

Análisis y diseño de Algoritmos

2do Parcial

Heap Sort, Quick Sort, Counting Sort

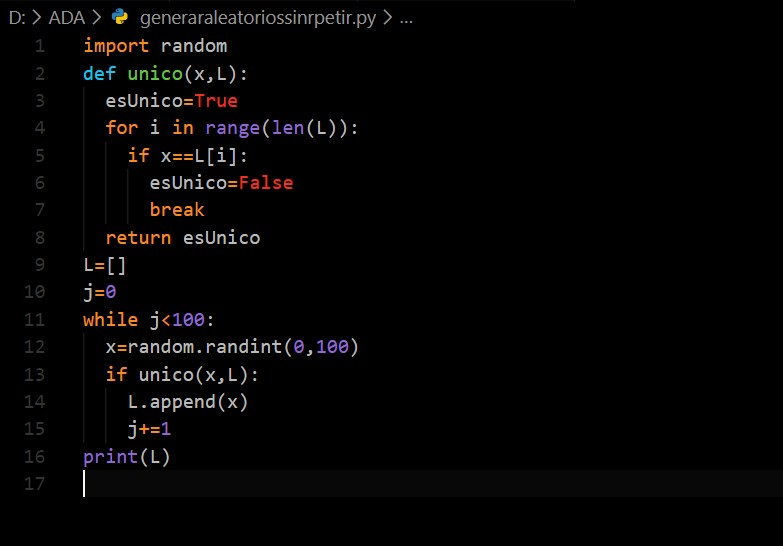
Maestro: Dr. Rodrigo Soule De Castro

Alumno: Daniel Alejandro Morales Castillo

Fecha: 26/abril/2020

Matricula: 31122738

Primeramente se creo un código que genera arreglos de diferentes tamaños y números sin repetirse. Nos servirá para probar los algoritmos en código.



***Heap Sort***

Es un método de ordenamiento basado con comparación, usa el Montículo o Heap como estructura de datos. Este método es más lento que otros métodos, pero es más eficaz en escenarios más rigurosos.

Se define como un método no recursivo, no estable y con complejidad Computacional.

### Funcionamiento:

Este algoritmo consiste en almacenar todos los elementos en un montículo y luego extraer el nodo que queda como raíz en iteraciones sucesivas obteniendo el conjunto ordenado. Para esto el método realiza los siguientes pasos:

1. Se construye el Heap/montículo a partir del arreglo original.

2. La raíz se coloca en el arreglo.

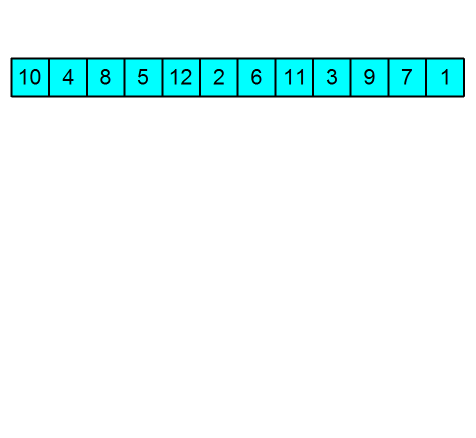
3. El último elemento del montículo se vuelve la raíz.

