

Esteban Enrique Cárcamo Urizar

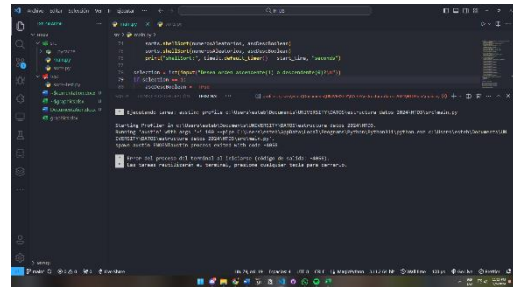
23016

Algoritmos y Estructuras de Datos

Moisés Alonso

Repo: <https://github.com/CarEsteban/estructura-datos-2024/tree/main/HTD3>

Profiler Austin:



Se tuvieron diferentes inconvenientes al momento de usar el profiler Austin, se buscó documentación o vídeos de uso del profiler y no se encontró, por lo cual se usaron dos alternativas diferentes para sacar los resultados en segundos de cada sort.

289704 function calls (211273 primitive calls) in 2.938 seconds

Ordered by: cumulative time

ncalls	tottime	percall	cume	percall	filename:lineo(function)
5/1	0.000	0.000	2.938	2.938	{built-in method builtins.exec}
1	0.000	0.000	2.938	2.938	main.py:1(<module>)
1	1.519	1.519	1.519	1.519	{built-in method builtins.input}
1	0.000	0.000	1.405	1.405	main.py:42(run_algorithmsDoubleOrden)
2	1.227	0.614	1.227	0.614	sorts.py:175(selectionSort)
2	0.003	0.002	0.065	0.033	sorts.py:43(heapSort)
71390/8998	0.062	0.000	0.062	0.000	sorts.py:13(heapify)
11998/2	0.044	0.000	0.056	0.028	sorts.py:58(mergeSort)
2	0.000	0.000	0.018	0.009	main.py:3(generatorRandom)
2	0.002	0.001	0.018	0.009	main.py:5(elistcomp)
2	0.000	0.000	0.017	0.008	sorts.py:165(radixSort)
8	0.017	0.002	0.017	0.002	sorts.py:127(counting)
6000	0.002	0.000	0.016	0.000	random.py:350(randint)
2	0.015	0.007	0.015	0.007	sorts.py:195(shellSort)
6000	0.006	0.000	0.014	0.000	random.py:284(randrange)
146501	0.013	0.000	0.013	0.000	{built-in method builtins.len}
4009/1	0.002	0.000	0.012	0.012	sorts.py:116(quickSort)
2004	0.010	0.000	0.010	0.000	sorts.py:93(partition)
6000	0.004	0.000	0.006	0.000	random.py:235(_randbelow_with_getrandbits)
9/3	0.000	0.000	0.004	0.001	<frozen importlib._bootstrap>:1167(_find_and_load)
9/3	0.000	0.000	0.004	0.001	<frozen importlib._bootstrap>:1122(_find_and_load_unlocked)
9/3	0.000	0.000	0.003	0.001	<frozen importlib._bootstrap>:666(_load_unlocked)
2	0.002	0.001	0.003	0.001	sorts.py:3(gnomeSort)
4/3	0.000	0.000	0.003	0.001	<frozen importlib._bootstrap_external>:934(exec_module)
13/6	0.000	0.000	0.002	0.000	<frozen importlib._bootstrap>:233(_call_with_frames_removed)
18000	0.002	0.000	0.002	0.000	{built-in method operator.index}
4	0.000	0.000	0.001	0.000	<frozen importlib._bootstrap_external>:1007(get_code)
9	0.000	0.000	0.001	0.000	<frozen importlib._bootstrap>:1856(_find_spec)

Live Share

Usando cProfile, se logró ver de una forma general cuando tardaban todos los métodos usados, se usó el comando: > python -m cProfile main.py para poder correr el Profiler y dar un estimado del tiempo.

