

Profiler Utilizado

Se utilizó el profiler que ya viene integrado con NetBeans. Este se empleó analizando únicamente la clase main. Entre los resultados nos mostraba cuanto tiempo se había tardado en procesar cada instrucción dentro del main, dichas instrucciones eran llamar a cada sort para que ordenara determinada cantidad de números.

Ejemplo:

Name	Total Time
main	9.21 ms (100%)
hojatrabajo3.SortMain. main (String[])	9.21 ms (100%)
Self time	8.46 ms (91.9%)
hojatrabajo3.RadixSort. radixsort (int[], int)	0.238 ms (2.6%)
hojatrabajo3.MergeSort. mergesort (int[])	0.218 ms (2.4%)
hojatrabajo3.QuickSort. sort (int[])	0.172 ms (1.9%)
hojatrabajo3.GnomeSort. gnomeSort (int[])	0.087 ms (0.9%)
hojatrabajo3.QuickSort. <init> ()	0.016 ms (0.2%)
hojatrabajo3.BurbujaSort. <init> ()	0.008 ms (0.1%)
hojatrabajo3.BurbujaSort. burbuja (int[])	0.008 ms (0.1%)

Resultados obtenidos

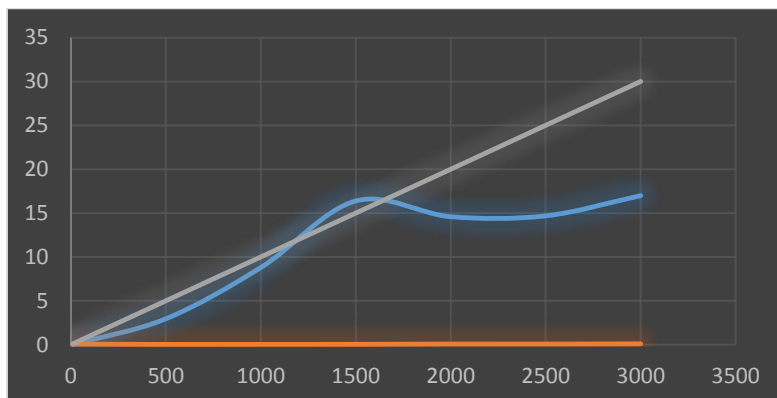
Cantidad de numeros		10	500	1000	1500	2000	2500	3000
GnomeSort	Desordenado	0.017	2.95	8.8	16.4	14.6	14.7	17
	Ordenado	0.087	0.025	0.032	0.043	0.088	0.086	0.12
	Teorico
BurbujaSort	Desordenado	0.017	2.3	6.38	9.34	10.9	9.77	26.2
	Ordenado	0.008	5.93	3.76	6.9	10.1	9.23	33.2
	Teorico
MergeSort	Desordenado	1.43	2.39	3.14	3.39	9.42	6.16	5.89
	Ordenado	0.218	2.57	3.6	4.19	8.5	5.97	8.37
	Teorico							
QuickSort	Desordenado	0.08	0.368	0.777	9.29	5.32	1.63	1.49
	Ordenado	0.172	0.46	0.595	1.33	1.48	1.15	2.12
	Teorico
RadixSort	Desordenado	0.232	0.364	1.1	1.56	2.2	2.39	2.73
	Ordenado	0.238	0.539	0.911	1.32	1.82	1.77	2.57
	Teorico

