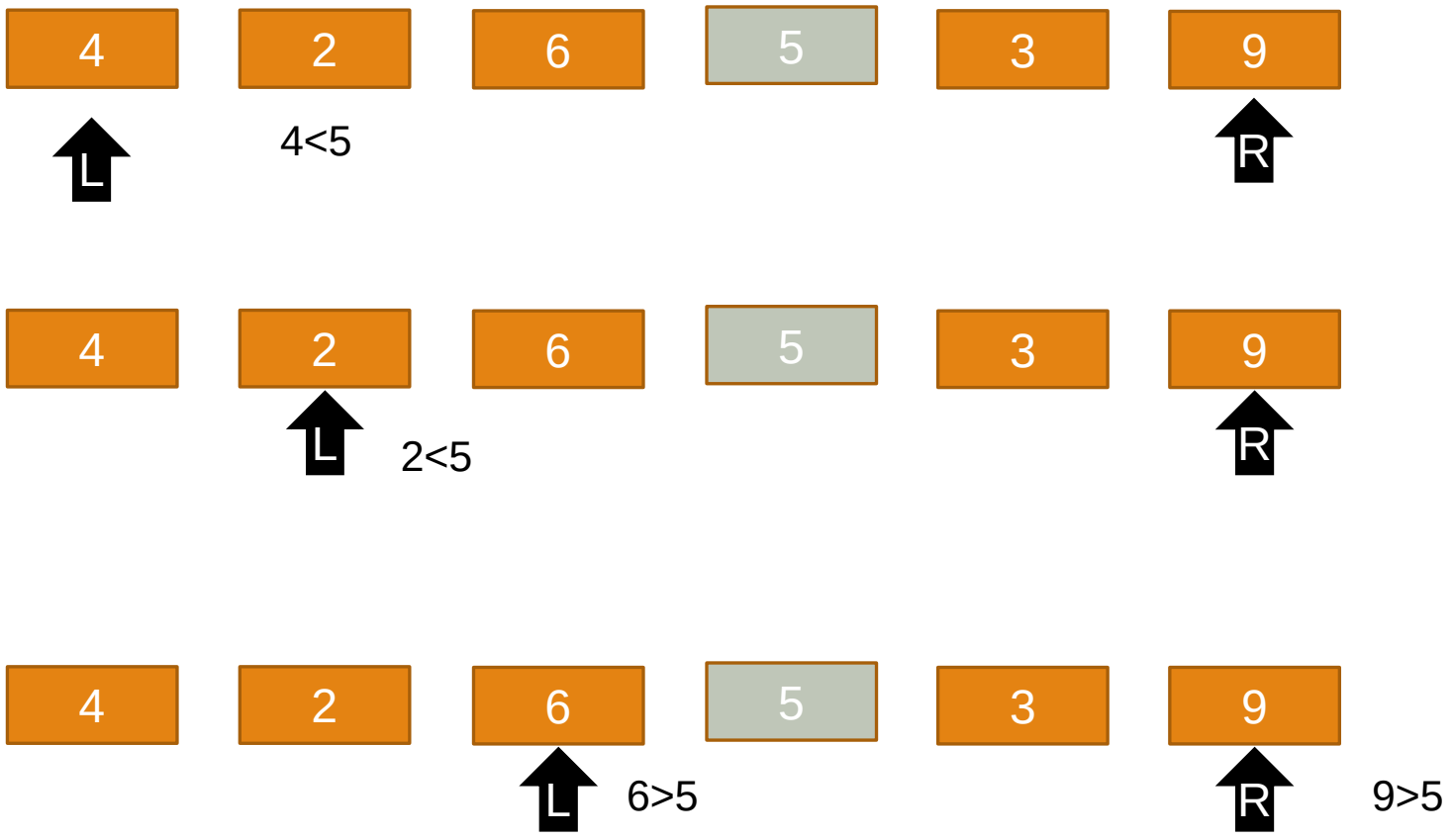
- Se basa en la estrategia “dividir para vencer”.
- Consiste en dividir los datos de la lista a ordenar en particiones separadas por un elemento al que llamaremos “pivote”.

