

- Fernando Hengstenberg **Carné** 17699
- David Valenzuela **Carné** 171001

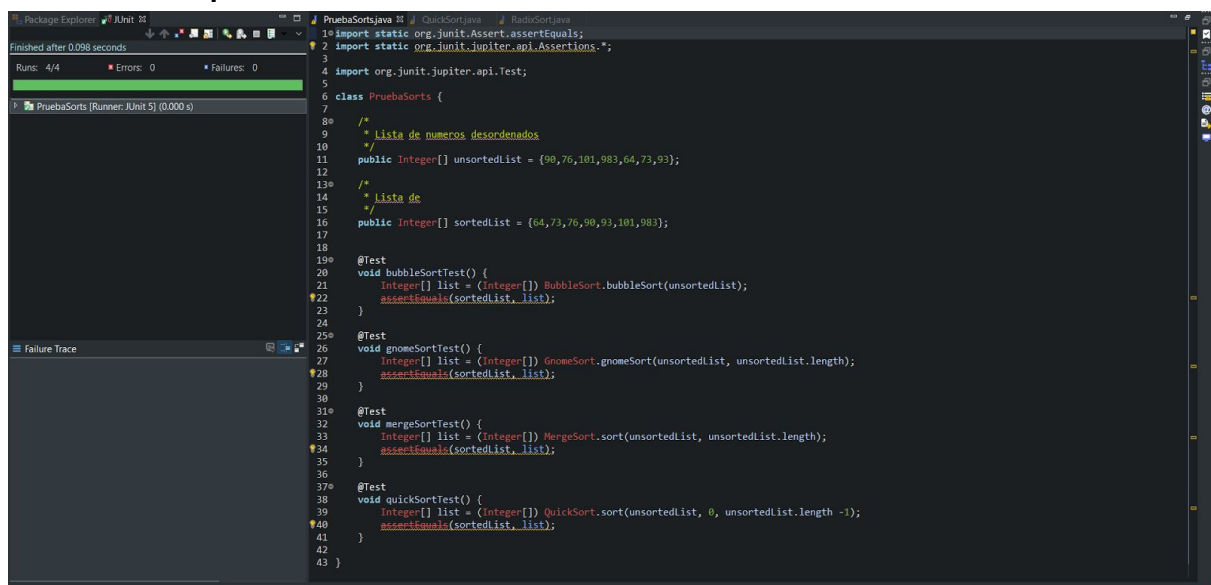
Link repositorio de github: <https://github.com/val171001/Lab3Sort.git>

Brief:

Cada algoritmo utilizo un lista de números enteros entre 0 y 10,000 en orden aleatorio, de tamaño entre 10 y 3000. Cada algoritmo ordena los datos de menor a mayor. Para todas las gráficas el eje “x” representa el número de datos (números enteros) que el algoritmo ordeno, y el eje “y” representa el tiempo que tardó en ordenar los datos en microsegundos.

Los datos en las gráficas fueron obtenidas utilizando un profiler para Java. Jprofiler fue el escogido para obtener los datos.

Pruebas JUnit para sorts:



```
1 import static org.junit.Assert.assertEquals;
2 import static org.junit.jupiter.api.Assertions.*;
3
4 import org.junit.jupiter.api.Test;
5
6 class PruebaSorts {
7
8     /*
9     * lista de numeros desordenados
10    */
11    public Integer[] unsortedList = {90,76,101,983,64,73,93};
12
13    /*
14    * lista de
15    */
16    public Integer[] sortedList = {64,73,76,90,93,101,983};
17
18    @Test
19    void bubbleSortTest() {
20        Integer[] list = (Integer[]) BubbleSort.bubbleSort(unsortedList);
21        assertEquals(sortedList, list);
22    }
23
24    @Test
25    void gnomeSortTest() {
26        Integer[] list = (Integer[]) GnomeSort.gnomeSort(unsortedList, unsortedList.length);
27        assertEquals(sortedList, list);
28    }
29
30    @Test
31    void mergeSortTest() {
32        Integer[] list = (Integer[]) MergeSort.sort(unsortedList, unsortedList.length);
33        assertEquals(sortedList, list);
34    }
35
36    @Test
37    void quickSortTest() {
38        Integer[] list = (Integer[]) QuickSort.sort(unsortedList, 0, unsortedList.length - 1);
39        assertEquals(sortedList, list);
40    }
41
42
43 }
```

Prueba de JUnit consiste en verificar que los sorts están realmente ordenando los datos, se genero una lista en orden aleatorio, y una lista con los mismos elemento ya ordenados de menor a mayor. Cada prueba ordena la lista que esta desordenada y utiliza los asserts de JUnit para verificar que estén ordenados. (Imagen y código incluida en el repositorio de github).