4. La nueva raíz se intercambia con el elemento de mayor valor de cada nivel.

5. Tras el paso anterior la raíz vuelve a ser el mayor del montículo.

6. Se repite el paso 2 hasta que quede el arreglo ordenado.

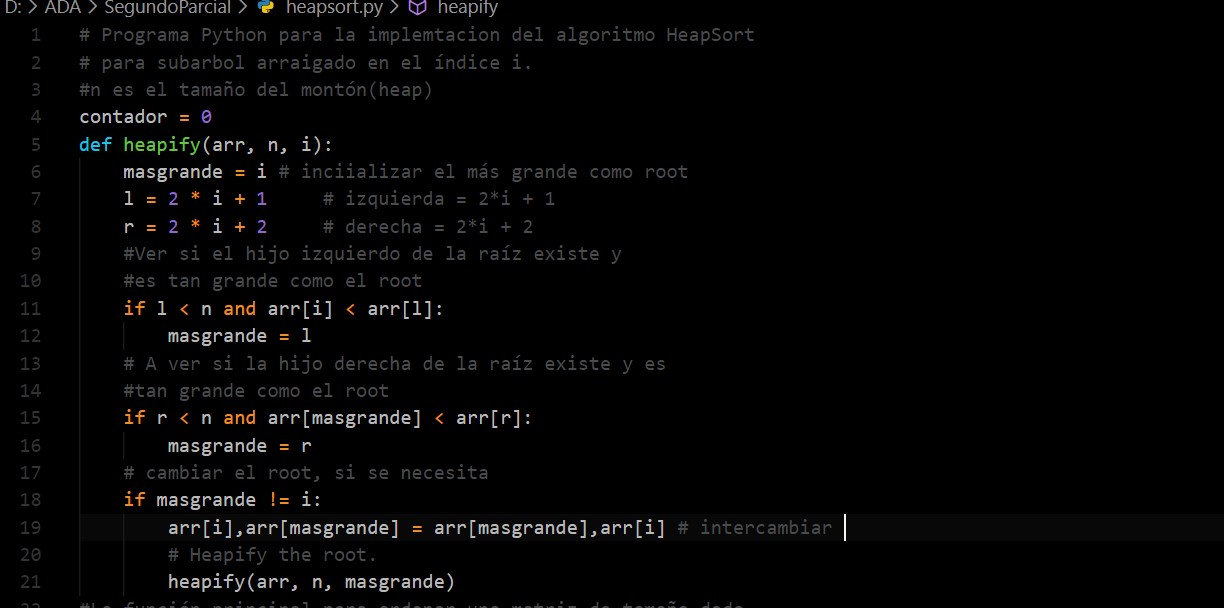
**Ejemplo gráfico:**

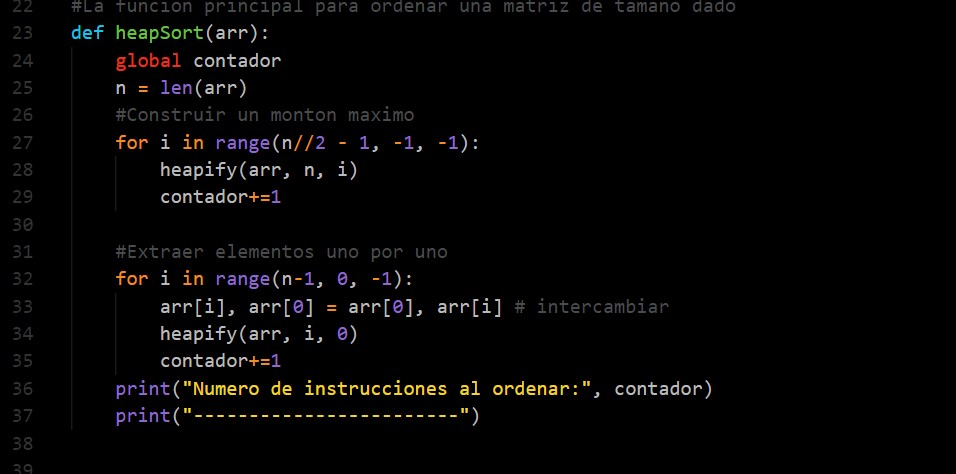


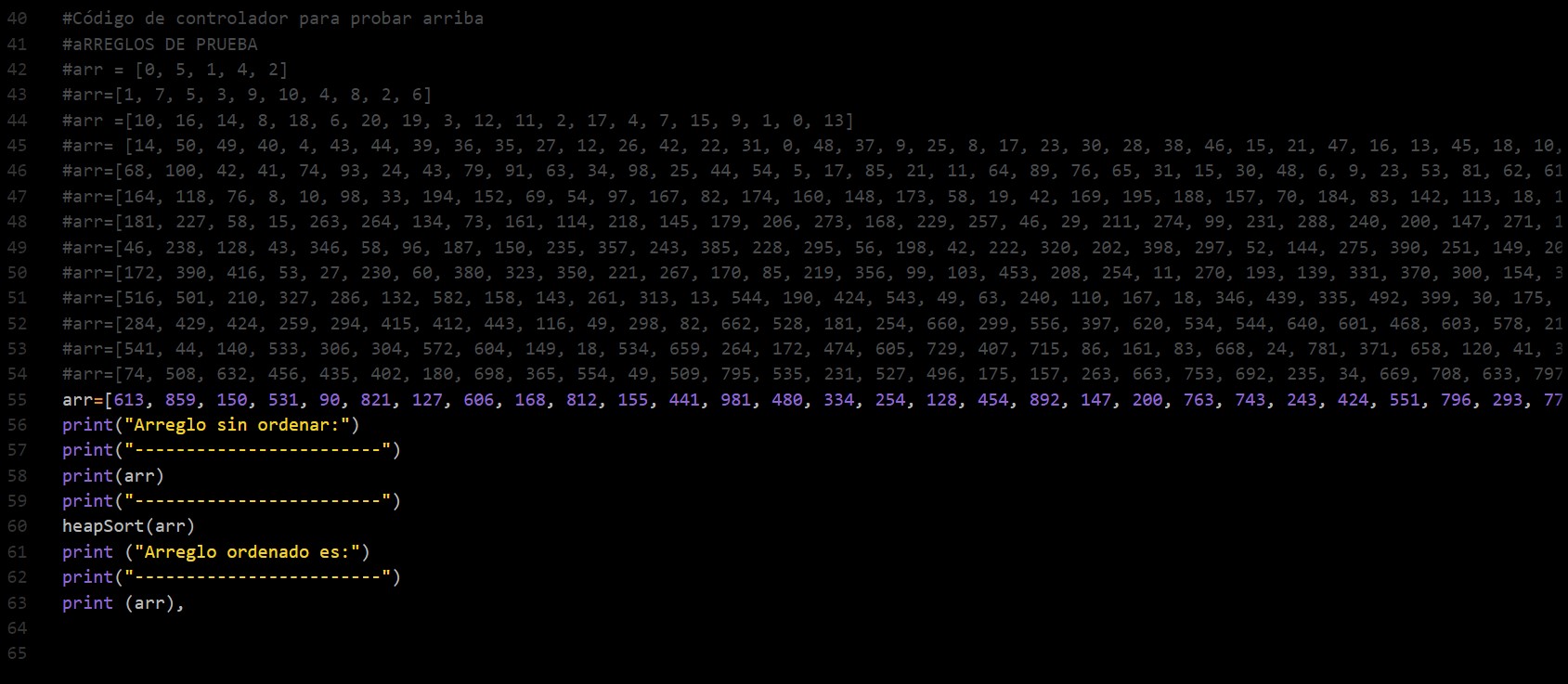
**Complejidad computacional:**

***o (n log n)***

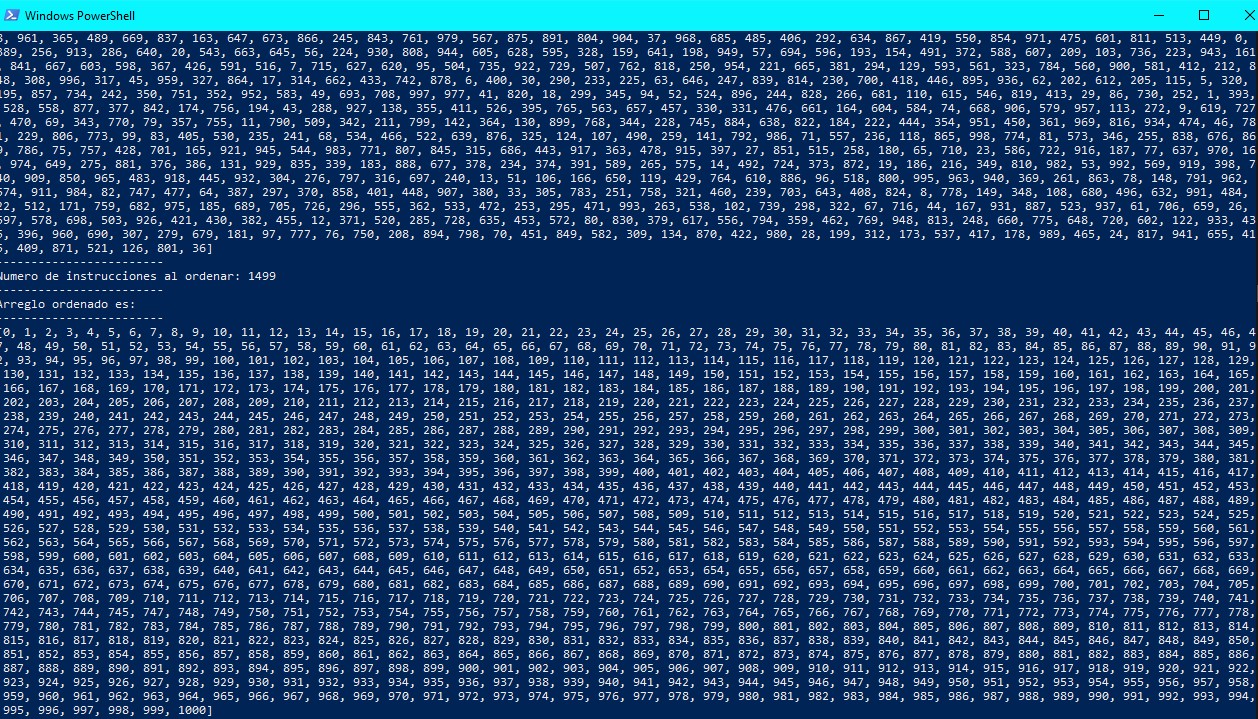
