

Universidad del Valle de Guatemala

Hoja de Trabajo 8

Irene Molina 13480

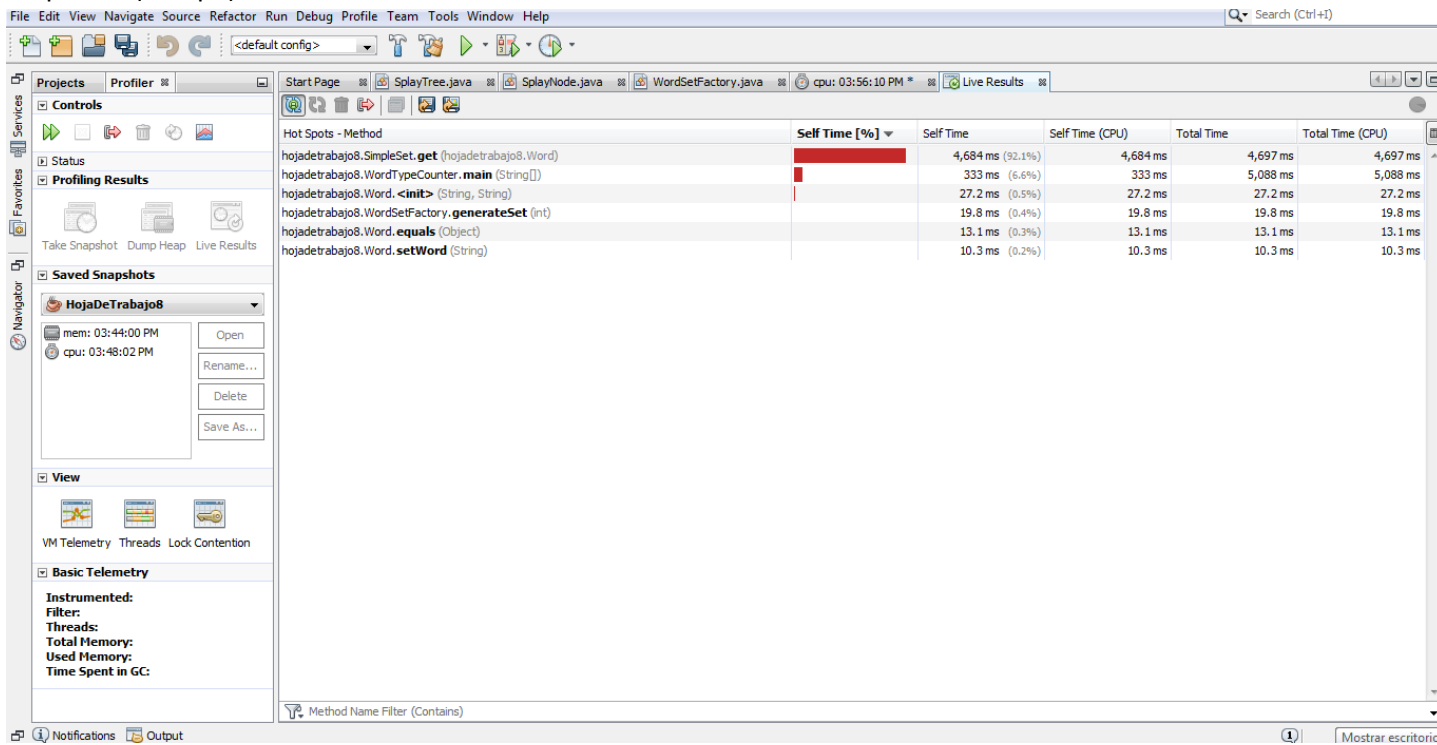