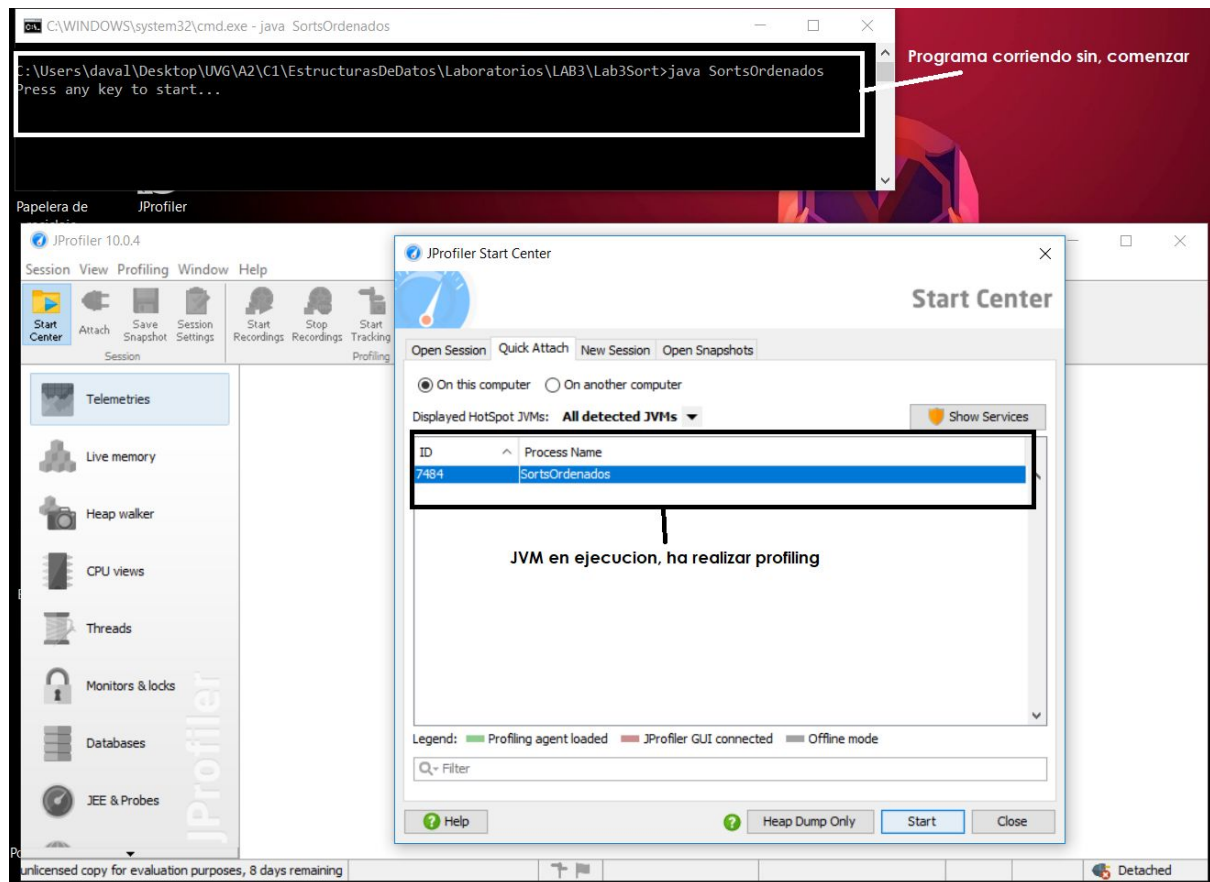
Proceso de obtención de datos.

Para la obtención del rendimiento de los programas se utilizó Jprofiler, este mide el tiempo total de tiempo que tarda cada método en terminar.

El primer paso para comenzar a realizar el profiling de los programas es adjuntar la máquina virtual de Java a JProfiler. Esto se realiza ejecutando el programa, sin comenzar la ejecución de los métodos que se busca realizar el profiling.