Al final, se terminó pidiendo a la IA que generase un código para poder contar los tiempos de cada función de una forma manual, usando la librería timeit.

```
#Se le pidió a la IA que creara una función para medir el tiempo, ya que Austin Profiler no funcionaba
def run_algorithms(ascDescBoolean):
    numerosAleatorios = generatorRandom()
    start_time = timeit.default_timer()
    sorts.quickSort(numerosAleatorios, 0, len(numerosAleatorios) - 1, ascDescBoolean)
    print("quickSort:", timeit.default_timer() - start_time, "seconds")

    start_time = timeit.default_timer()
    sorts.gnomeSort(numerosAleatorios, ascDescBoolean)
    print("gnomeSort:", timeit.default_timer() - start_time, "seconds")

    start_time = timeit.default_timer()
    sorts.heapSort(numerosAleatorios, ascDescBoolean)
    print("heapSort:", timeit.default_timer() - start_time, "seconds")

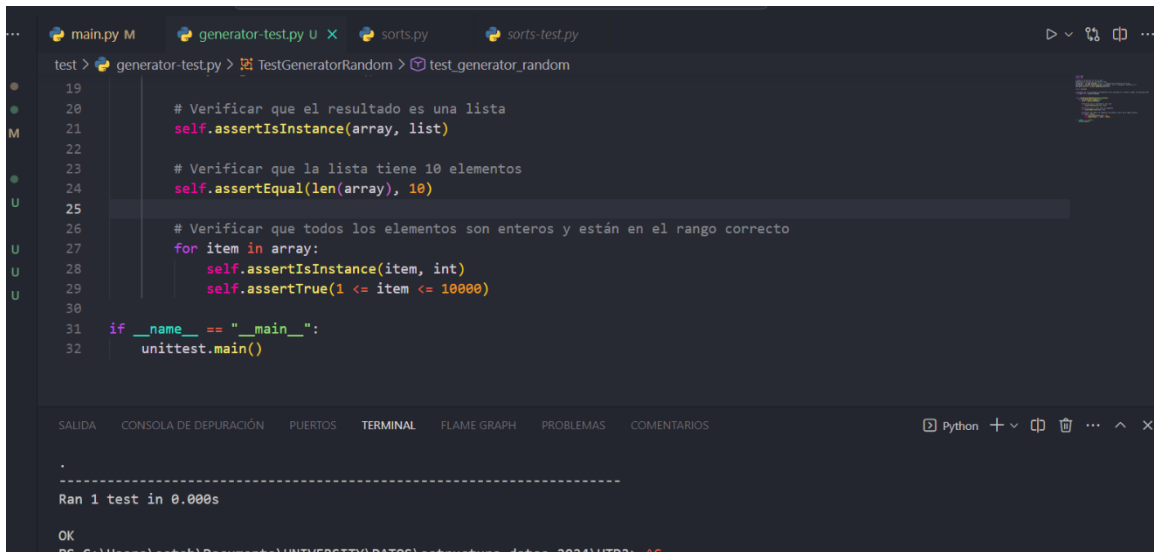
    start_time = timeit.default_timer()
    sorts.mergeSort(numerosAleatorios, ascDescBoolean)
    print("mergeSort:", timeit.default_timer() - start_time, "seconds")

    start_time = timeit.default_timer()
    sorts.radixSort(numerosAleatorios, ascDescBoolean)
    print("radixSort:", timeit.default_timer() - start_time, "seconds")

    start_time = timeit.default_timer()
    sorts.selectionSort(numerosAleatorios, ascDescBoolean)
    print("selectionSort:", timeit.default_timer() - start_time, "seconds")

    start_time = timeit.default_timer()
    sorts.shellSort(numerosAleatorios, ascDescBoolean)
    print("shellSort:", timeit.default_timer() - start_time, "seconds")
```

Esteban Enrique Cárcamo Urízar
23016
Algoritmos y Estructuras de Datos
Moisés Alonso
Tests