Jorge Garcia 13175

Moises Urias 13015

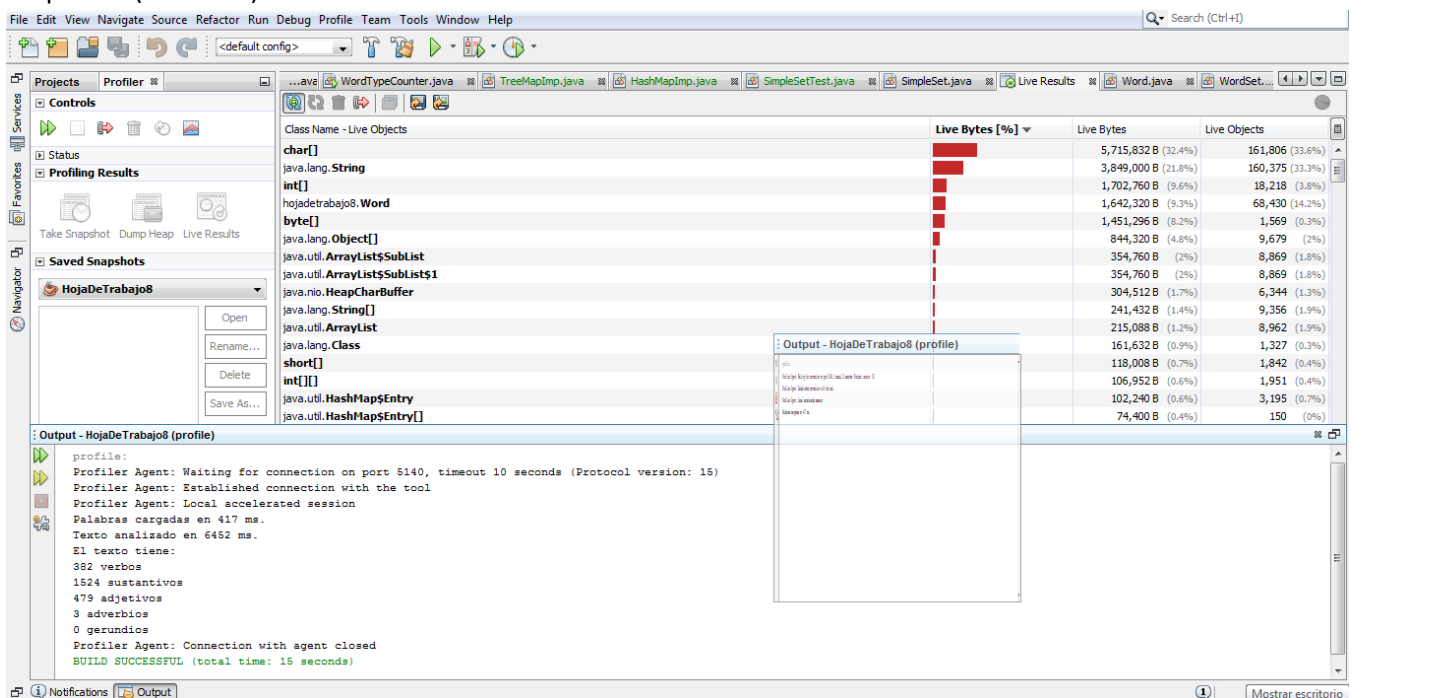
<https://github.com/mol13480/HojaDeTrabajo8>