**Código en Python**







**Salida por consola al ordenar un arreglo de mil elementos(1-1000)**

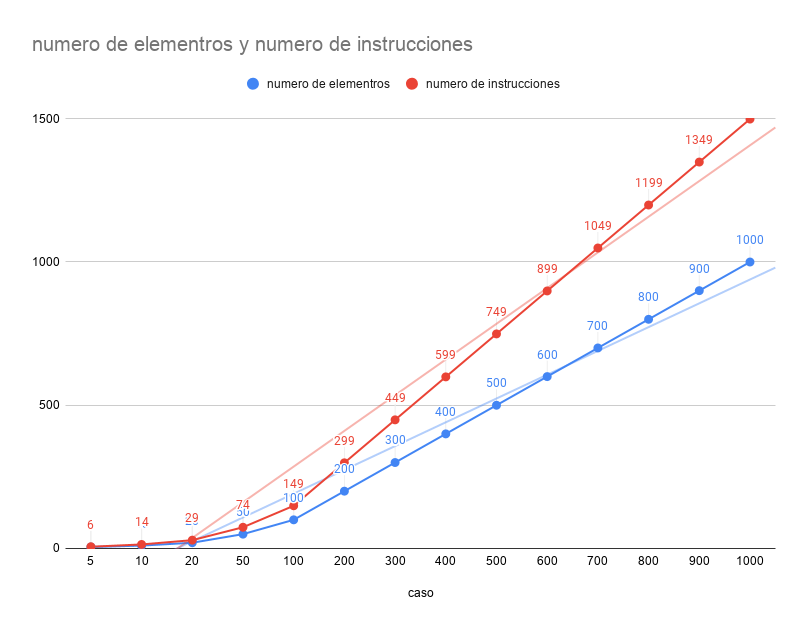


|  |  |
| --- | --- |
| Ventajas | Desventajas |
| Funciona efectivamente con datos desordenados. | No es estable. |
| Su desempeño es en promedio tan bueno como el Quicksort. | Método complejo. |
| No utiliza memoria adicional. |  |
|  |  |

