

Pablo Ruiz 18259

## Tiempo en correr los sorts pruebas de Junit y análisis del profiler

10 Elementos:

CPU profiling	Tracing	Call Tree	Time (ms)		Avg. Time (ms)
			Time (ms)	%	
CPU usage telemetry	Call tree – All threads merged	Main.java:31 java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	0.3	0 %	0.3
		Main.java:30 java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	0.3	0 %	0.3
		Main.java: java.io.FileOutputStream.write(byte[], int, int)	0.2	0 %	0.1
		Main.java:32 java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	0.2	0 %	0.2
		Main.java:21 SortMethods.radixSort(List)	0.1	0 %	0.1
		Main.java: java.io.FileOutputStream.write(byte[], int, int)	0.1	0 %	< 0.1
		Main.java:9 jdk.internal.loader.URLClassPath\$FileLoader.getResource(String, boolean)	0.1	0 %	0.1
		Main.java:19 jdk.internal.loader.URLClassPath\$FileLoader.getResource(String, boolean)	0.1	0 %	0.1
		Main.java:22 SortMethods.selectionSort(List)	< 0.1	0 %	< 0.1
		Main.java:20 SortMethods.mergeSort(List)	< 0.1	0 %	< 0.1
CPU usage telemetry	Call tree – By thread	Main.java: SortMethods.insertionSort(List)	< 0.1	0 %	< 0.1
		Main.java:23 SortMethods.quickSort(List, int, int)	< 0.1	0 %	< 0.1
		Main.java: java.nio.CharBuffer.wrap(char[], int, int)	< 0.1	0 %	< 0.1
		Main.java: sun.net.util.URLUtil.urlNoFragString(URL)	< 0.1	0 %	< 0.1
		Main.java:21 SortMethods.radixSort(List)	< 0.1	0 %	< 0.1
		Main.java:20 SortMethods.mergeSort(List)	< 0.1	0 %	< 0.1
		Main.java:22 SortMethods.selectionSort(List)	< 0.1	0 %	< 0.1
		Main.java:19 jdk.internal.loader.URLClassPath\$FileLoader.getResource(String, boolean)	< 0.1	0 %	< 0.1
		Main.java:9 jdk.internal.loader.URLClassPath\$FileLoader.getResource(String, boolean)	< 0.1	0 %	< 0.1
		Main.java: java.io.FileOutputStream.write(byte[], int, int)	< 0.1	0 %	< 0.1

375 Elementos:

CPU profiling	Tracing	Call Tree	Time (ms)		Avg. Time (ms)
			Time (ms)	%	
CPU usage telemetry	Call tree – All threads merged	<All threads>	1,046	100 %	
		com.intellij.rt.execution.application.AppMainV2\$1.run()	812	78 %	812
		Main.main(String[])	222	21 %	222
		Main.java: DocumentMethods.generateRandomData(int, String)	70	7 %	70
		Main.java:10 DocumentMethods.getDataArray(String)	51	5 %	51
		Main.java:22 SortMethods.selectionSort(List)	21	2 %	21
		Main.java: SortMethods.insertionSort(List)	20	2 %	20
		Main.java:23 SortMethods.quickSort(List, int, int)	18	2 %	18
		Main.java: java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	7	1 %	7
		Main.java:20 SortMethods.mergeSort(List)	6	1 %	6
		Main.java:11 DocumentMethods.getDataArray(String)	2	0 %	2
		Main.java:21 SortMethods.radixSort(List)	1	0 %	1
		Main.java:30 java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	0.8	0 %	0.8
		Main.java: jdk.internal.loader.Resource.getByteBuffer()	0.6	0 %	0.3
		Main.java:22 SortMethods.selectionSort(List)	0.5	0 %	0.2

750 Elementos:

CPU profiling	Tracing	Call Tree	Time (ms)		Avg. Time (ms)
			Time (ms)	%	
CPU usage telemetry	Call tree – All threads merged	<All threads>	1,099	100 %	
		com.intellij.rt.execution.application.AppMainV2\$1.run()	794	72 %	794
		Main.main(String[])	293	27 %	293
		Main.java: DocumentMethods.generateRandomData(int, String)	66	6 %	66
		Main.java:23 SortMethods.quickSort(List, int, int)	59	5 %	59
		Main.java:10 DocumentMethods.getDataArray(String)	54	5 %	54
		Main.java:22 SortMethods.selectionSort(List)	51	5 %	51
		Main.java: SortMethods.insertionSort(List)	30	3 %	30
		Main.java:20 SortMethods.mergeSort(List)	8	1 %	8
		Main.java:21 SortMethods.radixSort(List)	3	0 %	3
		Main.java:11 DocumentMethods.getDataArray(String)	2	0 %	2
		Main.java: java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	1	0 %	1
		Main.java: jdk.internal.loader.Resource.getByteBuffer()	0.4	0 %	0.2
		Main.java:32 java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	0.4	0 %	0.4
		Main.java:22 SortMethods.selectionSort(List)	0.3	0 %	0.1