## Resultados Profiler

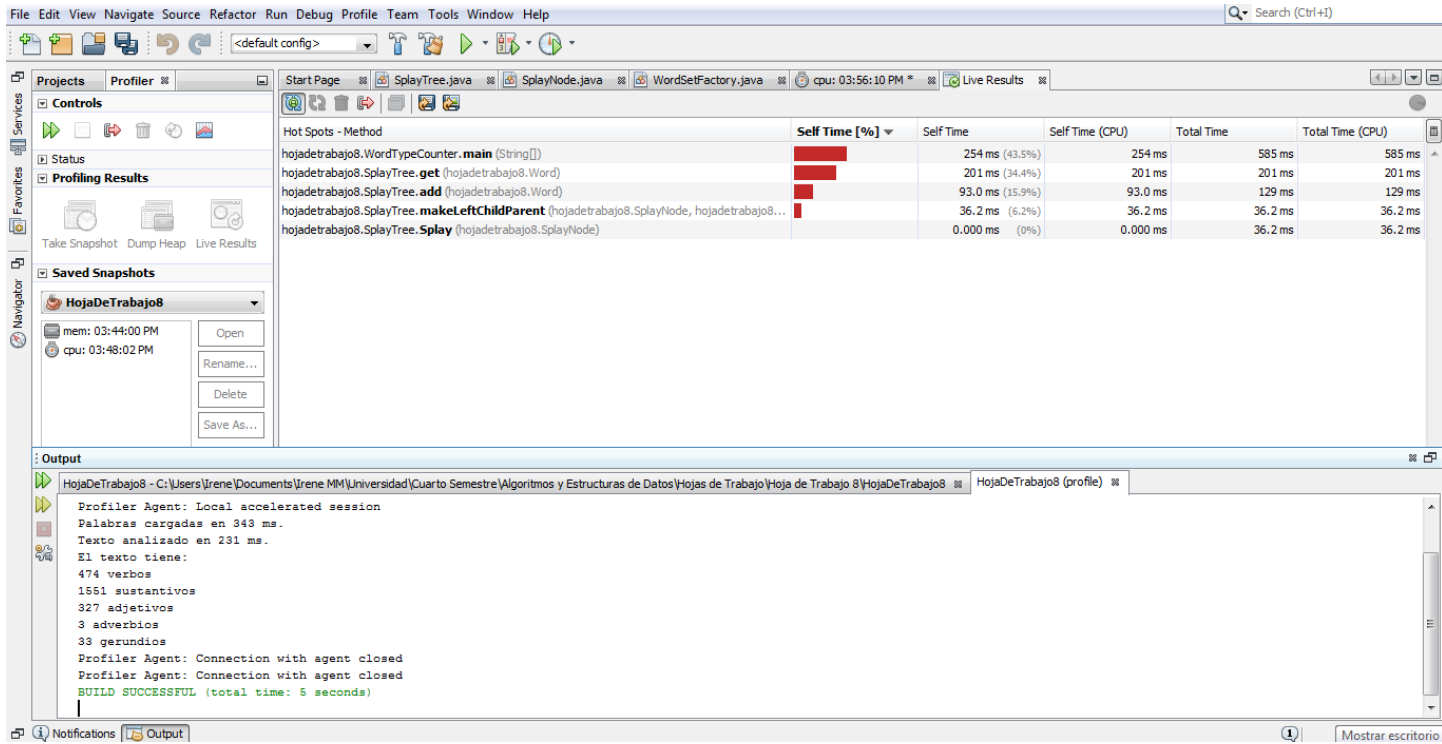
### 1. Simple Set (tiempo)



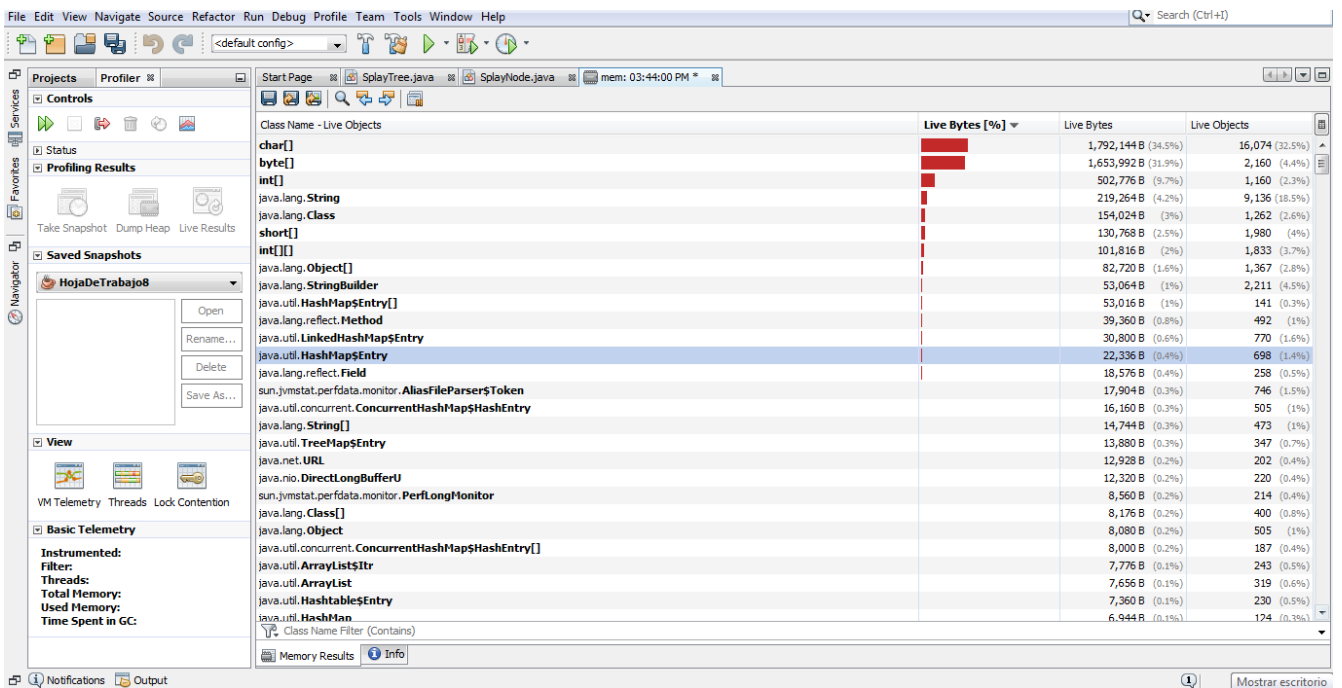
### Simple Set (memoria)