### Conclusión:

Este método es conveniente cuando se trata de ordenar arreglos estáticos grandes a diferencia de otros métodos con el Quicksort y el Mergesort. Heapsort compite primariamente con Quicksort, otro método de ordenamiento muy eficiente para propósitos en general.

**Gráfica de Heap Sort**



***Quick Sort***

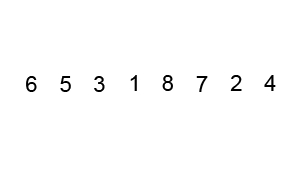
El algoritmo QuickSort se basa en la técnica de *"divide y vencerás"* por la que en cada recursión, el problema se divide en subproblemas de menor tamaño y se resuelven por separado (aplicando la misma técnica) para ser unidos de nuevo una vez resueltos.

**Funcionamiento:**

El método QuickSort basa su estrategia en la idea de que es más fácil ordenar una gran estructura de datos subdividiéndose en otras más pequeñas introduciendo un orden relativo entre ellas.

En otras palabras, si dividimos el arreglo a ordenar en dos subarreglos de forma que los elementos del subarreglo inferior sean más pequeños que los del subarreglo superior, y aplicamos el método reiteradamente, al final tendremos el arreglo inicial totalmente ordenado.

**Ejemplo Gráfico:**

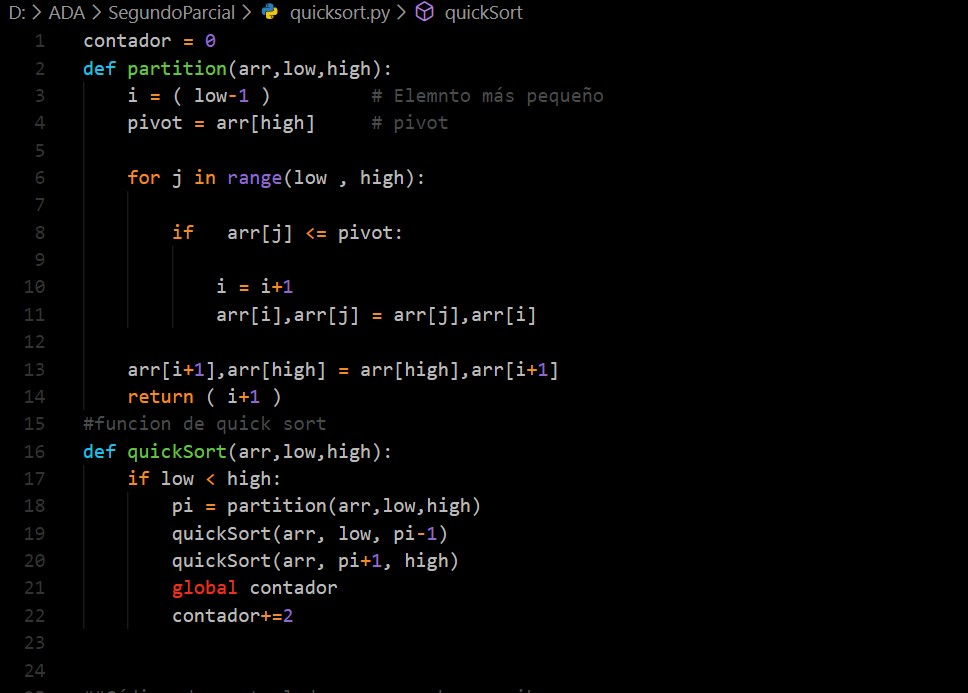


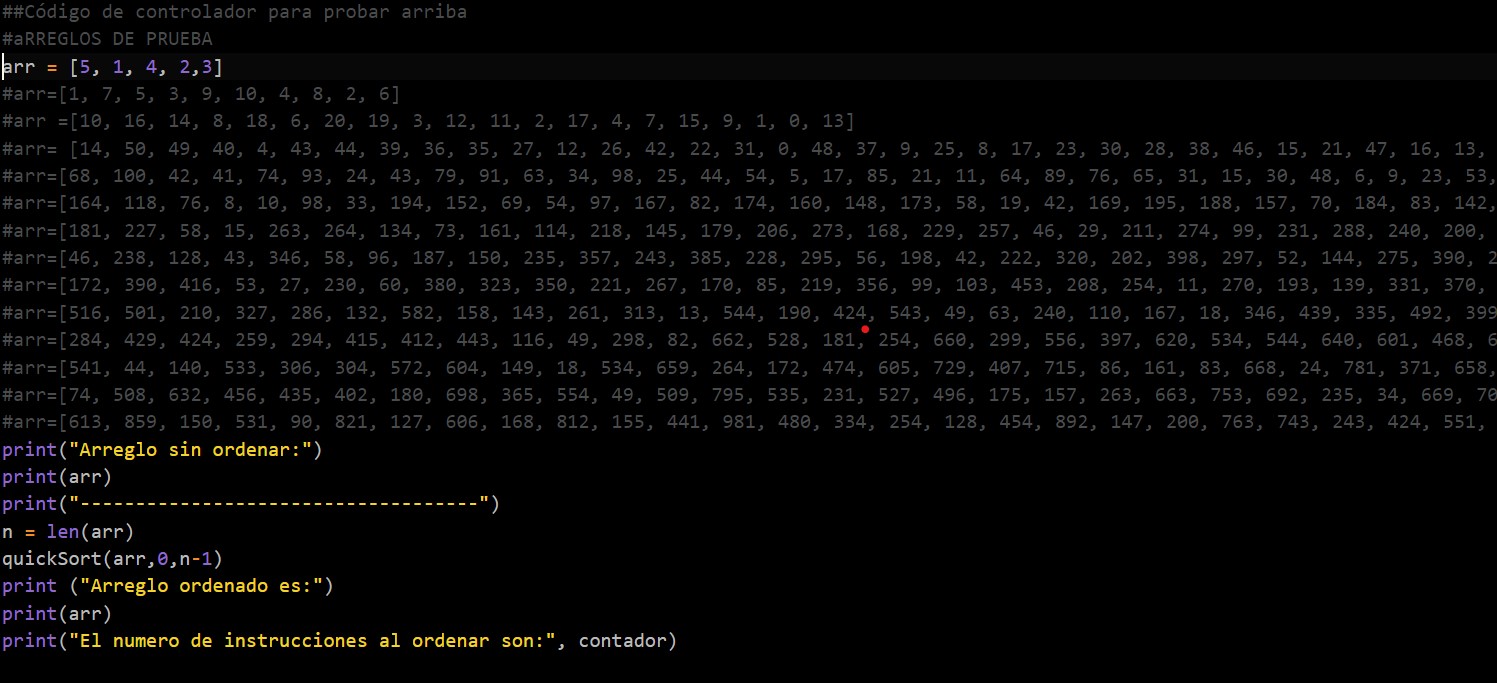
**Complejidad computacional:**

En el mejor de los casos tiene un costo de O(n\*log(n)).Que es cuando el pivote queda al medio del arreglo.

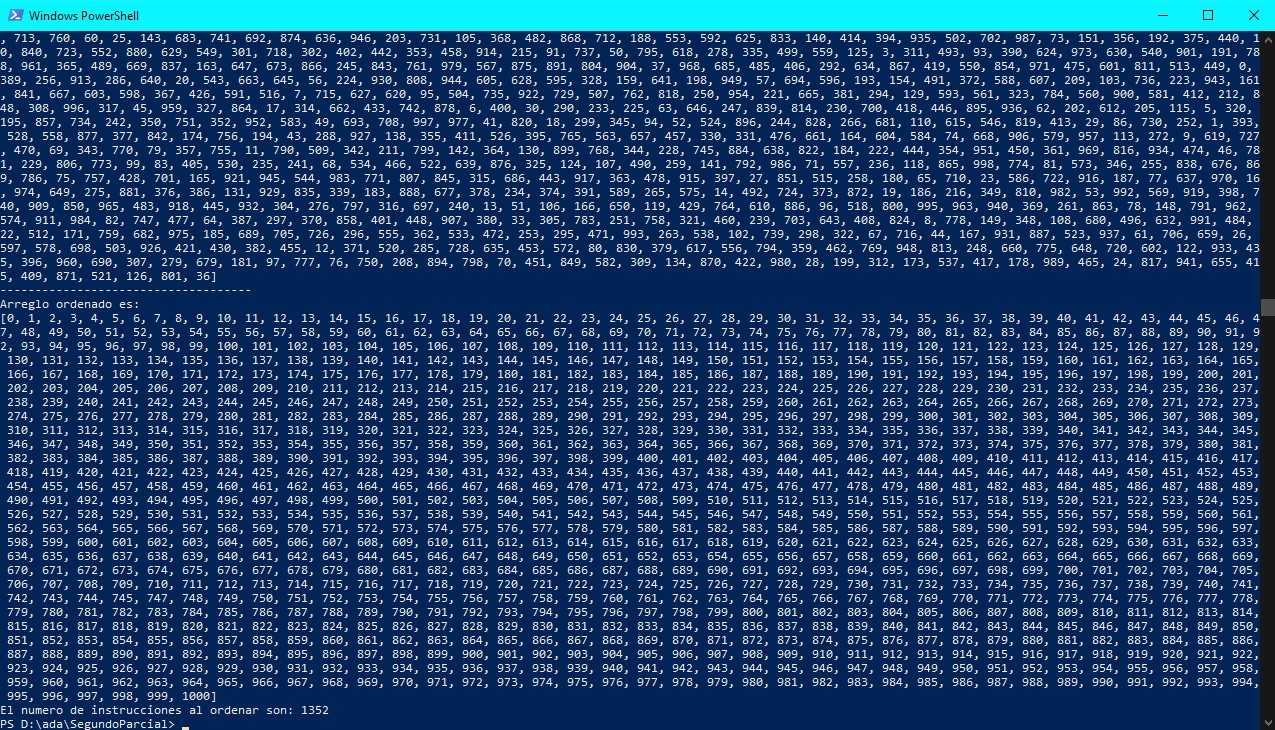
En el peor de los casos tiene un costo de O(n^2).