3.2. Traza (Quicksort).

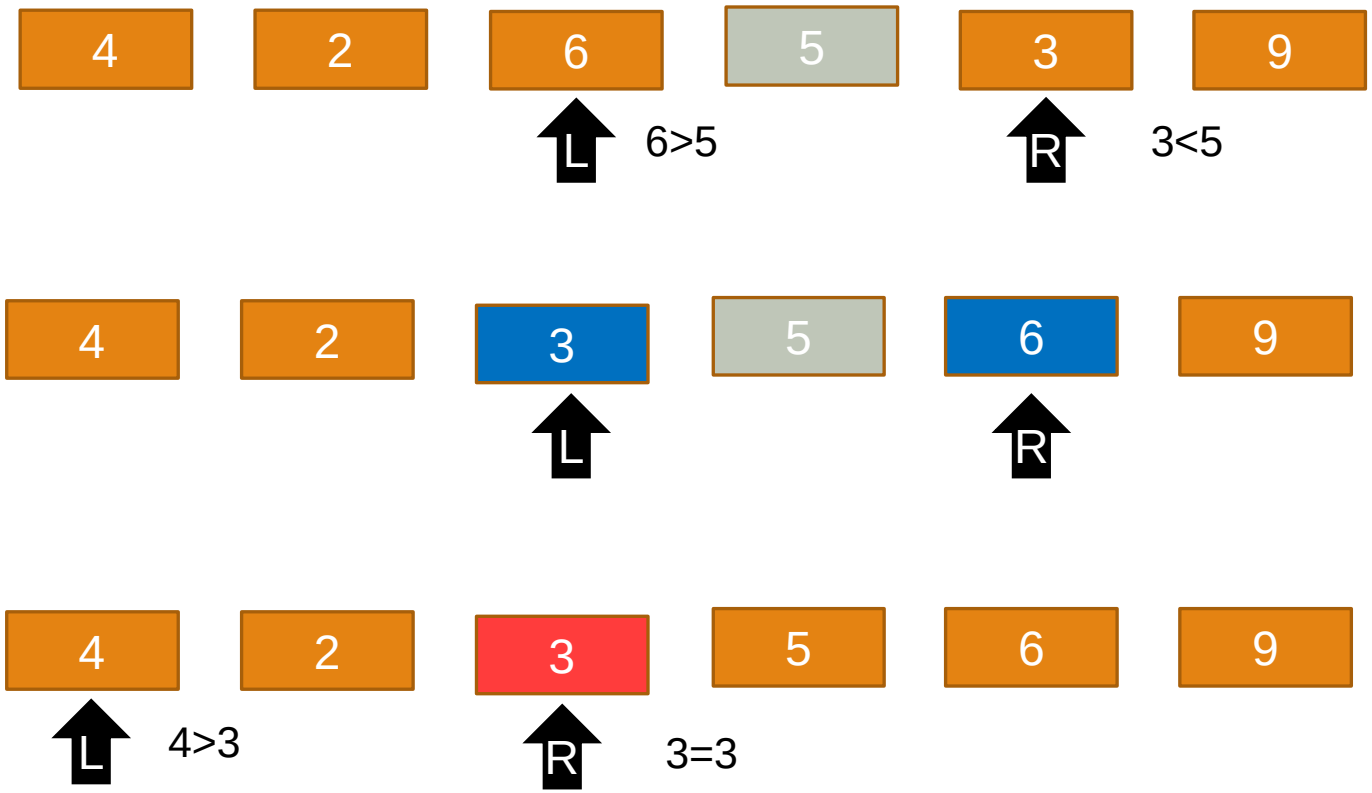
1. Determinamos el pivote

2. Ahora hay que mover todos los elemento