Hoja de Trabajo # 3

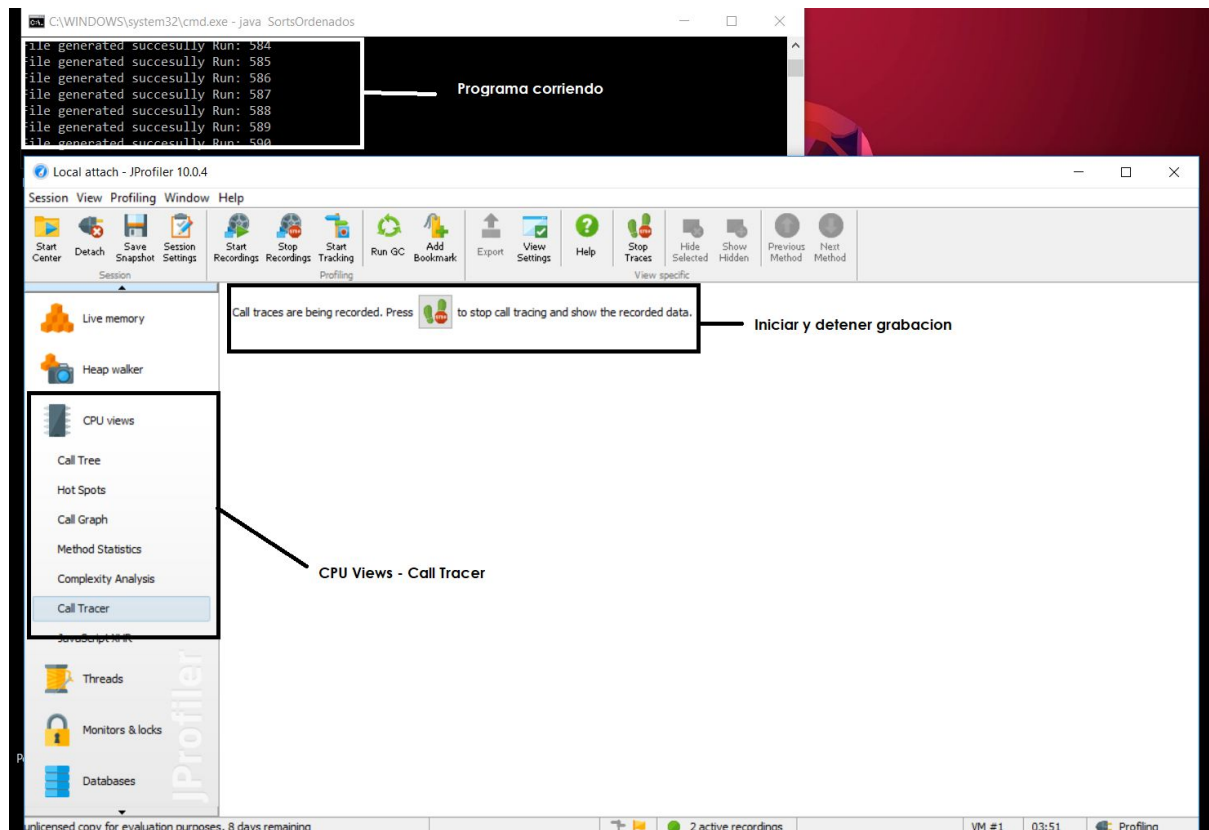
- Fernando Hengstenberg **Carné** 17699
- David Valenzuela **Carné** 171001



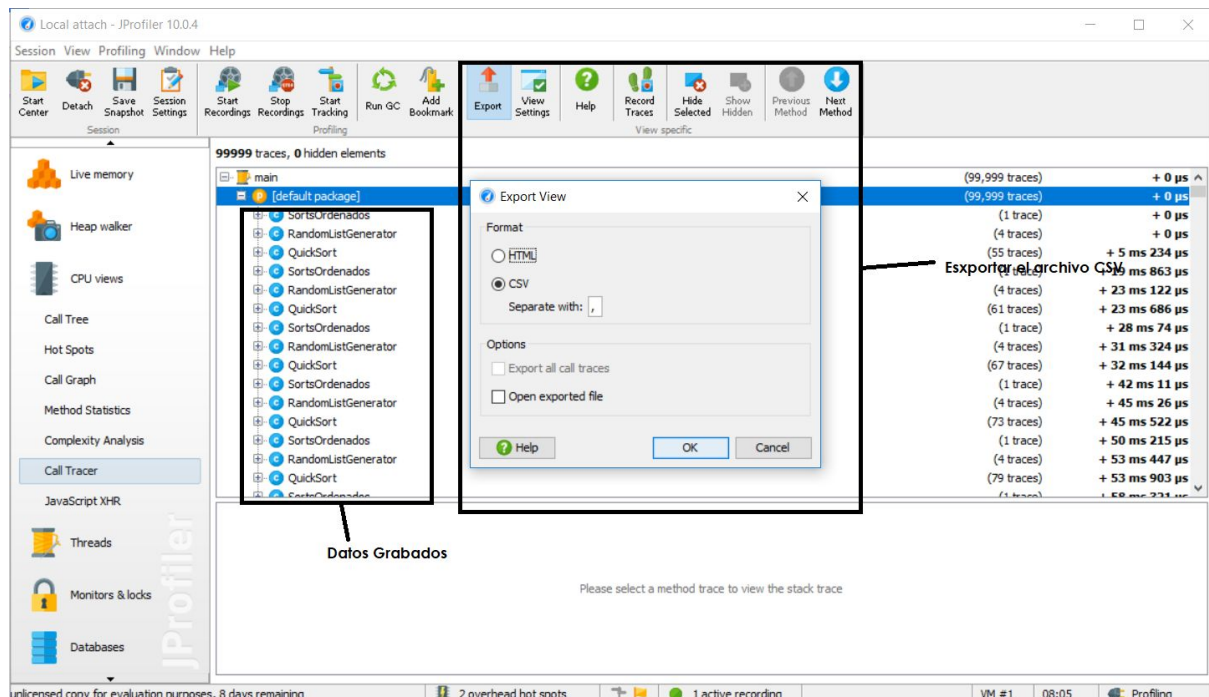
Al tener la máquina virtual de Java lista en el profiling, en la opción “CPU View”, se seleccionó la opción “Call Tracer”, este fue seleccionada ya que da el tiempo que le tomó a cada método en terminar cada vez que este fue llamado. Al estar en esta sección se debe de comenzar la grabación de “Call Tracer”, y comenzar la ejecución del programa.