En el caso promedio también tiene un costo de O(n\*log(n)).



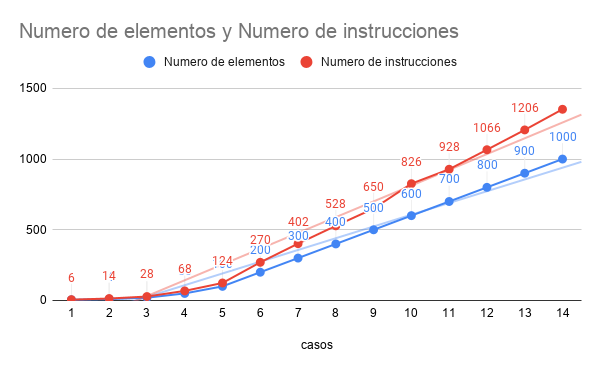


**Salida por consola al ordenar un arreglo de 1000 elementos(1-1000)**



|  |  |
| --- | --- |
| **Ventajas** | **Desventajas** |
| Muy rápido | Implementación un poco más complicada |
| No requiere memoria adicional | Recursividad |
|  |  |

**Gráfica Quick Sort**

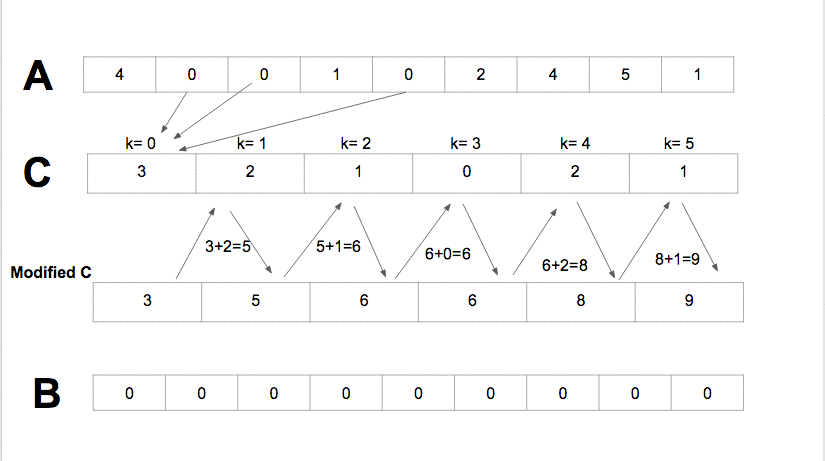


**Counting Sort**

El algoritmo de ordenamiento Counting Sort (Ordenamiento por Cuentas en español) es un algoritmo de ordenamiento en el que se cuenta el número de elementos de cada clase para luego ordenarlos. Sólo puede ser utilizado por tanto para ordenar elementos que sean contables, por ejemplo, los números enteros de un determinado intervalo, sin contar números reales.

El ordenamiento por conteo tiene la particularidad de que en ningún momento necesitamos realizar comparaciones entre los números, por lo que la cota mínima de O(*n*lg*n*) no aplica.