## 2. Splay Tree (tiempo)



## Splay Tree (memoria)



### 3. Hash Map (tiempo)

The screenshot shows the IntelliJ IDEA Profiler interface. The 'Hot Spots - Method' table lists the following methods and their performance metrics:

Hot Spots - Method	Self Time [%]	Self Time	Self Time (CPU)	Total Time	Total Time (CPU)
hojadetrabajo8.HashMapImp.get (hojadetrabajo8.Word)		637 ms	637 ms	637 ms	637 ms
hojadetrabajo8.WordTypeCounter.main (String[])		377 ms (35.2%)	377 ms	1,073 ms	1,073 ms
hojadetrabajo8.HashMapImp.add (hojadetrabajo8.Word)		58.4 ms (5.4%)	58.4 ms	58.4 ms	58.4 ms

The 'Output' window shows the following text:

```
:telecentro null
:trece null
Texto analizado en 722 ms.
El texto tiene:
493 verbos
1577 sustantivos
273 adjetivos
3 adverbios
42 gerundios
Profiler Agent: Connection with agent closed
Profiler Agent: Connection with agent closed
BUILD SUCCESSFUL (total time: 5 seconds)
```

### Hash Map (memoria)

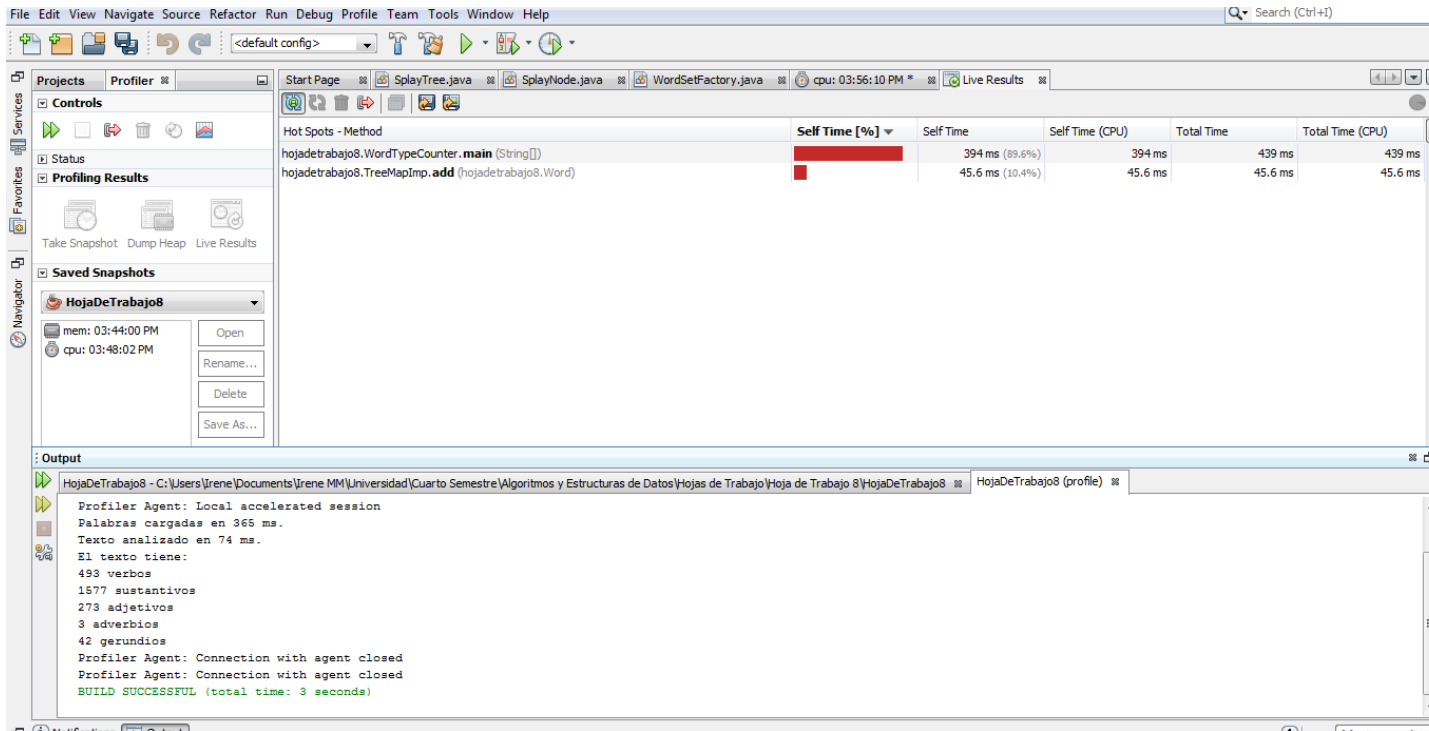
The screenshot shows the IntelliJ IDEA Profiler interface. The 'Class Name - Live Objects' table lists the following classes and their memory usage:

Class Name - Live Objects	Live Bytes [%]	Live Bytes	Live Objects
char[]		5,678,864 B (26.5%)	165,821 (28.2%)
java.lang.String		3,977,784 B (18.5%)	165,741 (28.1%)
java.nio.HeapCharBuffer		2,270,880 B (10.6%)	47,310 (8%)
java.util.HashMap\$Entry		1,598,432 B (7.5%)	49,951 (8.5%)
byte[]		1,434,640 B (6.7%)	1,561 (0.3%)
java.lang.Object[]		1,347,336 B (6.3%)	24,253 (4.1%)
java.util.ArrayList\$SubList\$1		945,480 B (4.4%)	23,637 (4%)
java.util.ArrayList\$SubList		945,480 B (4.4%)	23,637 (4%)
java.lang.String[]		582,040 B (2.7%)	24,105 (4.1%)
java.util.ArrayList		569,088 B (2.7%)	23,712 (4%)
hojadetrabajo8.Word		567,288 B (2.6%)	23,637 (4%)
java.util.HashMap\$Entry[]		442,928 B (2.1%)	143 (0%)
int[]		391,888 B (1.8%)	1,808 (0.3%)
java.lang.Class		161,520 B (0.8%)	1,326 (0.2%)
short[]		117,920 B (0.5%)	1,840 (0.3%)
int[][]		106,928 B (0.5%)	1,950 (0.3%)