menores que el pivote a su izquierda y los mayores a su derecha

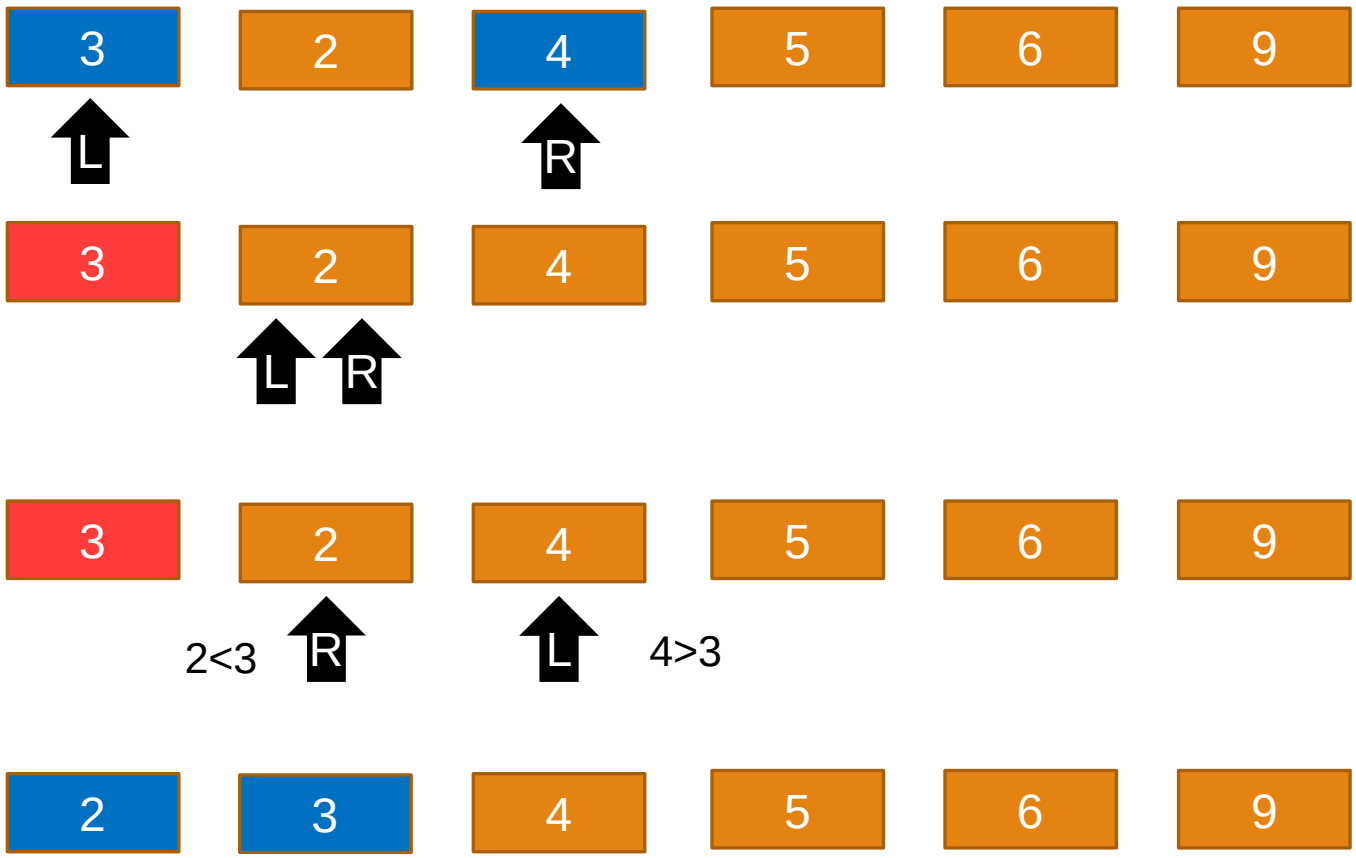
3.2. Traza (Quicksort).