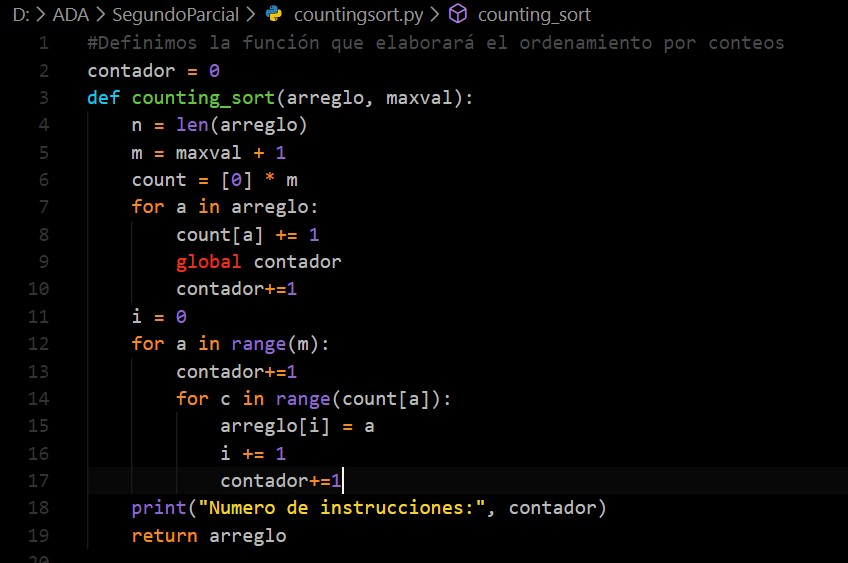
**Ejemplo Gráfico:**

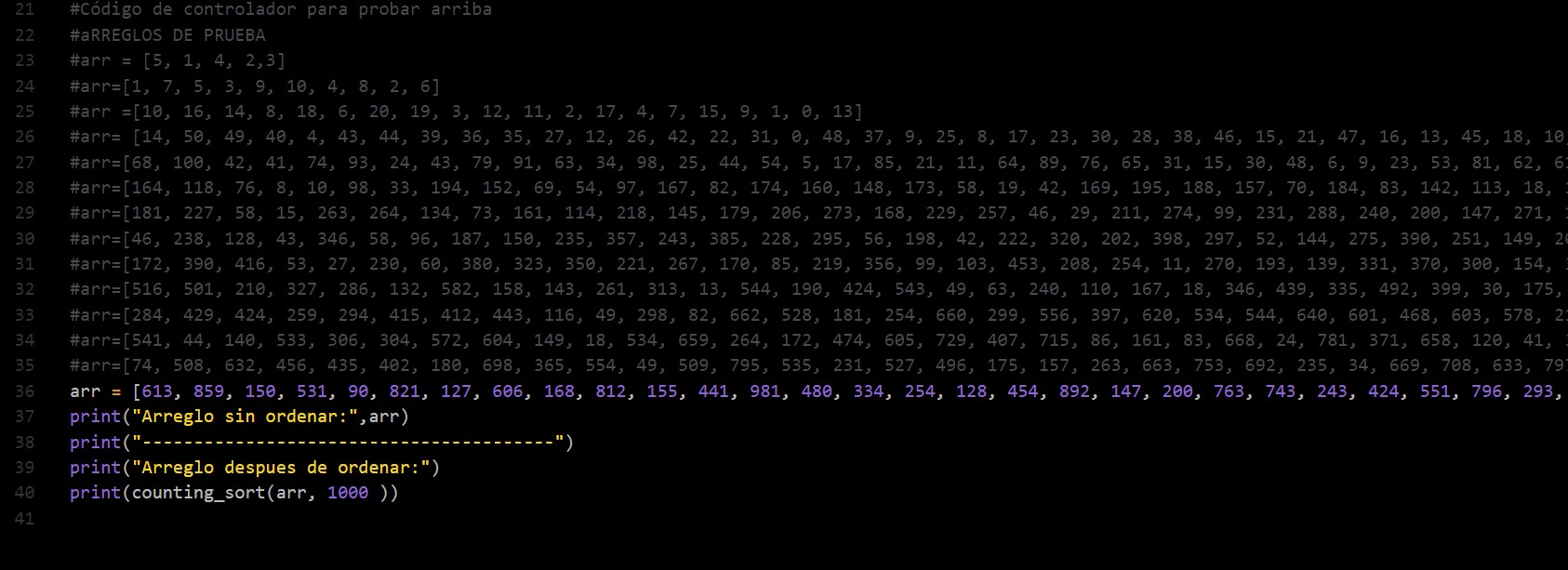


**Complejidad Computacional:**

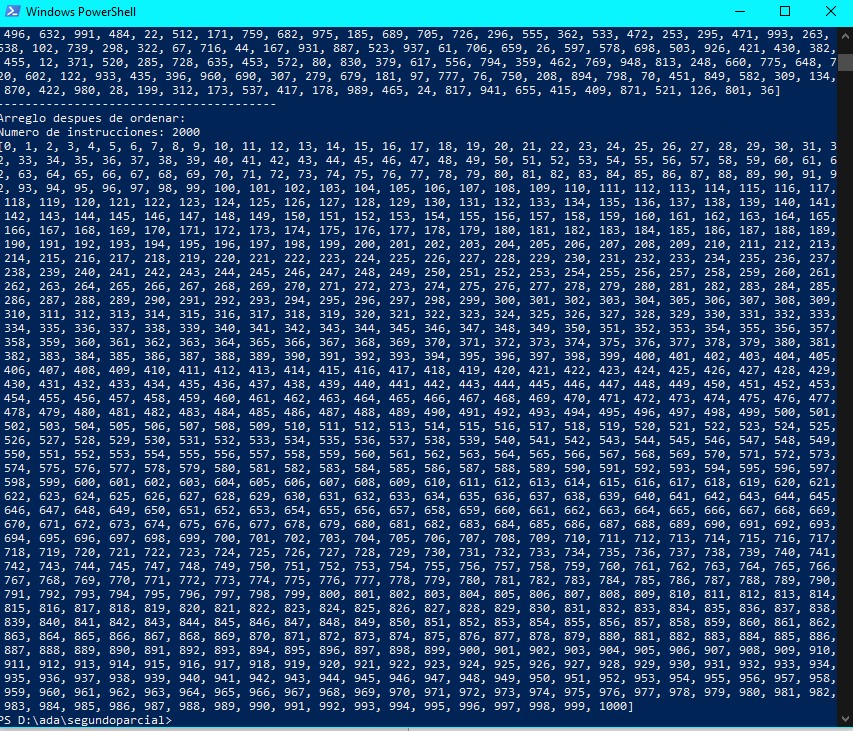
O(n+k)