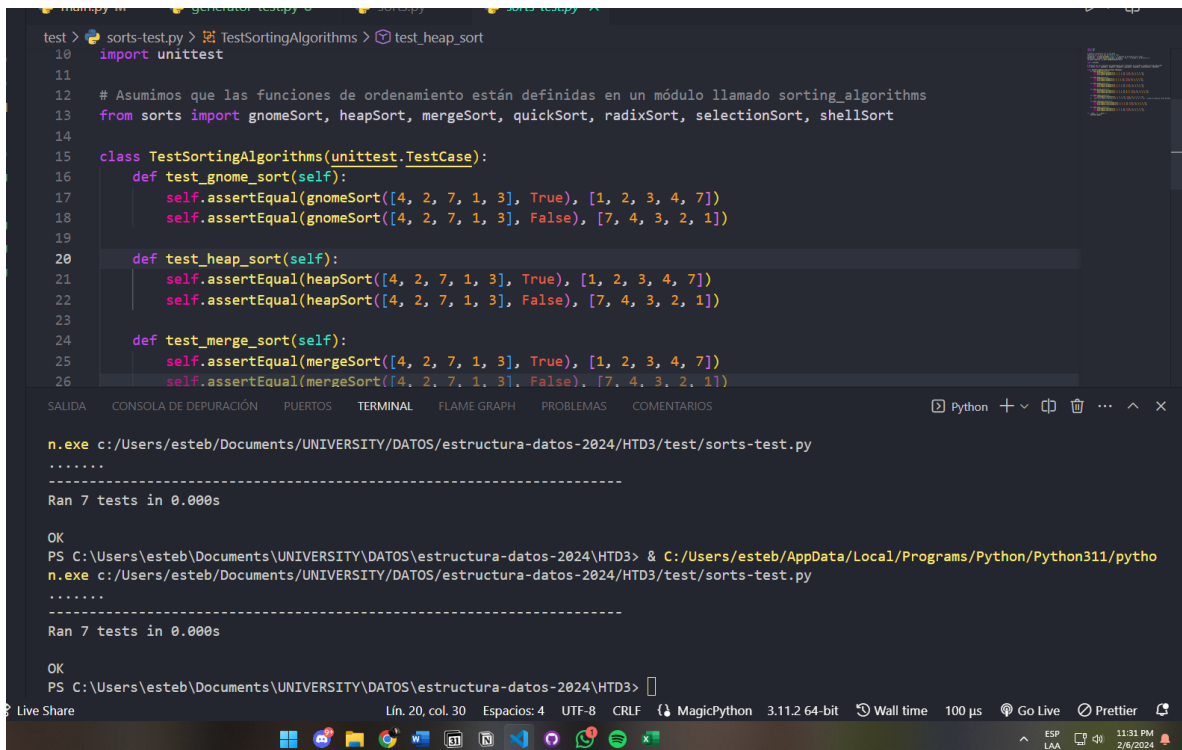


```
test > generator-test.py > TestGeneratorRandom > test_generator_random
19
20 # Verificar que el resultado es una lista
21 self.assertEqual(array, list)
22
23 # Verificar que la lista tiene 10 elementos
24 self.assertEqual(len(array), 10)
25
26 # Verificar que todos los elementos son enteros y están en el rango correcto
27 for item in array:
28     self.assertIsInstance(item, int)
29     self.assertTrue(1 <= item <= 10000)
30
31 if __name__ == "__main__":
32     unittest.main()

-----
Ran 1 test in 0.000s

OK
PS C:\Users\esteb\Documents\UNIVERSITY\DATOS\estructura-datos-2024\HTD3>
```

