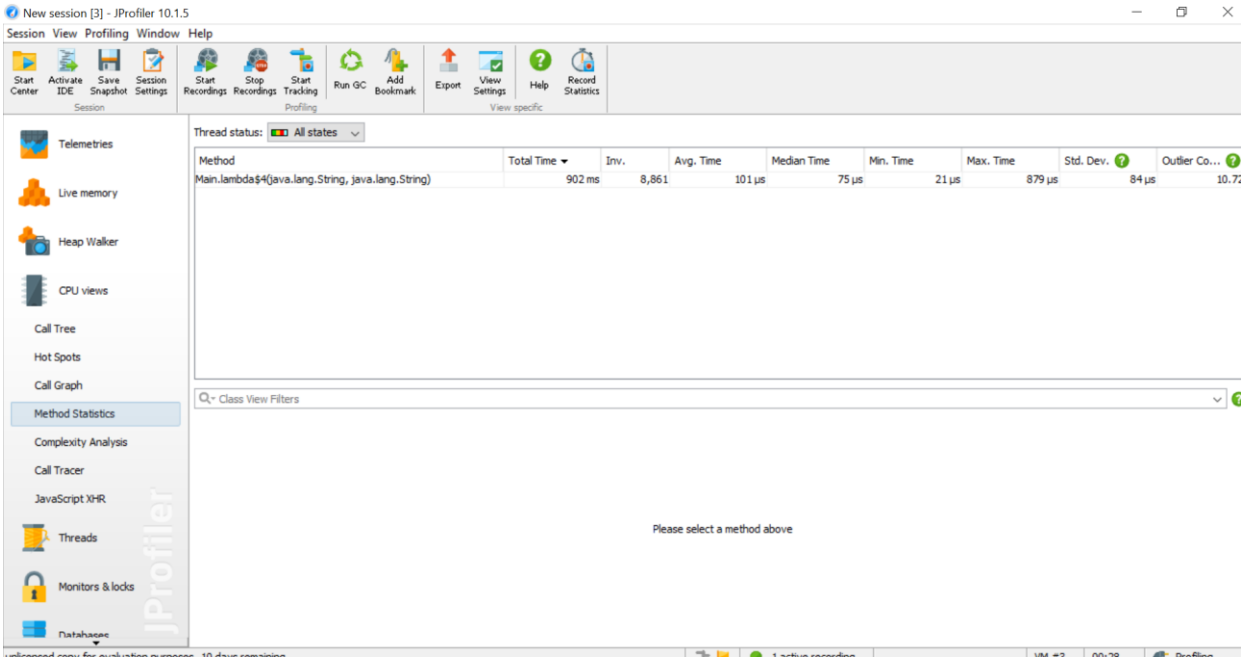
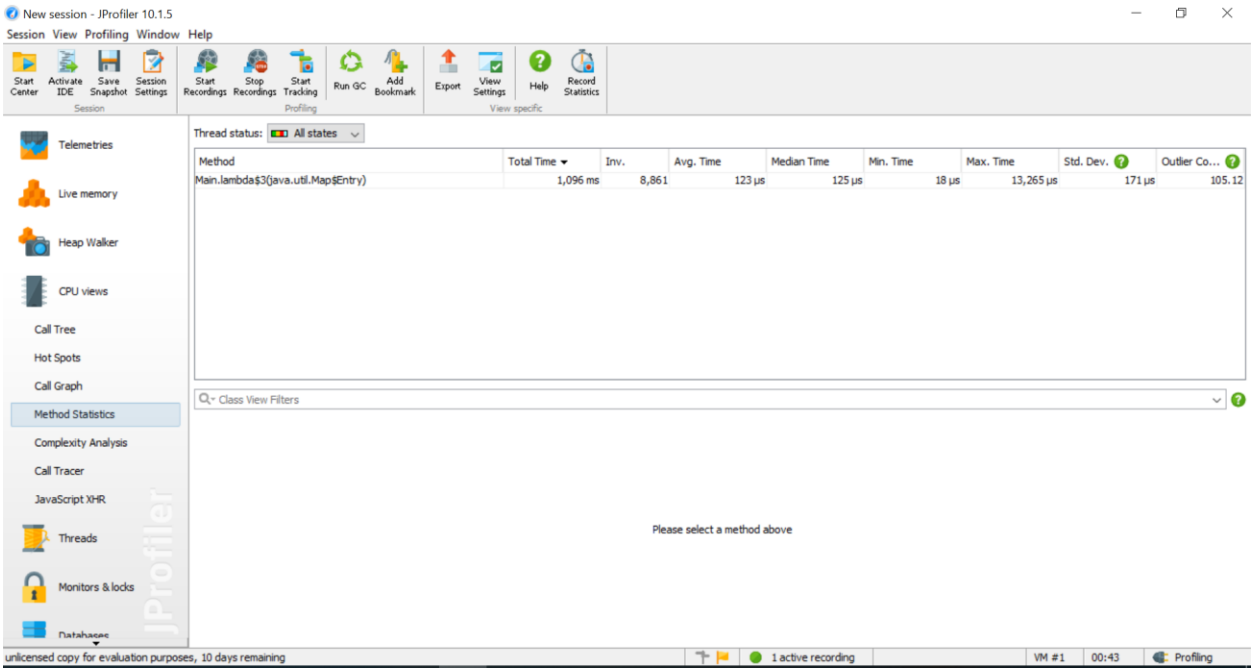
