ARRAYS, STRINGS Y HASH TABLES

Daniel Blanco Calviño

STRING == ARRAY

- Los ejercicios de **arrays y strings son intercambiables.**

- Un array es una **colección de elementos**. Ejemplo: [1, 2, 3, 4].
- Un string es una **colección de caracteres**. Ejemplo: "Hola" = ['H', 'o', 'l', 'a'].

TABLAS HASH

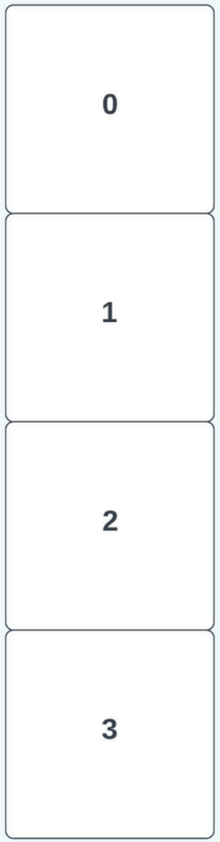
- Estructura de datos que almacena **valores dado su hash.**
  - Hash: función que **codifica un valor para generar una clave**. Suele ser int o long.
  - **Muy eficientes en la búsqueda** del valor dada una clave.

- Para insertar un par:
  - Se calcula el **hash de la clave**. Dos diferentes valores pueden tener el mismo hash.
  - Se mapea el hash a un índice del array. Dos valores podrían ir al mismo índice.
    - index = hash(key) % array_length
  - En cada índice hay una **lista enlazada** de claves y valores.
    - Necesario debido a colisiones.