Funciona el generador de números random



```
test > sorts-test.py > TestSortingAlgorithms > test_heap_sort
10 import unittest
11
12 # Asumimos que las funciones de ordenamiento están definidas en un módulo llamado sorting_algorithms
13 from sorts import gnomeSort, heapSort, mergeSort, quickSort, radixSort, selectionSort, shellSort
14
15 class TestSortingAlgorithms(unittest.TestCase):
16     def test_gnome_sort(self):
17         self.assertEqual(gnomeSort([4, 2, 7, 1, 3], True), [1, 2, 3, 4, 7])
18         self.assertEqual(gnomeSort([4, 2, 7, 1, 3], False), [7, 4, 3, 2, 1])
19
20     def test_heap_sort(self):
21         self.assertEqual(heapSort([4, 2, 7, 1, 3], True), [1, 2, 3, 4, 7])
22         self.assertEqual(heapSort([4, 2, 7, 1, 3], False), [7, 4, 3, 2, 1])
23
24     def test_merge_sort(self):
25         self.assertEqual(mergeSort([4, 2, 7, 1, 3], True), [1, 2, 3, 4, 7])
26         self.assertEqual(mergeSort([4, 2, 7, 1, 3], False), [7, 4, 3, 2, 1])

n.exe c:/Users/esteb/Documents/UNIVERSITY/DATOS/estructura-datos-2024/HTD3/test/sorts-test.py
.....
Ran 7 tests in 0.000s

OK
PS C:\Users\esteb\Documents\UNIVERSITY\DATOS\estructura-datos-2024\HTD3> & C:/Users/esteb/AppData/Local/Programs/Python/Python311/pytho
n.exe c:/Users/esteb/Documents/UNIVERSITY/DATOS/estructura-datos-2024/HTD3/test/sorts-test.py
.....
Ran 7 tests in 0.000s

OK
PS C:\Users\esteb\Documents\UNIVERSITY\DATOS\estructura-datos-2024\HTD3>
```

Funcionan los tests que validan el funcionamiento de cada sort.

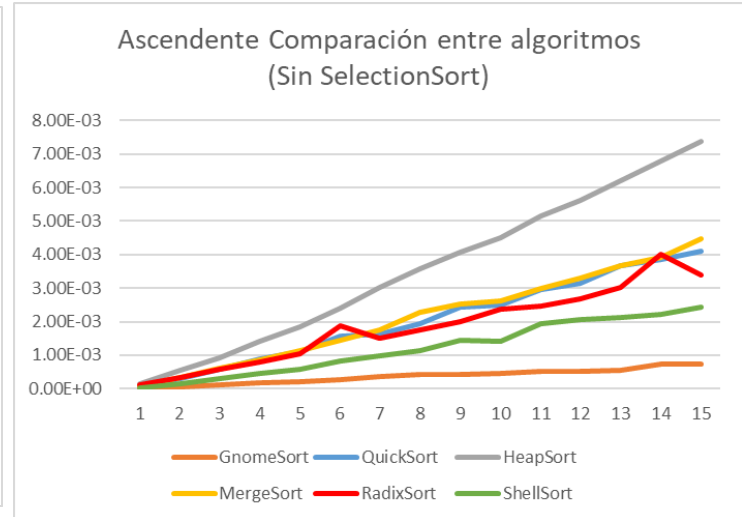
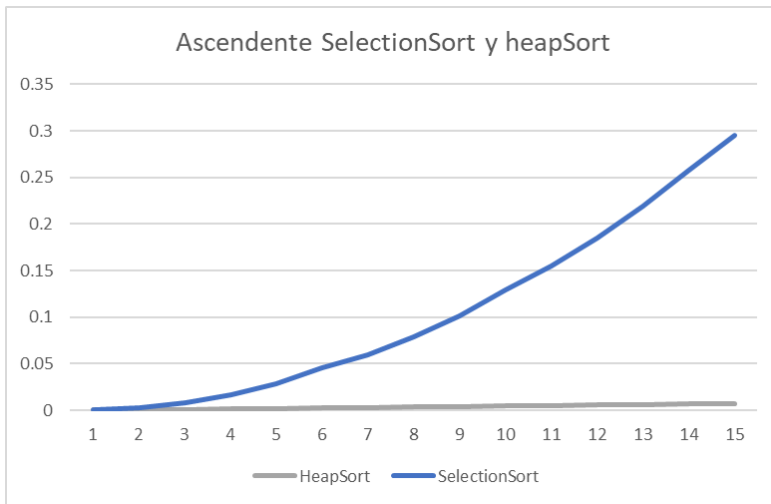
Esteban Enrique Cárcamo Urízar

