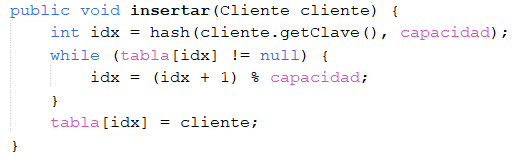
Notación asintotica Big-O

1. **HashLineal**

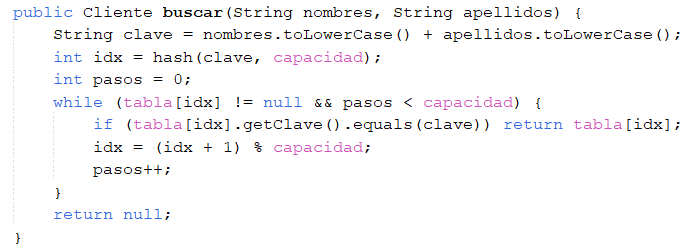
* **insertar(Cliente cliente)**

****

Mejor caso: O(1)

Peor caso: O(n), donde n es el tamaño de la tabla (capacidad)

* **buscar(String nombres, String apellidos)**

****

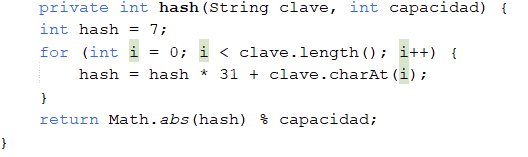
En el mejor caso, el cliente está en la primera posición → una comparación.

En el peor caso, recorre hasta n posiciones sin encontrarlo.

Mejor caso: O(1)

Peor caso: O(n)

* **hash(String clave, int capacidad)**

****

Si m es la longitud de la clave → O(m)

Conclusión

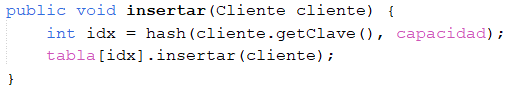
| Método | Complejidad Mejor Caso | Complejidad Peor Caso |
| --- | --- | --- |
| insertar | O(1) | O(n) |
| buscar | O(1) | O(n) |
| hash | O(m) | O(m) |

Donde:  
n es el tamaño de la tabla hash (capacidad).

m es la longitud de la clave del cliente (nombres + apellidos).

1. **HashEncadenamiento**

* **insertar(Cliente cliente)**

****

**Complejidad:**

El hash toma tiempo O(m) donde m es la longitud de la clave (como en el caso anterior).

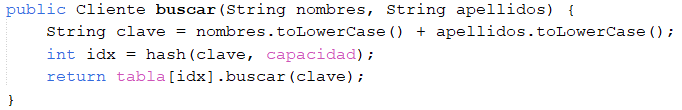
El insertar de un árbol binario tiene:

Mejor caso: O(log k) si el árbol está balanceado.

Peor caso: O(k) si el árbol está desbalanceado (forma de lista).

Donde k es la cantidad de elementos en ese árbol (número de colisiones en esa celda).

* **buscar(String nombres, String apellidos)**

****

**Complejidad:**

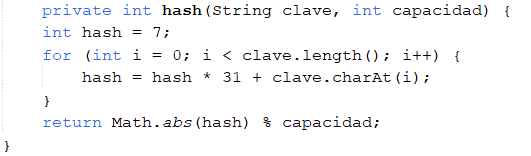
El hash: O(m)

El buscar en el ABB:

Mejor caso: O(log k)

Peor caso: O(k)

* **hash(String clave, int capacidad)**

****

**Complejidad:**

Recorre todos los caracteres de la clave → O(m)

Conclusión

| Método | Mejor Caso | Peor Caso |
| --- | --- | --- |
| insertar | O(m + log k) | O(m + k) |
| buscar | O(m + log k) | O(m + k) |
| hash | O(m) | O(m) |

### **Donde:**

* m: longitud de la clave.
* k: número de elementos almacenados en el árbol de una celda específica del hash.
* n: total de elementos insertados.
* capacidad: número de celdas del hash → afecta cuántos elementos terminan en cada árbol.