Gráficas

Azul	Desordenado
Anaranjado	Ordenado
Gris	Teórico

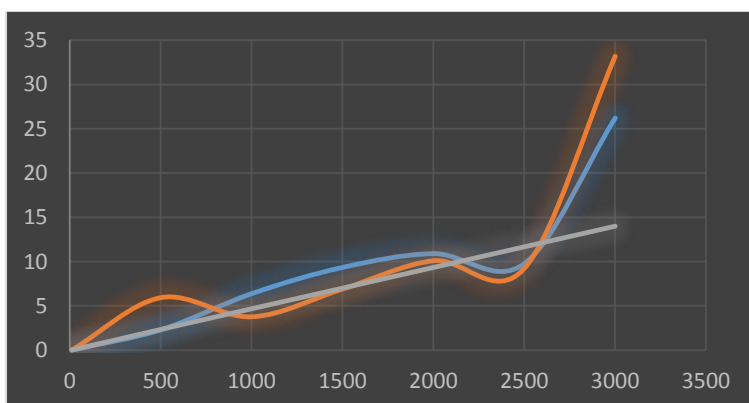
GnomeSort

Complejidad $O(n)$



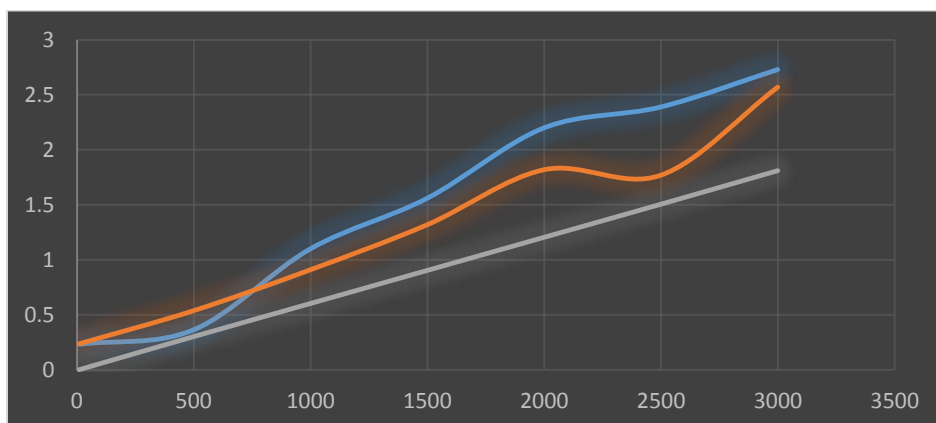
BurbujaSort

Complejidad $O(n^2)$



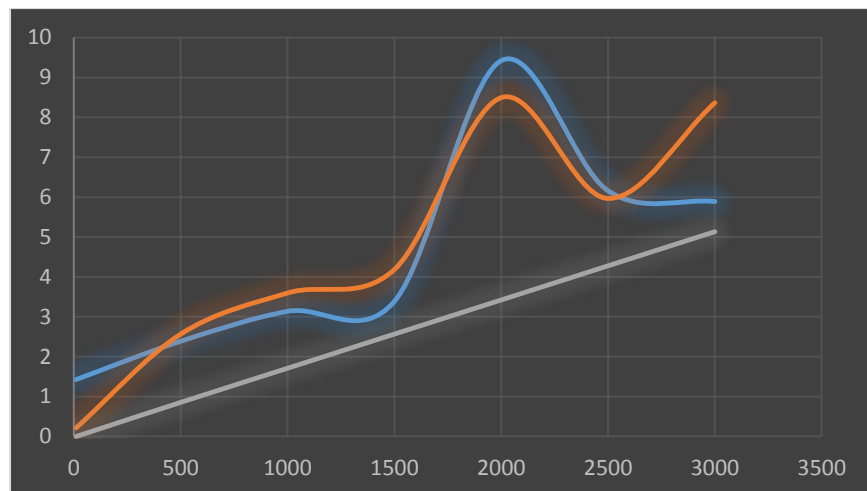
RadixSort

Complejidad $O(nk)$



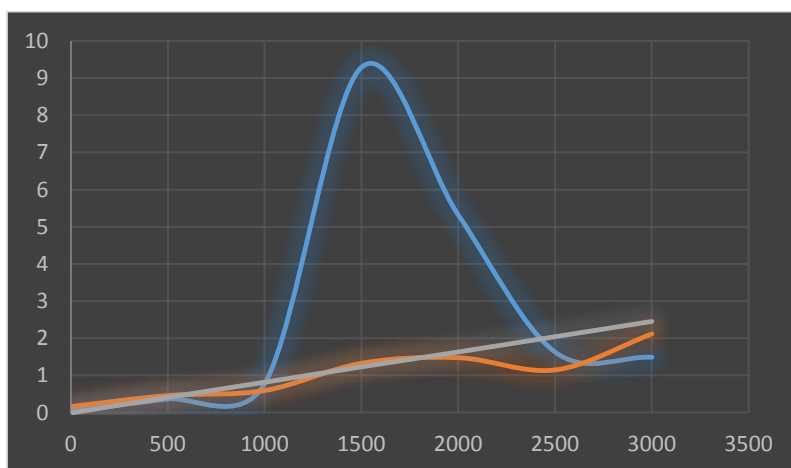