1125 Elementos:

CPU profiling	Call Tree		Time (ms)	Avg. Time (ms)
			↓	
Tracing	<All threads>		1,502 100 %	
CPU usage telemetry	com.intellij.rt.execution.application.AppMainV2\$1.run()		1,043 69 %	1,043
Call tree – All threads merged	Main.main(String[])		440 29 %	440
Call tree – By thread	Main.java:23 DocumentMethods.generateRandomData(int, String)		92 6 %	92
Flame graph	Main.java:23 SortMethods.quickSort(List, int, int)		91 6 %	91
Hot spots	Main.java:22 SortMethods.selectionSort(List)		80 5 %	80
Method list	Main.java:10 DocumentMethods.getDataArray(String)		68 5 %	68
Java EE	Main.java:20 SortMethods.insertionSort(List)		61 4 %	61
	Main.java:20 SortMethods.mergeSort(List)		21 1 %	21
Database	Main.java:21 SortMethods.radixSort(List)		6 0 %	6
	Main.java:11 DocumentMethods.getDataArray(String)		2 0 %	2
	Main.java: java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])		1 0 %	1
	Main.java: jdk.internal.loader.Resource.getByteBuffer()		0.6 0 %	0.3
	Main.java:32 java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])		0.5 0 %	0.5

1500 Elementos:

CPU profiling	Call Tree		Time (ms)	Avg. Time (ms)
			↓	
Tracing	<All threads>		1,584 100 %	
CPU usage telemetry	com.intellij.rt.execution.application.AppMainV2\$1.run()		1,071 68 %	1,071
Call tree – All threads merged	Main.main(String[])		500 32 %	500
Call tree – By thread	Main.java:23 SortMethods.quickSort(List, int, int)		110 7 %	110
Flame graph	Main.java: SortMethods.insertionSort(List)		101 6 %	101
Hot spots	Main.java:22 SortMethods.selectionSort(List)		83 5 %	83
Method list	Main.java: DocumentMethods.generateRandomData(int, String)		79 5 %	79
Java EE	Main.java:10 DocumentMethods.getDataArray(String)		70 4 %	70
	Main.java:20 SortMethods.mergeSort(List)		19 1 %	19
Database	Main.java:21 SortMethods.radixSort(List)		10 1 %	10
	Main.java:11 DocumentMethods.getDataArray(String)		7 0 %	7
	Main.java: java.nio.CharBuffer.arrayOffset()		1 0 %	< 0.1
	Main.java: java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])		0.9 0 %	0.9
	Main.java: jdk.internal.loader.Resource.getByteBuffer()		0.5 0 %	0.2

1875 Elementos:

CPU profiling	Call Tree		Time (ms)	Avg. Time (ms)
			↓	
Tracing	<All threads>		2,087 100 %	
CPU usage telemetry	com.intellij.rt.execution.application.AppMainV2\$1.run()		1,299 62 %	1,299
Call tree – All threads merged	Main.main(String[])		774 37 %	774
Call tree – By thread	Main.java:22 SortMethods.selectionSort(List)		203 10 %	203
Flame graph	Main.java:23 SortMethods.quickSort(List, int, int)		188 9 %	188
Hot spots	Main.java: SortMethods.insertionSort(List)		139 7 %	139
Method list	Main.java: DocumentMethods.generateRandomData(int, String)		109 5 %	109
Java EE	Main.java:10 DocumentMethods.getDataArray(String)		68 3 %	68
	Main.java:20 SortMethods.mergeSort(List)		29 1 %	29
Database	Main.java:21 SortMethods.radixSort(List)		12 1 %	12
	Main.java:11 DocumentMethods.getDataArray(String)		4 0 %	4
	Main.java: java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])		1 0 %	1
	Main.java:32 java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])		0.8 0 %	0.8
	Main.java: jdk.internal.loader.Resource.getByteBuffer()		0.6 0 %	0.3