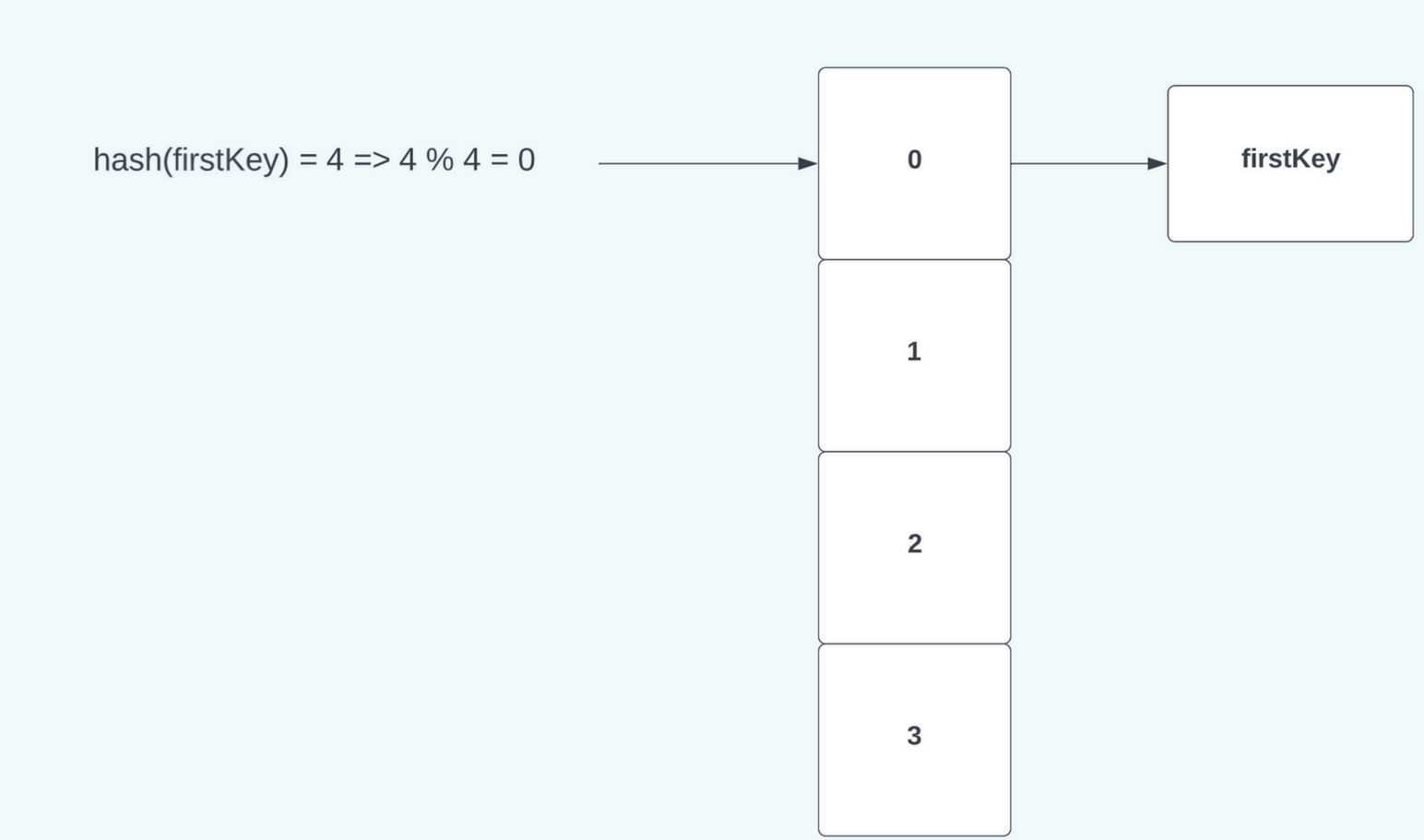
TABLAS HASH

- Para la **búsqueda** dada una clave se realiza el mismo proceso:
  - Se calcula el **hash de la clave**.
  - Se mapea el hash a un índice del array.
  - Recorremos la lista buscando la clave hasta encontrarla o concluir que la clave no se encuentra.