3.2. Traza (Quicksort).



3.2. Traza (Quicksort).



3.3. Pseudocódigo (Quicksort).

```
Procedimiento Quicksort (T[1…n], e, d)
        i <- e
        j <- d
        x<- T [(e+d/2)]
        mientras i =< j hacer
                mientras T(j) < x hacer
                        i <- i+1
                fmientras
                mientras T(j) > x hacer
                        j <- j-1
                fmientras
                si i =< j entonces
                        aux <- T(i)
                        T(i) <- T(j)
                        T(j) <- aux
                        i = i + 1
                        j = j – 1
                fsi
        fmientras
        si e < j
                llamar Quicksort (T[1…n], e, d)
        fsi
        si i < j
                llamar Quicksort (T[1…n], e, d)
        fsi
fprocedimiento
```

3.4. Código (Quicksort).

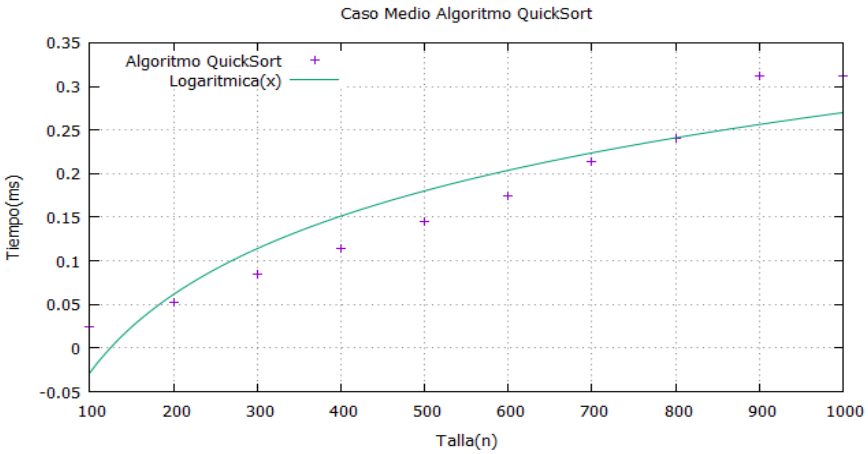
```
template <typename elem>
void quick_sort(vector<elem>& T)
{
      quick_sort(T, 0, T.size() - 1);
}
template <typename elem>
void quick_sort(vector<elem>& T, int e, int d)
{
      if (e < d)
      {
            int q = partition(T, e, d);
            quick_sort(T, e, q);
            quick_sort(T, q + 1, d);
      }
}
template <typename elem>
int partition(vector<elem>& T, int e, int d)
{
      elem x = T[e]; int i = e - 1; int j = d + 1;
      for (;;)
      {
            while (X < T[--j]);
            while (T[++i] < x);
            if (i >= j) return j;
            swap(T[i], T[j]);
      }
}
```

3.5. Eficiencia (Quicksort).

La eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

- Caso Mejor: El pivote acaba en el centro de la lista dividiéndolas en dos sublistas de igual tamaño. T(n) = O(n log n)
- Caso Peor: El pivote acaba en un extremo de la lista. T = O(n^2)
- Caso medio: T(n) = O(n log n)

3.6. Gráfica de Coste. QUICKSORT




```
Tiempos medios para el algoritmo de Ordenacion Rapida ( Quicksort )

       Talla           Tiempo (ns)

       1000            305.8
       2000            642.7
       3000            1003.8
       4000            1383.1
       5000            1784.5
       6000            2172
       7000            2589.9
       8000            3004.9
       9000            3450.3
       10000           3962.1
```

4. Bibliografía.

- R. Guerequeta, A. Vallecillo. Técnicas de Diseño de Algoritmos. Servicio de Publicaciones de la Universidad de Málaga.1998. Segunda Edición: Mayo 2000.
- Tema 5 y 6 de la asignatura.
- https://elbauldelprogramador.com/algoritmos-de-ordenacion/