

**Profiler utilizado: JProfiler (IntelliJ)****HashMap** : Mas lento

- ▼ m  100.0% – 30,802 ms – 1 inv. Main.main
  - ▶ m 0.1% – 19,064  $\mu$ s – 1 inv. MostrarElementosProfiler.mostrarElementos
  - ▶ m 0.0% – 12,002  $\mu$ s – 1 inv. java.lang.ClassLoader.loadClass
  - ▶ m 0.0% – 600  $\mu$ s – 0 inv. MostrarElementosProfiler.<init>

**TreeMap**: Mas rápida

- ▼ m  100.0% – 6,605 ms – 1 inv. Main.main
  - ▶ m 0.3% – 19,695  $\mu$ s – 1 inv. MostrarElementosProfiler.mostrarElementos
  - ▶ m 0.2% – 15,220  $\mu$ s – 1 inv. java.lang.ClassLoader.loadClass
  - ▶ m 0.0% – 453  $\mu$ s – 0 inv. MostrarElementosProfiler.<init>

**LinkedMap**: Intermedio

- ▼ m  100.0% – 7,466 ms – 1 inv. Main.main
  - ▶ m 0.3% – 20,275  $\mu$ s – 1 inv. MostrarElementosProfiler.mostrarElementos
  - ▶ m 0.2% – 12,731  $\mu$ s – 1 inv. java.lang.ClassLoader.loadClass
  - ▶ m 0.0% – 424  $\mu$ s – 0 inv. MostrarElementosProfiler.<init>

Calcule la complejidad de tiempo para la implementación HashMap, para mostrar todas las cartas. Indique como llegó a ese resultado.

Dos algoritmos pueden tener la misma complejidad, pero uno puede funcionar mejor que el otro. Recuerde que  $f(N)$  is  $O(N)$  significa que:

$$C1 \cdot N \leq \liminf_{N \rightarrow \infty} f(N) \leq C2 \cdot N$$

Donde  $C_1$  y  $C_2$  son constantes estrictamente positivas. La complejidad no dice nada sobre cuán pequeños o grandes son los valores de  $C$ . Para dos algoritmos diferentes, las constantes probablemente serán *diferentes*.

HashMap funciona según el principio del hash y utiliza internamente el código hash como base para almacenar el par clave-valor. Con la ayuda de hashcode, HashMap distribuye los objetos a través de los cubos de tal manera que hashmap coloca los objetos y los recupera en tiempo constante  $O(1)$ .

El mejor y promedio caso de HashMap para búsqueda, inserción y eliminación es  $O(1)$  y el peor de los casos es  $O(n)$ .