2250 Elementos:

CPU profiling	Call Tree	Time (ms)	Avg. Time (ms)
Trading	<All threads>	1,711 100 %	
CPU usage telemetry	com.intellij.rt.execution.application.AppMainV2\$1.run()	1,056 62 %	1,056
Call tree – All threads merged	Main.main(String[])	643 38 %	643
Call tree – By thread	Main.java:23 SortMethods.quickSort(List, int, int)	205 12 %	205
Flame graph	Main.java DocumentMethods.generateRandomData(int, String)	113 7 %	113
Hot spots	Main.java SortMethods.insertionSort(List)	99 6 %	99
Method list	Main.java:22 SortMethods.selectionSort(List)	94 5 %	94
Java EE	Main.java:10 DocumentMethods.getDataArray(String)	70 4 %	70
Database	Main.java:20 SortMethods.mergeSort(List)	24 1 %	24
	Main.java:21 SortMethods.radixSort(List)	14 1 %	14
	Main.java:11 DocumentMethods.getDataArray(String)	4 0 %	4
	Main.java java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	0.9 0 %	0.9
	Main.java jdk.internal.loader.Resource.getByteBuffer()	0.5 0 %	0.2
	Main.java:31 java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	0.4 0 %	0.4

2625 Elementos:

CPU profiling	Call Tree	Time (ms)	Avg. Time (ms)
Trading	<All threads>	1,292 100 %	
CPU usage telemetry	com.intellij.rt.execution.application.AppMainV2\$1.run()	789 61 %	789
Call tree – All threads merged	Main.main(String[])	484 37 %	484
Call tree – By thread	Main.java:23 SortMethods.quickSort(List, int, int)	197 15 %	197
Flame graph	Main.java SortMethods.insertionSort(List)	77 6 %	77
Hot spots	Main.java:22 SortMethods.selectionSort(List)	69 5 %	69
Method list	Main.java DocumentMethods.generateRandomData(int, String)	56 4 %	56
Java EE	Main.java:10 DocumentMethods.getDataArray(String)	48 4 %	48
Database	Main.java:20 SortMethods.mergeSort(List)	10 1 %	10
	Main.java:21 SortMethods.radixSort(List)	7 1 %	7
	Main.java:11 DocumentMethods.getDataArray(String)	3 0 %	3
	Main.java java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	0.5 0 %	0.5
	Main.java jdk.internal.loader.Resource.getByteBuffer()	0.4 0 %	0.2
	Main.java:29 java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	0.3 0 %	0.3

3000 Elementos: Muestra los tiempos de ejecución de los sorts en un conjunto de datos desordenados (blanco) y en el mismo conjunto ordenado (verde).

CPU profiling	Call Tree	Time (ms)	Avg. Time (ms)
Trading	Main.java:22 SortMethods.selectionSort(List)	277 14 %	277
CPU usage telemetry	Main.java:23 SortMethods.quickSort(List, int, int)	171 9 %	171
Call tree – All threads merged	Main.java SortMethods.insertionSort(List)	98 5 %	98
Call tree – By thread	Main.java DocumentMethods.generateRandomData(int, String)	72 4 %	72
Flame graph	Main.java:34 SortMethods.insertionSort(List)	68 4 %	68
Hot spots	Main.java:37 SortMethods.selectionSort(List)	44 2 %	44
Method list	Main.java:38 SortMethods.quickSort(List, int, int)	37 2 %	37
Java EE	Main.java:10 DocumentMethods.getDataArray(String)	31 2 %	31
Database	Main.java:20 SortMethods.mergeSort(List)	16 1 %	16
	Main.java:21 SortMethods.radixSort(List)	8 0 %	8
	Main.java:36 SortMethods.radixSort(List)	6 0 %	6
	Main.java:35 SortMethods.mergeSort(List)	5 0 %	5
	Main.java:11 DocumentMethods.getDataArray(String)	4 0 %	4
	Main.java java.lang.invoke.StringConcatFactory.makeConcatWithConstants(MethodHandles\$Lookup, String, MethodType, String, Object[])	0.5 0 %	0.5

Tabla 1. Tiempo de corrida de sorts según el número de datos

	Tiempo de corrida en milisegundos				
Número de datos	Merge	Quick	Insertion	Radix	Selection
10	0.1	0.1	0.1	0.1	0.1
375	6	18	20	1	21
750	8	59	30	3	51
1125	21	91	61	6	80
1500	19	110	101	10	83
1875	29	188	139	12	203
2250	24	205	99	14	94
2625	10	197	77	7	69
3000	16	171	98	8	277

