

Tiempos Obtenidos con Profiler

Opción 5 HashMap

```
✓ m ██████████ 100.0% - 1,142 ms - 1 inv. com.intellij.rt.junit.JUnitStarter.main
> m ██████████ 99.3% - 1,133 ms - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs
> m ██████████ 0.7% - 8,007 µs - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.createListeners
```

Opción 5 TreeHashMap

```
✓ m ██████████ 100.0% - 644 ms - 1 inv. com.intellij.rt.junit.JUnitStarter.main
> m ██████████ 99.0% - 637 ms - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs
> m ██████████ 1.0% - 6,458 µs - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.createListeners
```

Opción 5 LinkedHashMap

```
✓ m ██████████ 100.0% - 529 ms - 1 inv. com.intellij.rt.junit.JUnitStarter.main
> m ██████████ 99.2% - 525 ms - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs
> m ██████████ 0.7% - 3,931 µs - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.createListeners
```

Opción 6 HashMap

```
✓ m ██████████ 100.0% - 6,872 ms - 1 inv. com.intellij.rt.junit.JUnitStarter.main
> m ██████████ 99.9% - 6,868 ms - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs
> m ██████████ 0.1% - 3,860 µs - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.createListeners
```

Opción 6 TreeHashMap

```
✓ m ██████████ 100.0% - 3,708 ms - 1 inv. com.intellij.rt.junit.JUnitStarter.main
> m ██████████ 99.6% - 3,692 ms - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs
> m ██████████ 0.4% - 16,237 µs - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.createListeners
```

Opción 6 LinkedHashMap

```
✓ m ██████████ 100.0% - 8,674 ms - 1 inv. com.intellij.rt.junit.JUnitStarter.main
> m ██████████ 100.0% - 8,670 ms - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs
> m ██████████ 0.0% - 3,681 µs - 1 inv. com.intellij.junit4.JUnit4IdeaTestRunner.createListeners
```

La complejidad de HashMap es constante debido a al KeySet()

Análisis de Resultados

El Map que menos se tardó únicamente imprimiendo los datos sin ordenar fue el LinkedHashMap y el peor fue HashMap. Pero en la opción 6 como se utiliza un arreglo y si utilizamos Big O, vemos que la complejidad de todos los mapas queda opacados por la complejidad $O(n^2)$ del sort de burbuja que se utilizo. A diferencia de la opción 5 el que más se tardo fue el LinkedHashMap y el que menos se tardo fue el TreeMap.