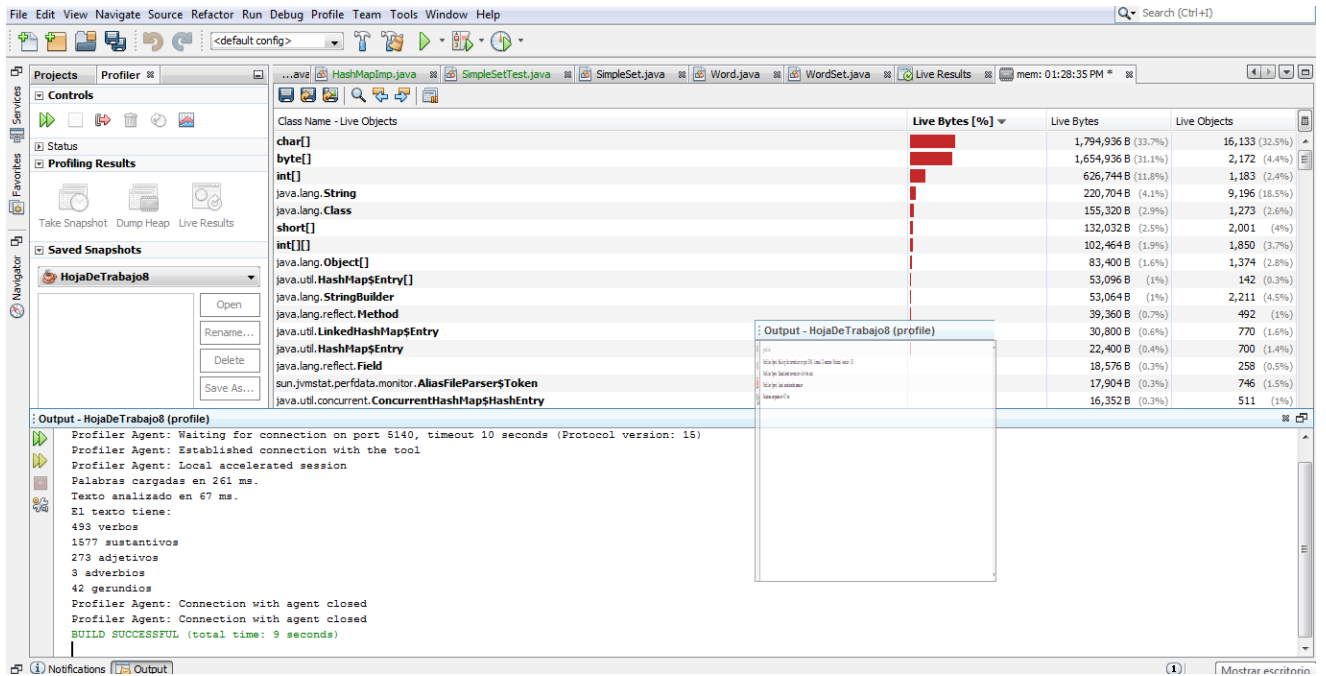
The 'Output' window shows the following text:

```
abolitionists n
abductions n
abbes n
Palabras cargadas en 6765 ms.
Texto analizado en 71 ms.
El texto tiene:
493 verbos
1577 sustantivos
273 adjetivos
3 adverbios
42 gerundios
Profiler Agent: Connection with agent closed
Profiler Agent: Connection with agent closed
BUILD SUCCESSFUL (total time: 11 seconds)
```