Figura 1. Tiempo de corrida de sorts

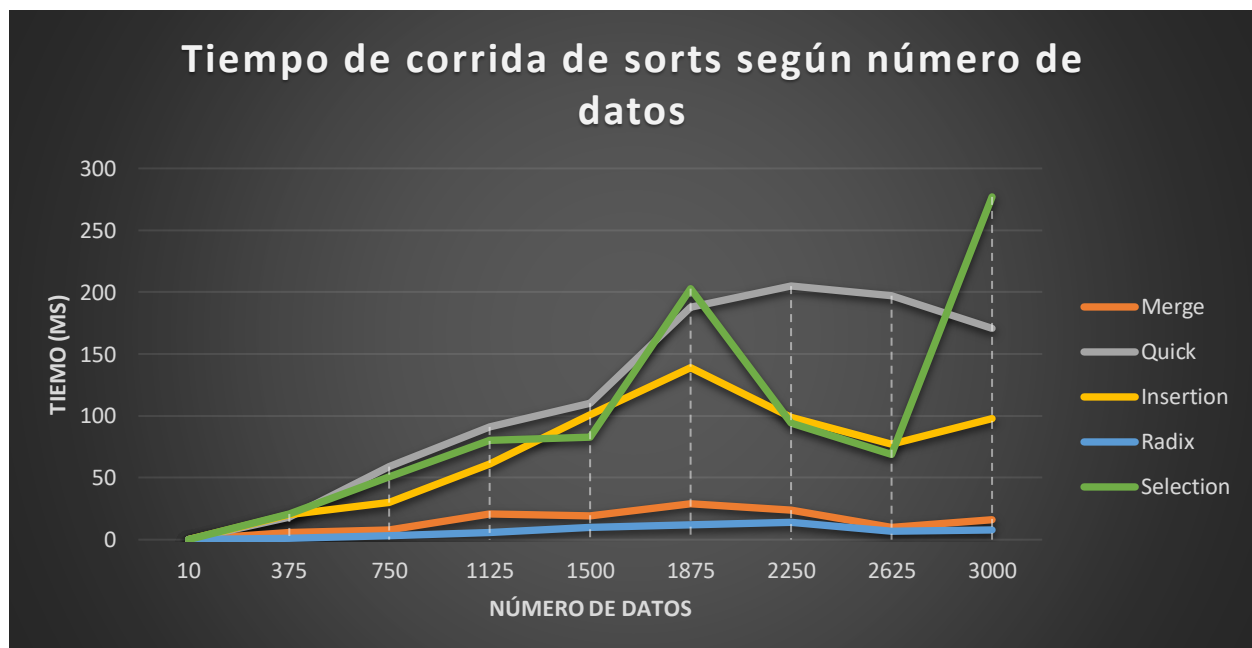
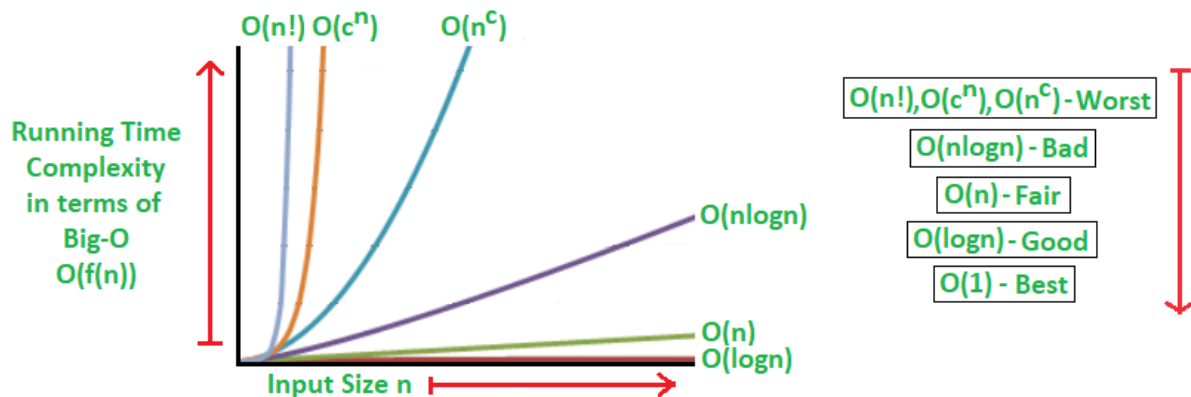


Figura 2. Tiempo de corrida teórico de sorts



Tomado de GeeksForGeeks <https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/>

En este caso, el merge sort y Quicksort deberían de tener una forma de  $O(n \log n)$ , el Selection e Insertion de  $O(n^2)$  y el radix de  $O(n)$ .

### Análisis del profiler

El profiler utilizado fue Yourkit. Se empleó a través de la descarga de un plugin para IntelliJ, en donde se permite realizar un profiling del código directamente desde el IDE. Para ello, se descargó Yourkit desde su sitio web (<https://www.yourkit.com/>) y se configuró en la máquina. Luego, desde la aplicación instalada, se configuró el plugin para IntelliJ. Finalmente, desde IntelliJ se configuró para que este hiciera Tracing del CPU. Se utilizó a través de IntelliJ, en donde solamente se le daba click al botón de profiling y se abría el Profiler. Al abrirse, se abría el snapshot creado y se selecciona los métodos en la pestaña izquierda del programa. Al abrir esto, se podían visualizar los tiempos de corrida.

Figura 3. Muestra de pruebas aprobadas de los sorts en Junit