23016

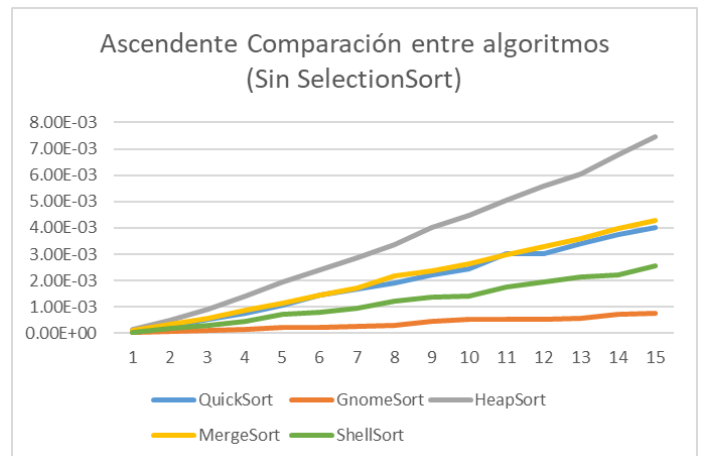
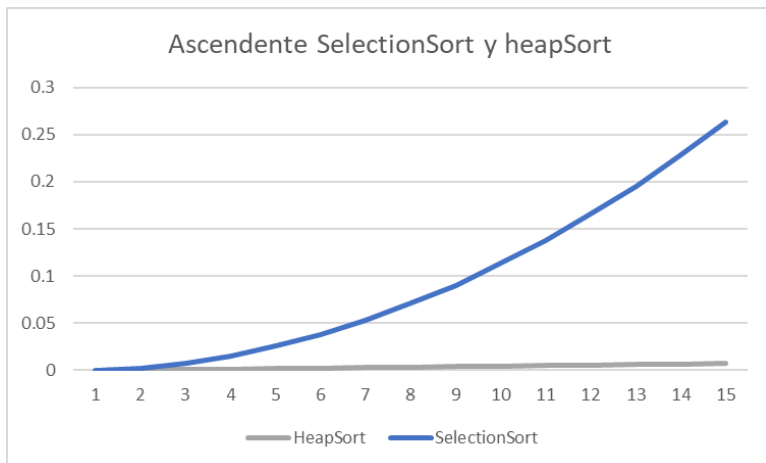
Algoritmos y Estructuras de Datos

Moisés Alonso

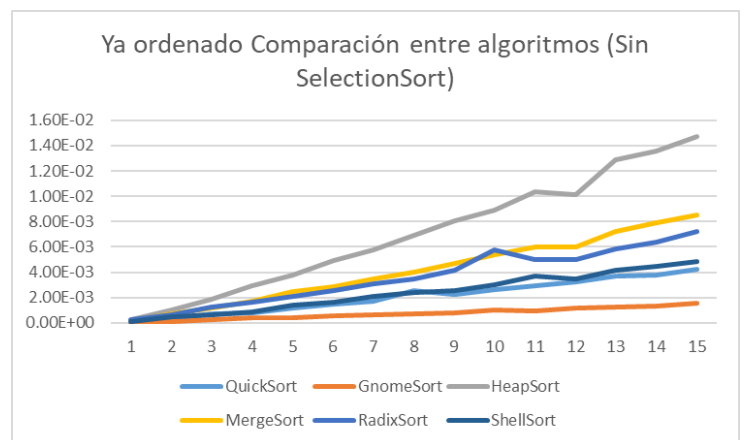
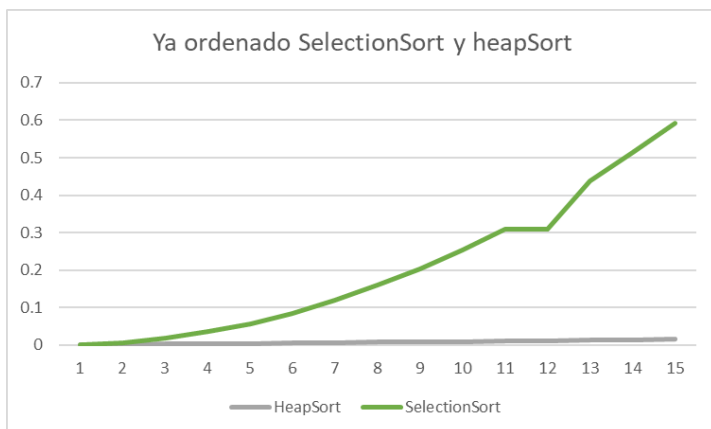
Gráficas comparando de forma ascendente cada sort