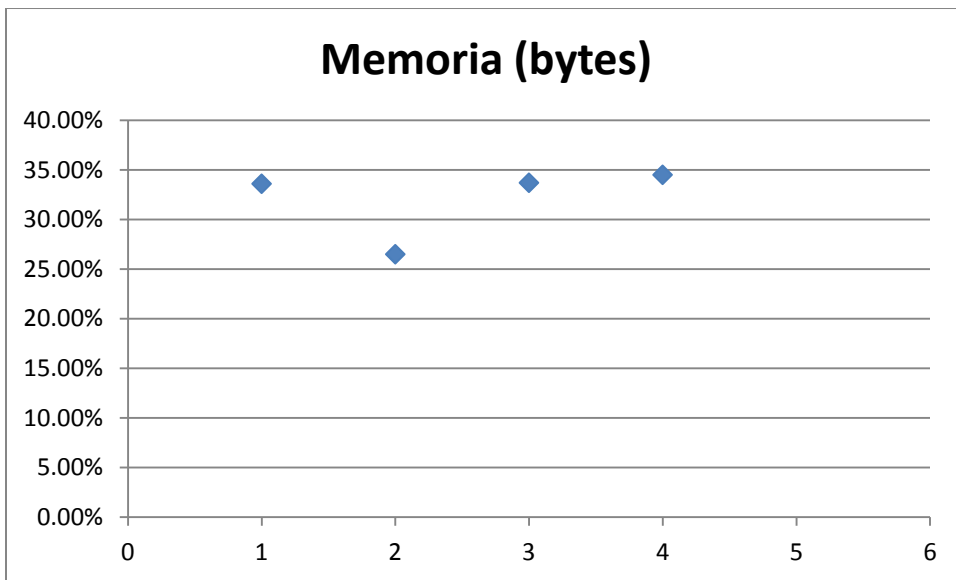
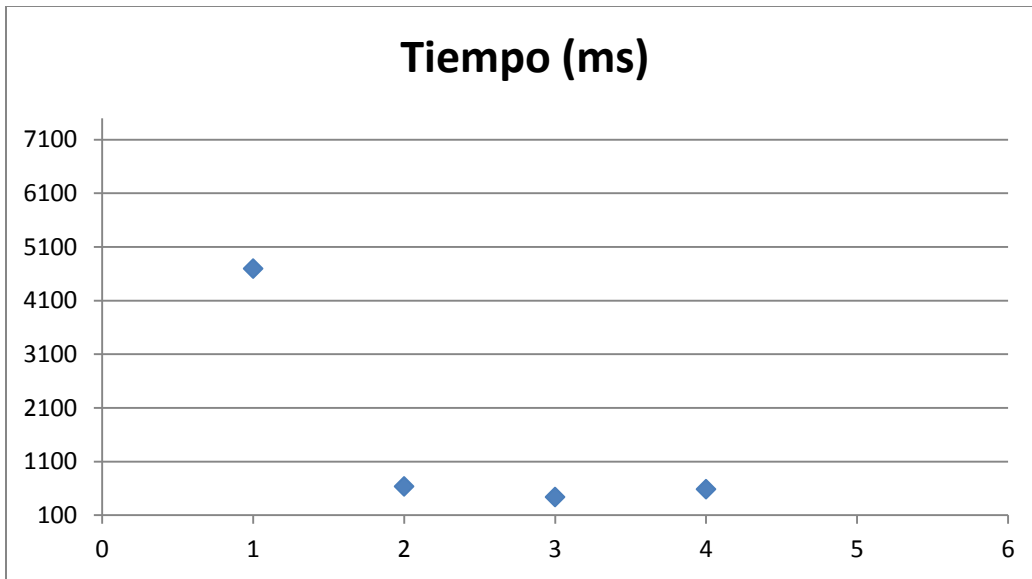
#### 4. Tree Map (tiempo)



#### Tree Map (memoria)



Implementación	Complejidad O() (add/get)	Tiempo de ejecución (ms)	Memoria (bytes)
1. Simple Set	O(1)/O(1)	4697	33.60%
2. Hash Map	O(n)/O(n)	637	26.50%
3. Tree map	O(nlog n)/ O(nlog n)	439	33.70%
4. Splay Tree	O(log n)/ O(log n)	585	34.5%
5. Red Black Tree	O(log n)/ O(log n)	--	--



Como se puede observar en la tabla de comparación y las gráficas anteriores, los menores tiempos de ejecución los tuvieron las implementaciones Hash Map, Tree Map y Splay Tree. Entre esas implementaciones, las que tienen menor complejidad son Splay Tree y Tree map. También se puede observar que estas dos tuvieron los menores tiempos de corrida utilizando el profiler. En cuanto a memoria utilizada, el Hash Table utilizó el menor porcentaje y los otros utilizaron porcentajes cercanos de memoria. Por lo tanto, a partir de estos resultados se puede observar que las dos mejores implementaciones en cuanto a memoria y tiempo de corrida son el Splay Tree y el Tree map.