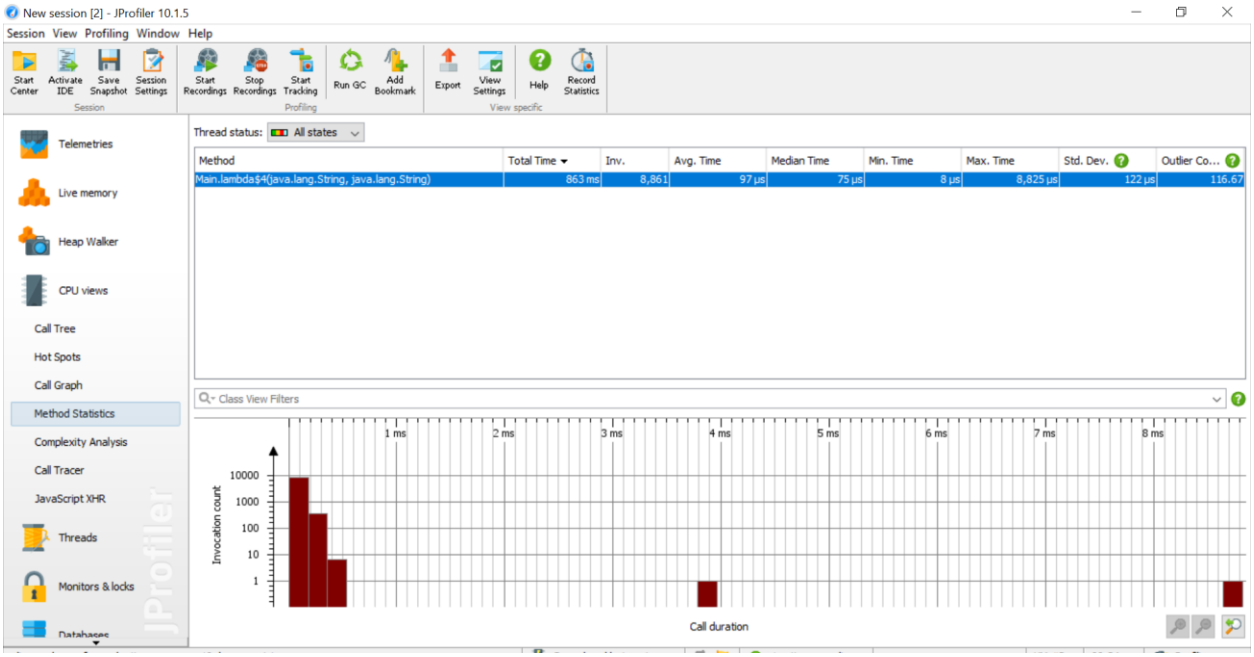
1. HASHMAP
   Solo imprimirla