Gráficas comparando de forma descendente cada sort

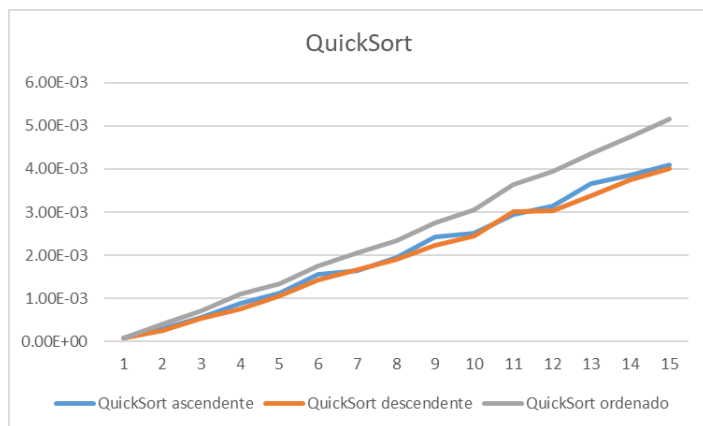


Gráficas comparando de forma ordenada, ordenando cada sort

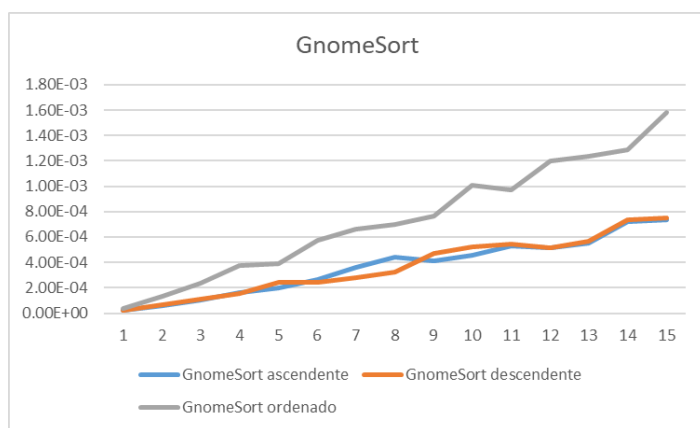


Comparación entre ascendente, descendente, teórico y de forma ordenada cada sort:

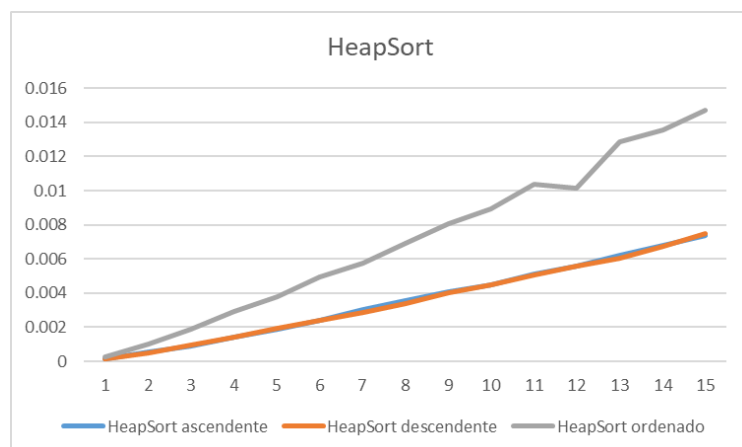
1. QuickSort



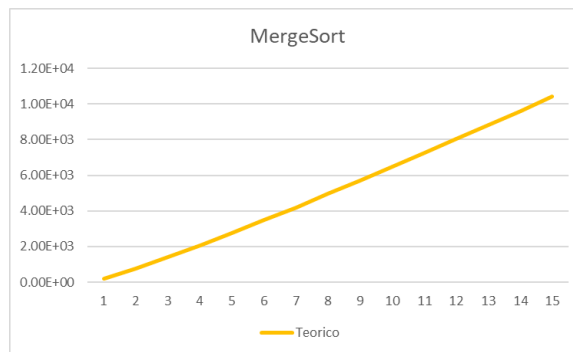
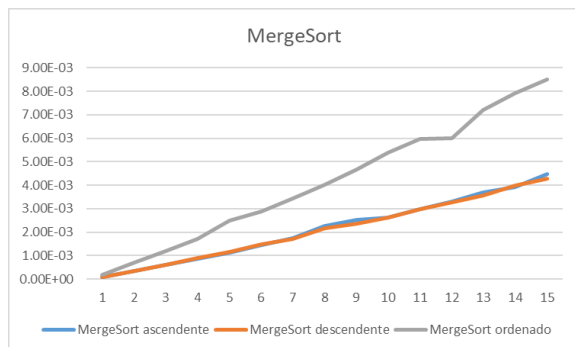
2. GnomeSort



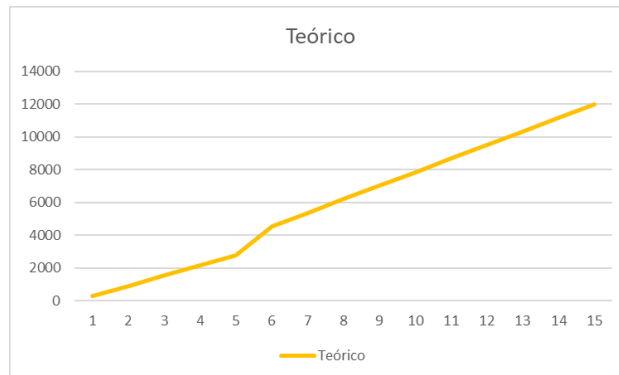
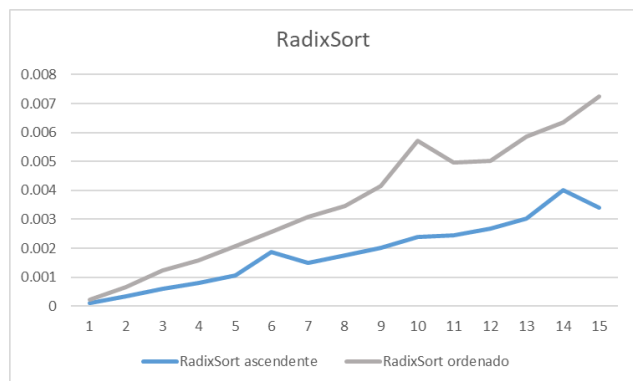
3. HeapSort



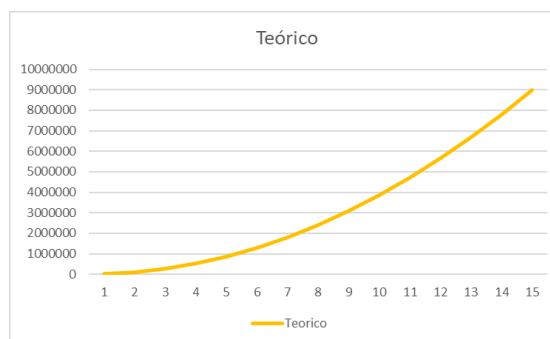
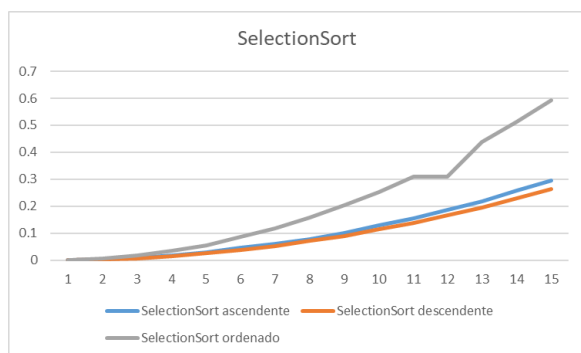
4. MergeSort



5. RadixSort



6. SelectionSort



7. ShellSort