**Código en Python:**

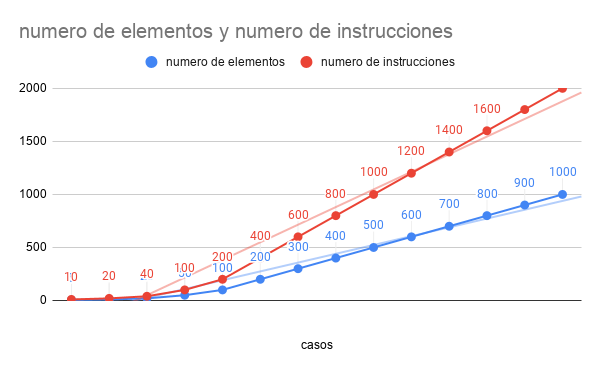




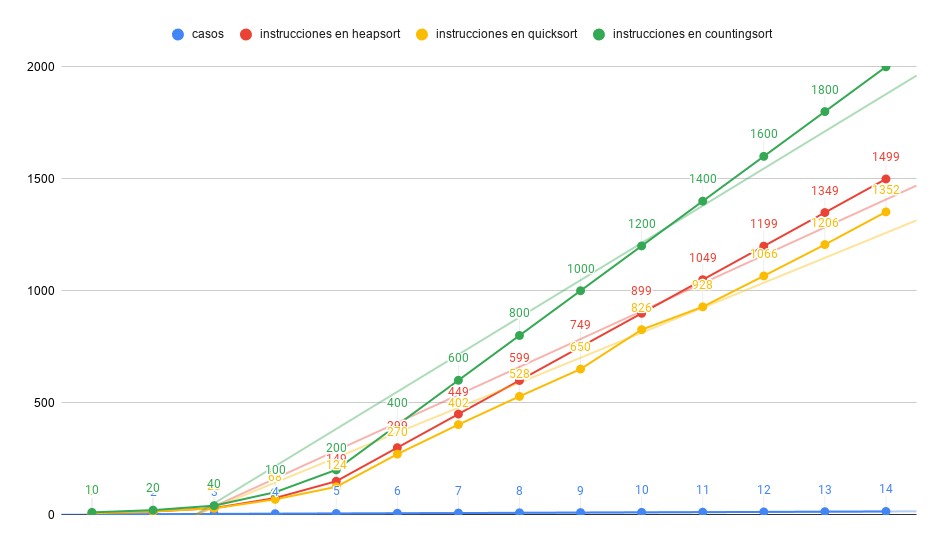
**Salida por consola al ordenar un arreglo de mil números(1-1000)**



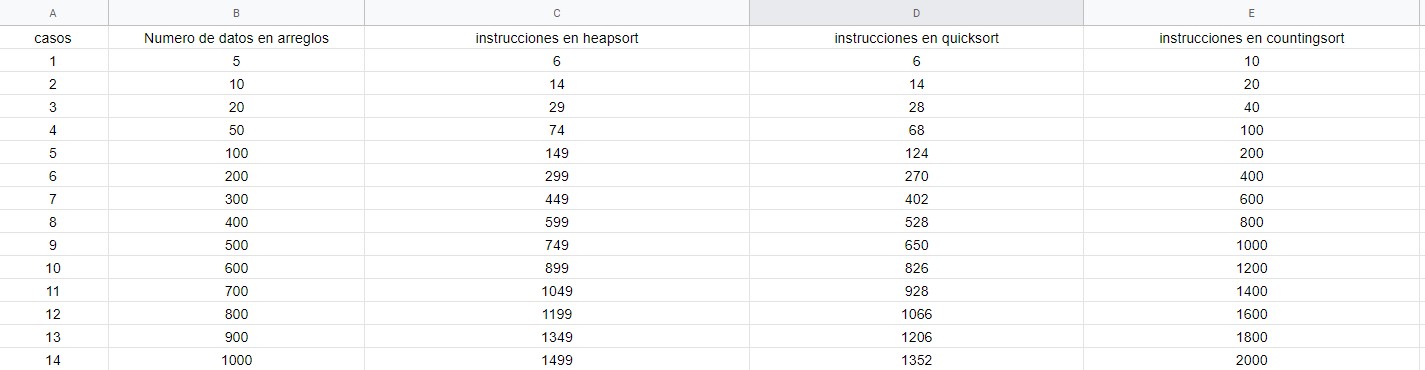
**Gráfica:**



**Comparativa de las gráficas de los tres algoritmos:**



**Conclusiones:**



**Se probaron arreglos de diferentes tamaños, que van desde un arreglo de tamaño 5 hasta uno de tamaño 1000.**

**Se pudo observar que el algoritmo más eficiente de estos tres fue el Quick Sort, y le sigue el Heap Sort y al final el Counting Sort.**