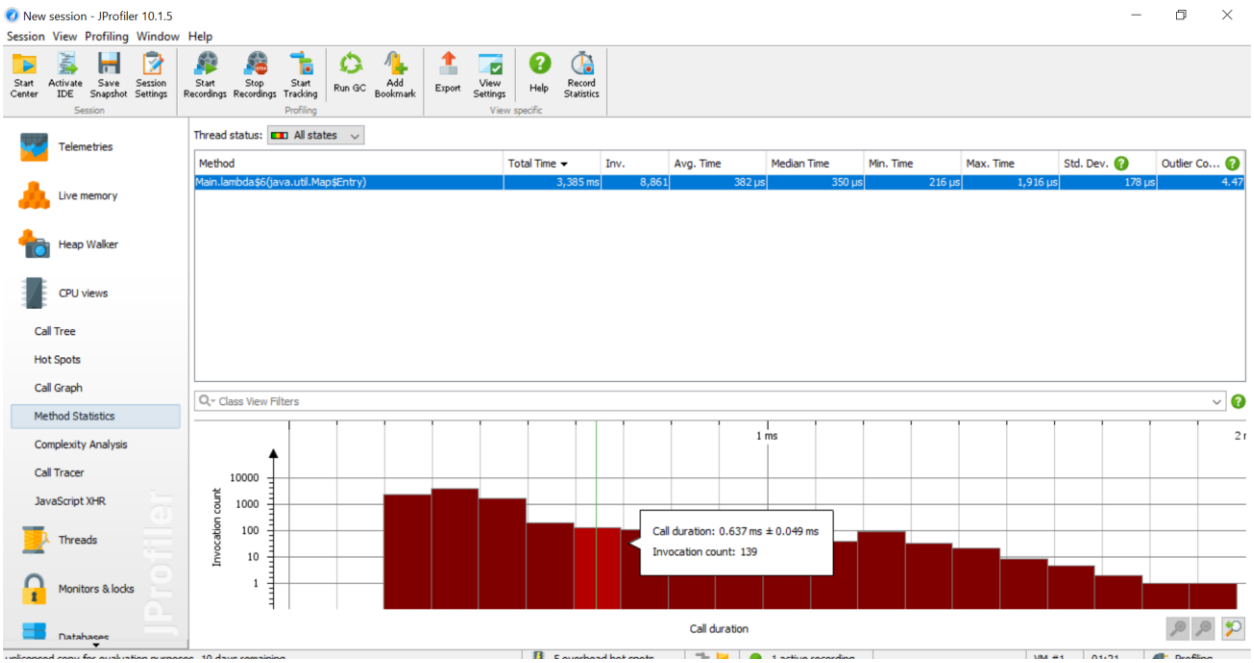


Ordenada

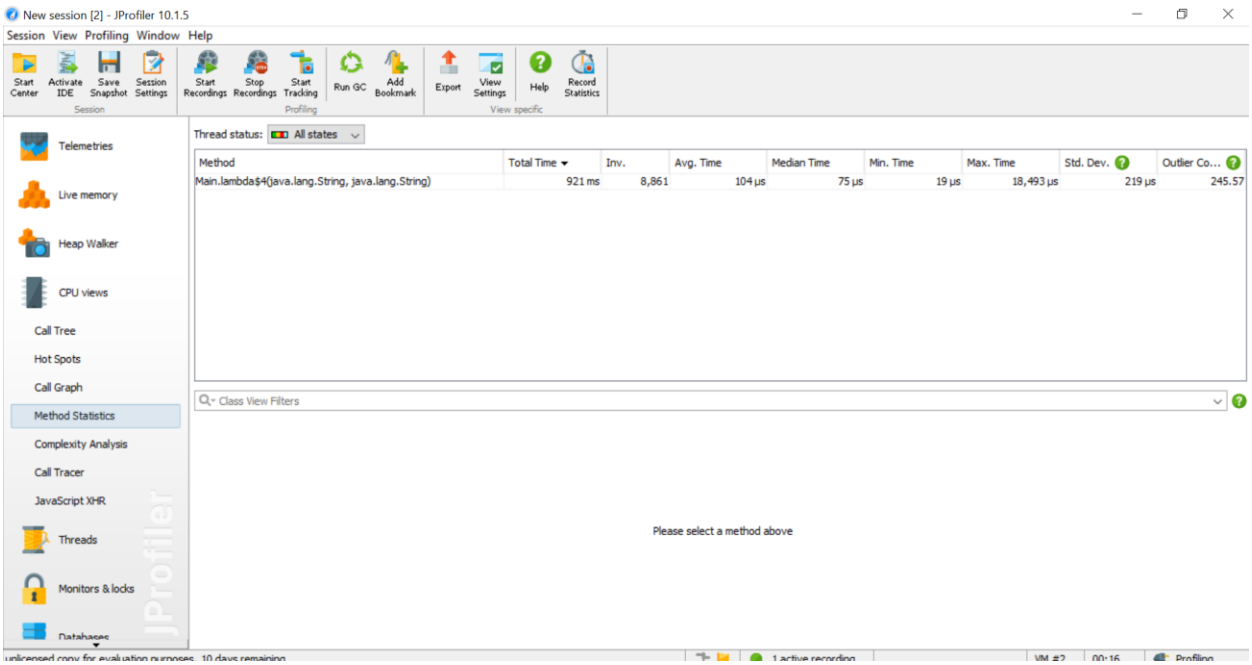


2. TreeMap

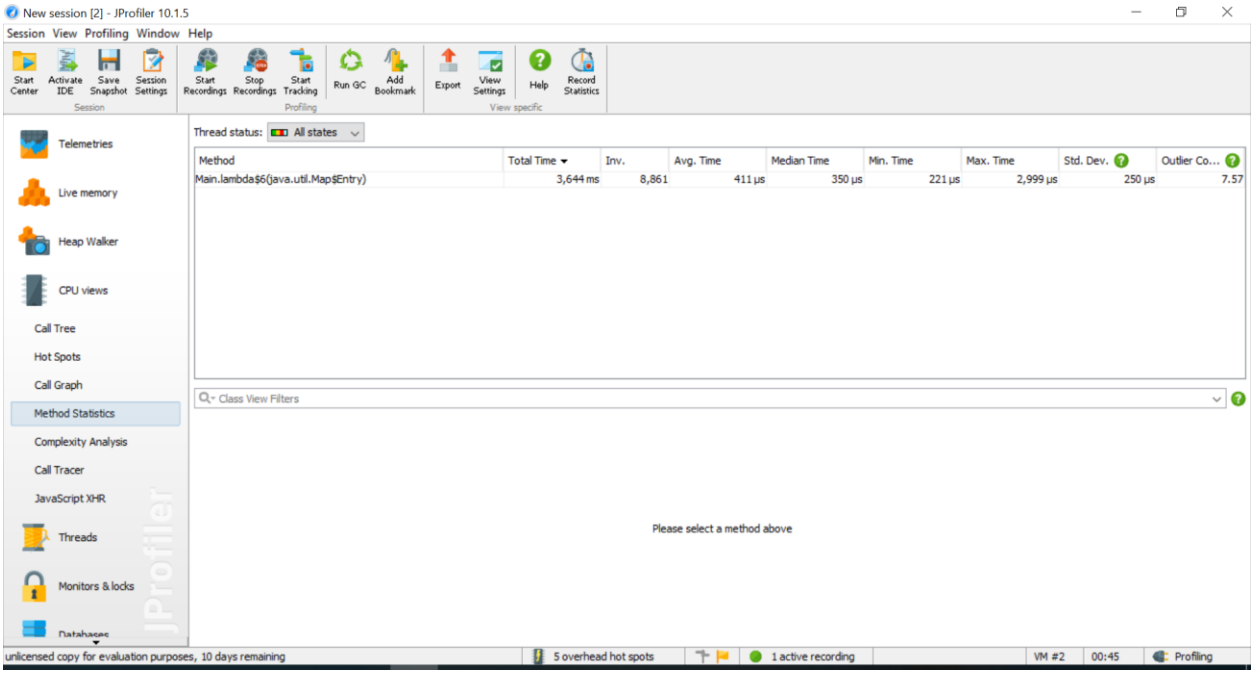

New session [2] - JProfiler 10.1.5
Session View Profiling Window Help
Start Center
Activate IDE
Save Snapshot
Session Settings
Start Recordings
Stop Recordings
Start Tracking
Run GC
Add Bookmark
Export
View Settings
Help
Record Statistics
Session
Profiling
View specific
Telemetries
Live memory
Heap Walker
CPU views
Call Tree
Hot Spots
Call Graph
Method Statistics
Complexity Analysis
Call Tracer
JavaScript XHR
Threads
Monitors & locks
Databases
JProfiler
Thread status: All states
Method
Total Time
Inv.
Avg. Time
Median Time
Min. Time
Max. Time
Std. Dev.
Outlier Co...
Main.lambda$4(java.lang.String, java.lang.String)
863 ms
8,861
97 µs
75 µs
8 µs
8,825 µs
122 µs
116.67
Class View Filters
1 ms
2 ms
3 ms
4 ms
5 ms
6 ms
7 ms
8 ms
Invocation count
10000
1000
100
10
1
Call duration


New session - JProfiler 10.1.5
Session View Profiling Window Help
Start Center
Activate IDE
Save Snapshot
Session Settings
Start Recordings
Stop Recordings
Start Tracking
Run GC
Add Bookmark
Export
View Settings
Help
Record Statistics
Session
Profiling
View specific
Telemetries
Live memory
Heap Walker
CPU views
Call Tree
Hot Spots
Call Graph
Method Statistics
Complexity Analysis
Call Tracer
JavaScript XHR
Threads
Monitors & locks
Databases
JProfiler
Thread status: All states
Method
Total Time
Inv.
Avg. Time
Median Time
Min. Time
Max. Time
Std. Dev.
Outlier Co...
Main.lambda$6(java.util.Map$Entry)
3,385 ms
8,861
382 µs
350 µs
216 µs
1,916 µs
178 µs
4.47
Class View Filters
1 ms
Invocation count
10000
1000
100
10
1
Call duration: 0.637 ms ± 0.049 ms
Invocation count: 139
Call duration

3.LinkedHashMap



Ordenada



Vemos que todas las implementaciones tienen tiempos muy parecidos, pero la más rápida fue TreeMap puesto que su tiempo fue un poco menor que las de HashMap y LinkedHashMap por algunos milisegundos.