Hoja de Trabajo # 3

- Fernando Hengstenberg **Carné** 17699
- David Valenzuela **Carné** 171001

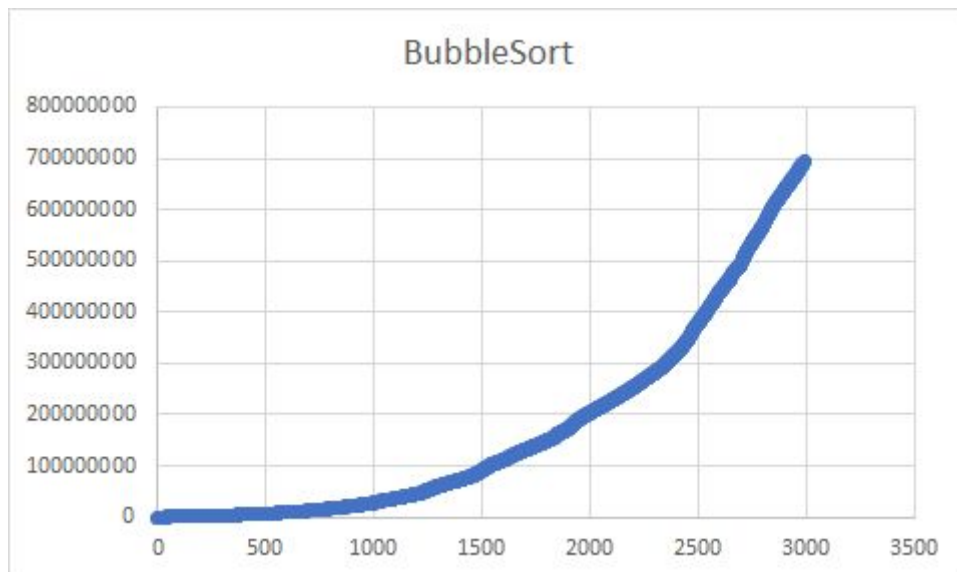


Al terminar la ejecución del programa los datos encontrados tienen que ser exportados para poder ser utilizados. JProfiler cuenta con la opción de exportar los datos como un CSV (Comma Separated Values), que luego fueron importados hacia excel donde se realizaron las gráficas.



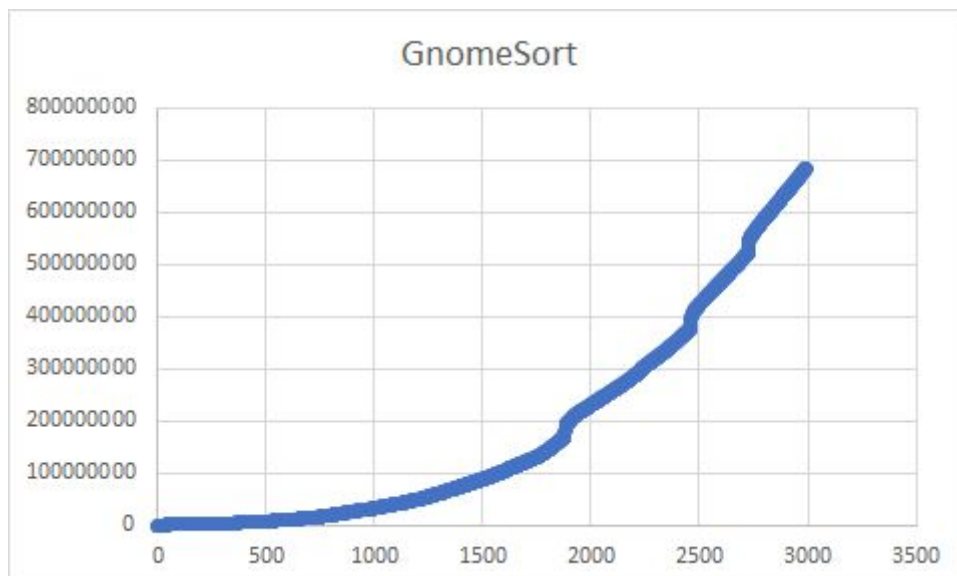
- Fernando Hengstenberg **Carné** 17699
- David Valenzuela **Carné** 171001