MergeSort

Complejidad de $O(n \log(n))$

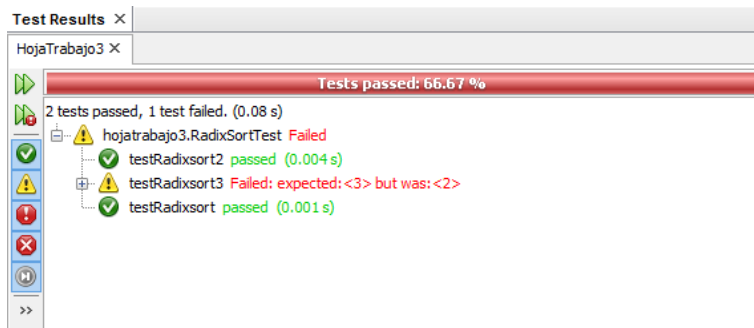
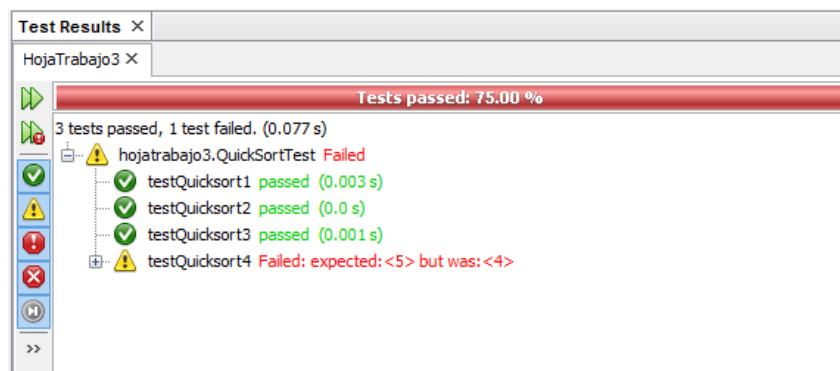
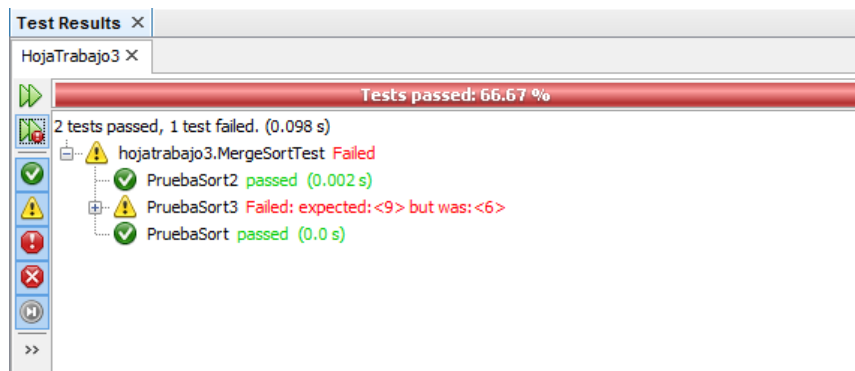
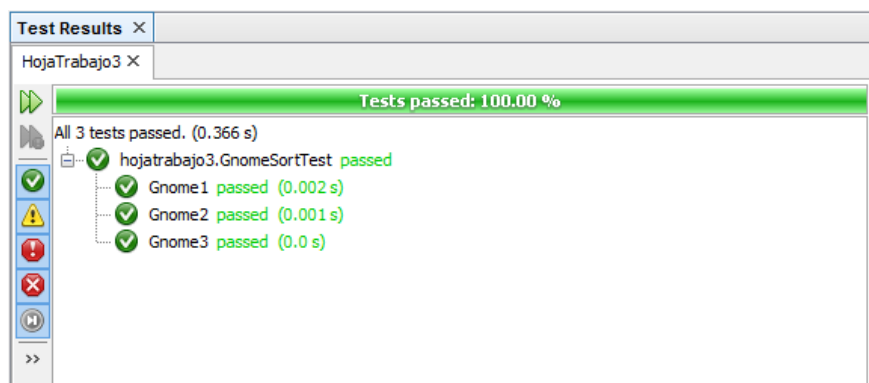


QuickSort

Complejidad de $O(n \log(n))$



Pruebas Unitarias



*Pruebas erróneas hechas intencionalmente