Graficas datos desordenados:



Rendimiento teórico:

- Peor caso: $O(n^2)$
- Mejor caso: $O(n)$
- Caso promedio: $O(n^2)$

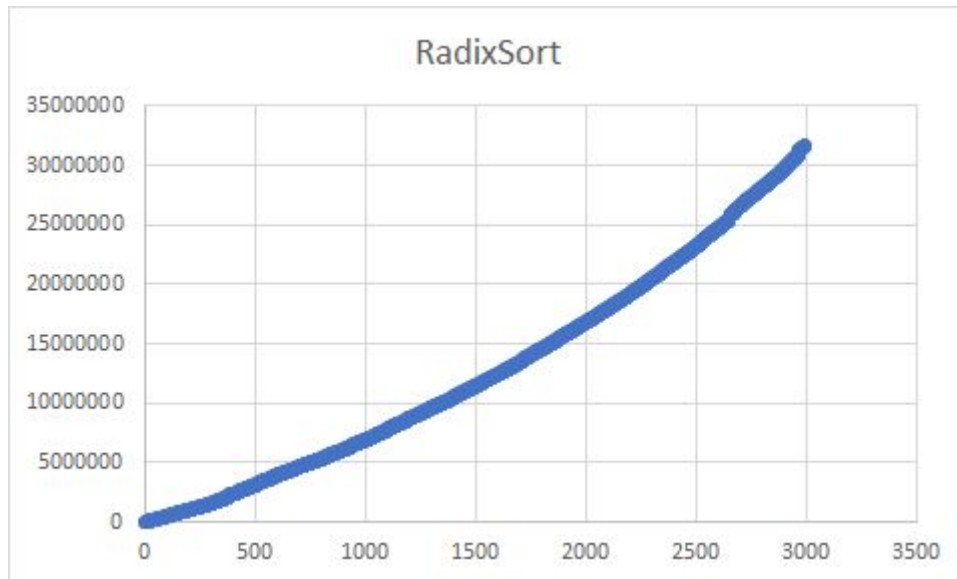


Rendimiento Teórico:

- Peor caso: $O(n^2)$
- Mejor caso: $O(n)$
- Caso promedio: $O(n^2)$

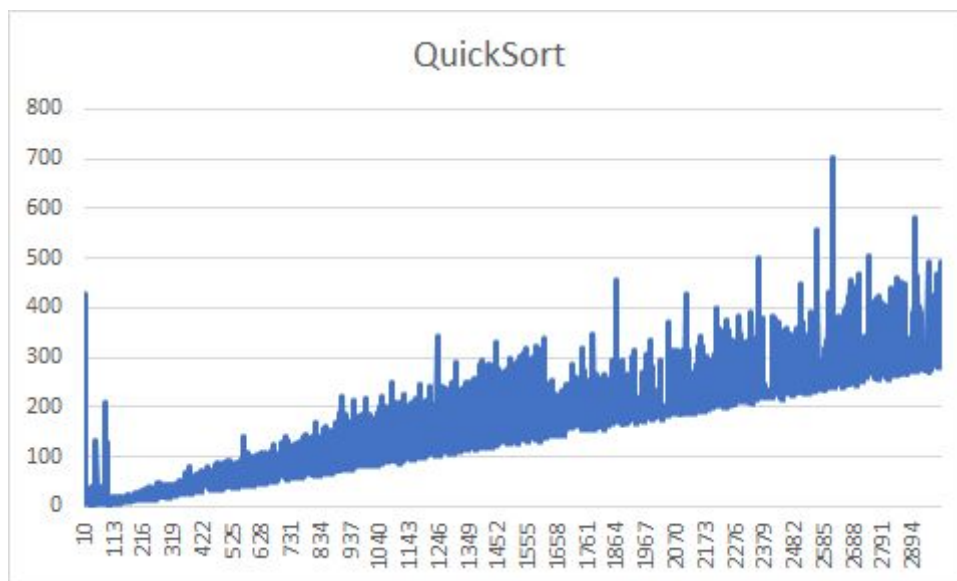
Hoja de Trabajo # 3

- Fernando Hengstenberg **Carné** 17699
- David Valenzuela **Carné** 171001



Rendimiento Teórico:

- Peor caso: $O(nk)$
- Mejor caso: $O(nk)$
- Caso promedio: $O(nk)$

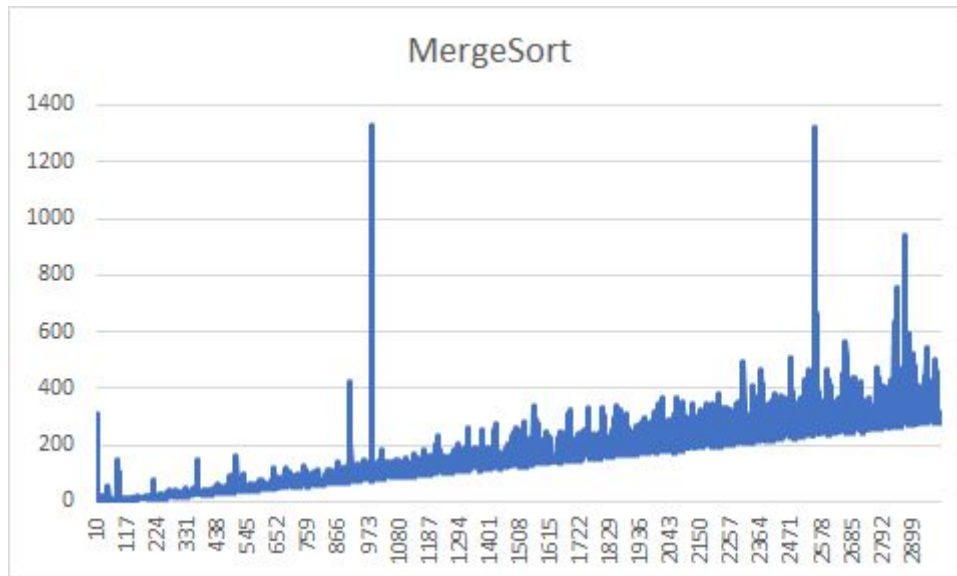


Rendimiento Teórico:

- Peor caso: $O(n^2)$
- Mejor caso: $O(n * \log(n))$
- Caso promedio: $O(n * \log(n))$

Hoja de Trabajo # 3

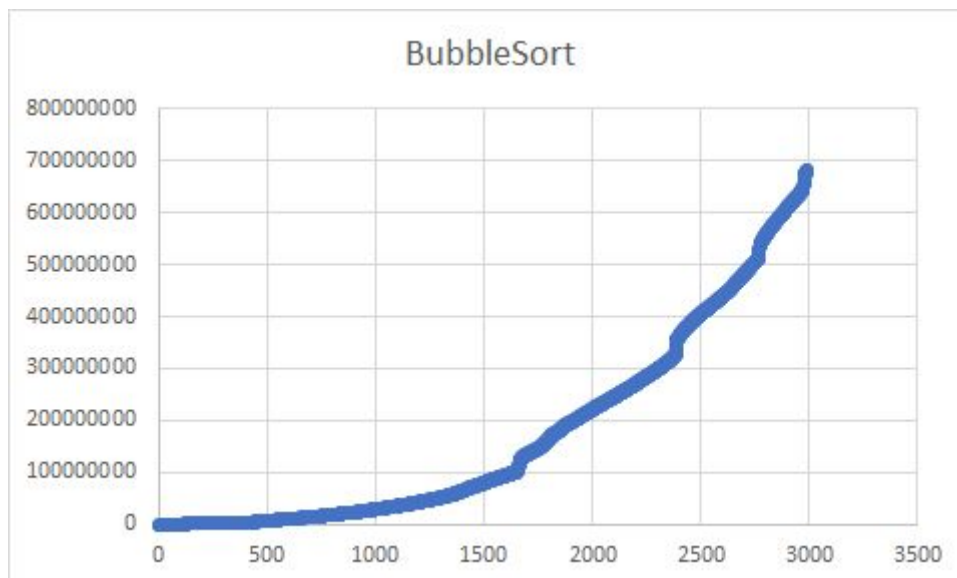
- Fernando Hengstenberg **Carné** 17699
- David Valenzuela **Carné** 171001



Rendimiento Teórico:

- Peor caso: $O(n * \log(n))$
- Mejor caso: $O(n * \log(n))$
- Caso promedio: $O(n * \log(n))$

Gráficas para datos ya ordenados:

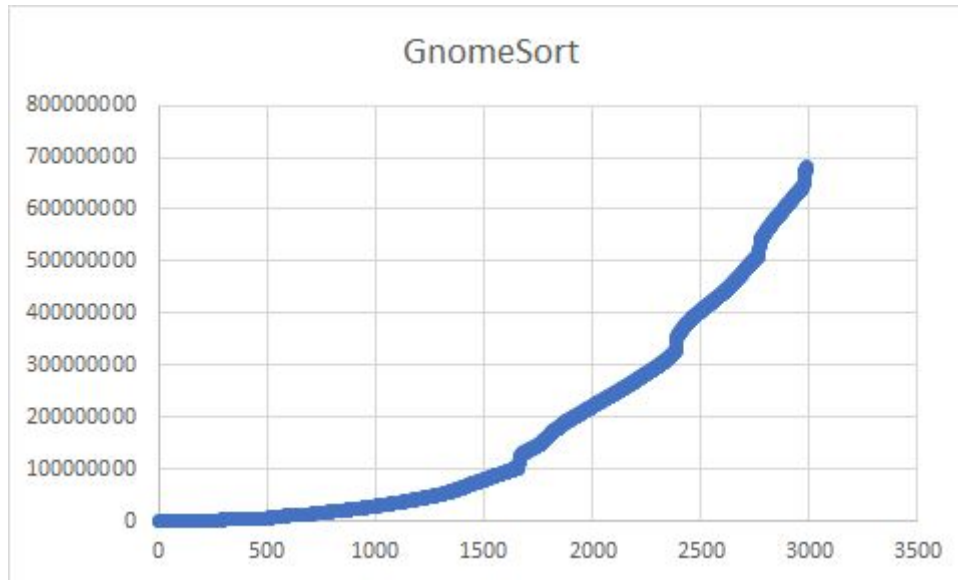


Rendimiento teórico:

- Peor caso: $O(n^2)$
- Mejor caso: $O(n)$

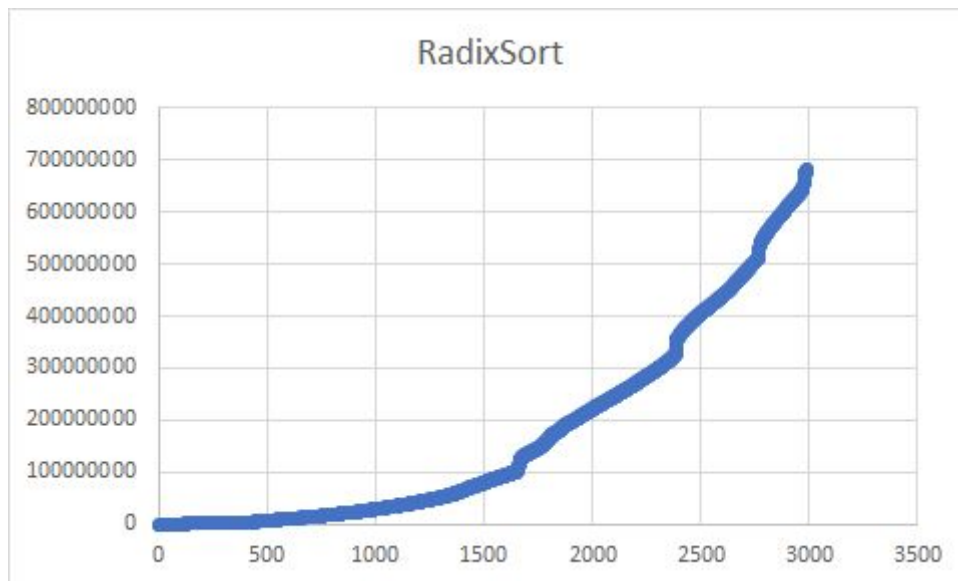
Hoja de Trabajo # 3

- Fernando Hengstenberg **Carné** 17699
- David Valenzuela **Carné** 171001
- Caso promedio: $O(n^2)$



Rendimiento Teórico:

- Peor caso: $O(n^2)$
- Mejor caso: $O(n)$
- Caso promedio: $O(n^2)$

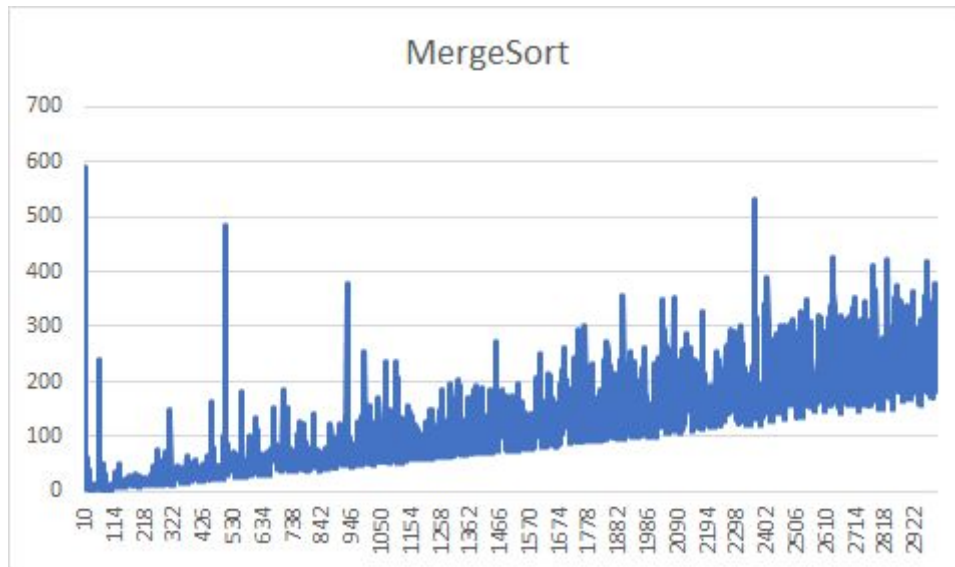


Rendimiento Teórico:

- Peor caso: $O(nk)$
- Mejor caso: $O(nk)$
- Caso promedio: $O(nk)$

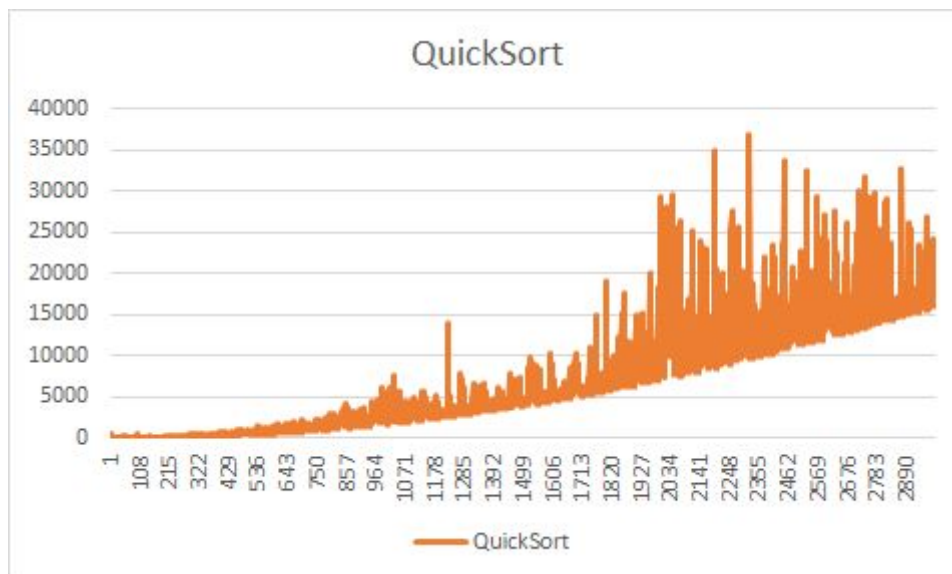
Hoja de Trabajo # 3

- Fernando Hengstenberg **Carné** 17699
- David Valenzuela **Carné** 171001



Rendimiento Teórico:

- Peor caso: $O(n * \log(n))$
- Mejor caso: $O(n * \log(n))$
- Caso promedio: $O(n * \log(n))$



Rendimiento Teórico:

- Peor caso: $O(n^2)$
- Mejor caso: $O(n * \log(n))$
- Caso promedio: $O(n * \log(n))$

- Fernando Hengstenberg **Carné** 17699
- David Valenzuela **Carné** 171001

Referencias

Geeks For Geeks. (12 de Febrero de 2016). Radix Sort. Obtenido de Geeks for Geeks: <https://www.geeksforgeeks.org/radix-sort/>

Gupta, M. (20 de Marzo de 2016). Gnome Sort. Obtenido de Geeks for Geeks: <https://www.geeksforgeeks.org/gnome-sort-a-stupid-one/>

Mishra, R. (25 de Mayo de 2016). Bubble Sort. Obtenido de Geeks for Geeks: <https://www.geeksforgeeks.org/bubble-sort/>

Mishra, R. (25 de Junio de 2017). Merge Sort. Obtenido de Geeks for Geeks: <https://www.geeksforgeeks.org/merge-sort/>

Mishra, R. (13 de Mayo de 2017). Quicksort. Obtenido de Geeks for Geeks: <https://www.geeksforgeeks.org/quick-sort/>

Mohit, K. (18 de Julio de 2017). Iterative Quick Sort. Obtenido de GeeksforGeeks : <http://www.geeksforgeeks.org/iterative-quick-sort/>

Moore, K., Oilling, G., & Takvan, K. (18 de Mayo de 2016). Sorting Algorithms. Obtenido de Brilliant: <https://brilliant.org/wiki/sorting-algorithms/>