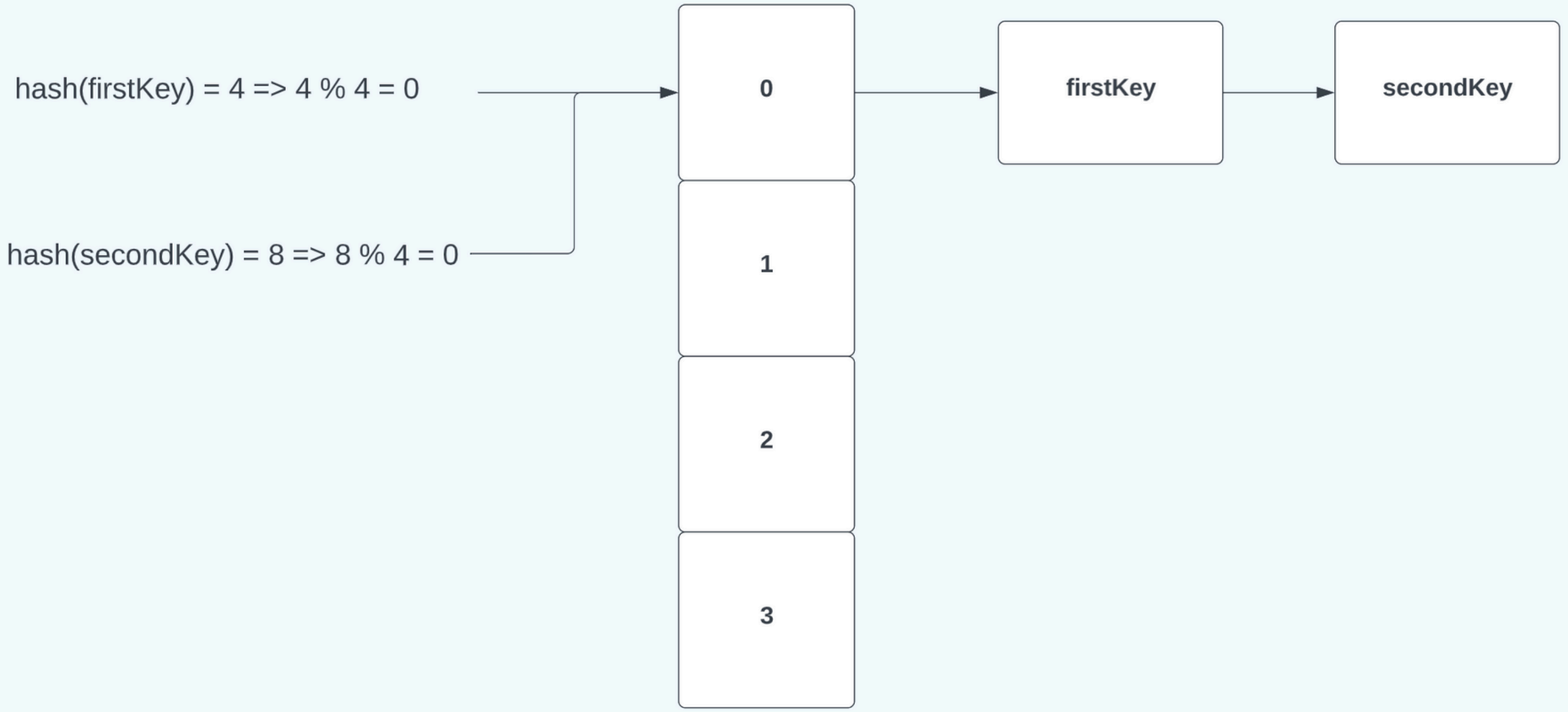
TABLAS HASH - EJEMPLO



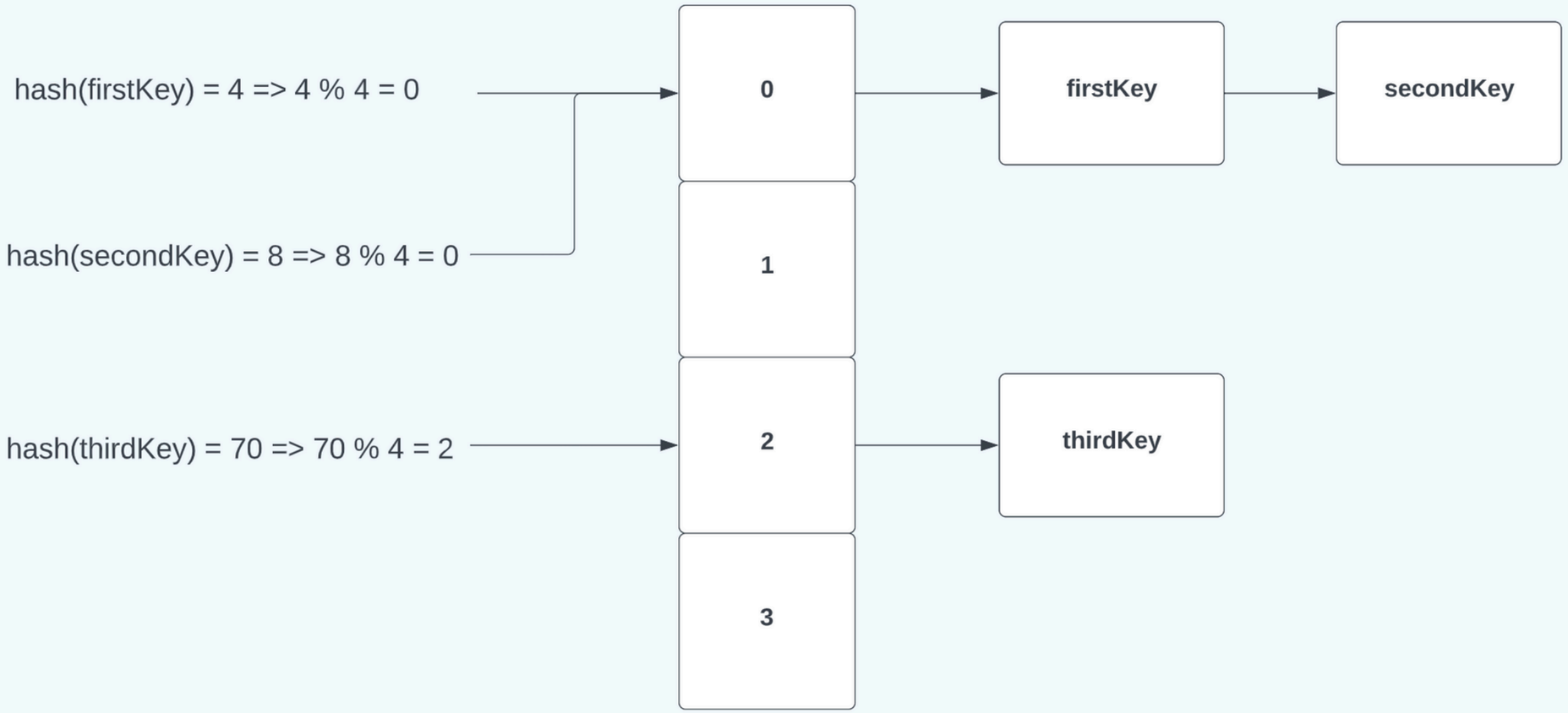
TABLAS HASH - EJEMPLO



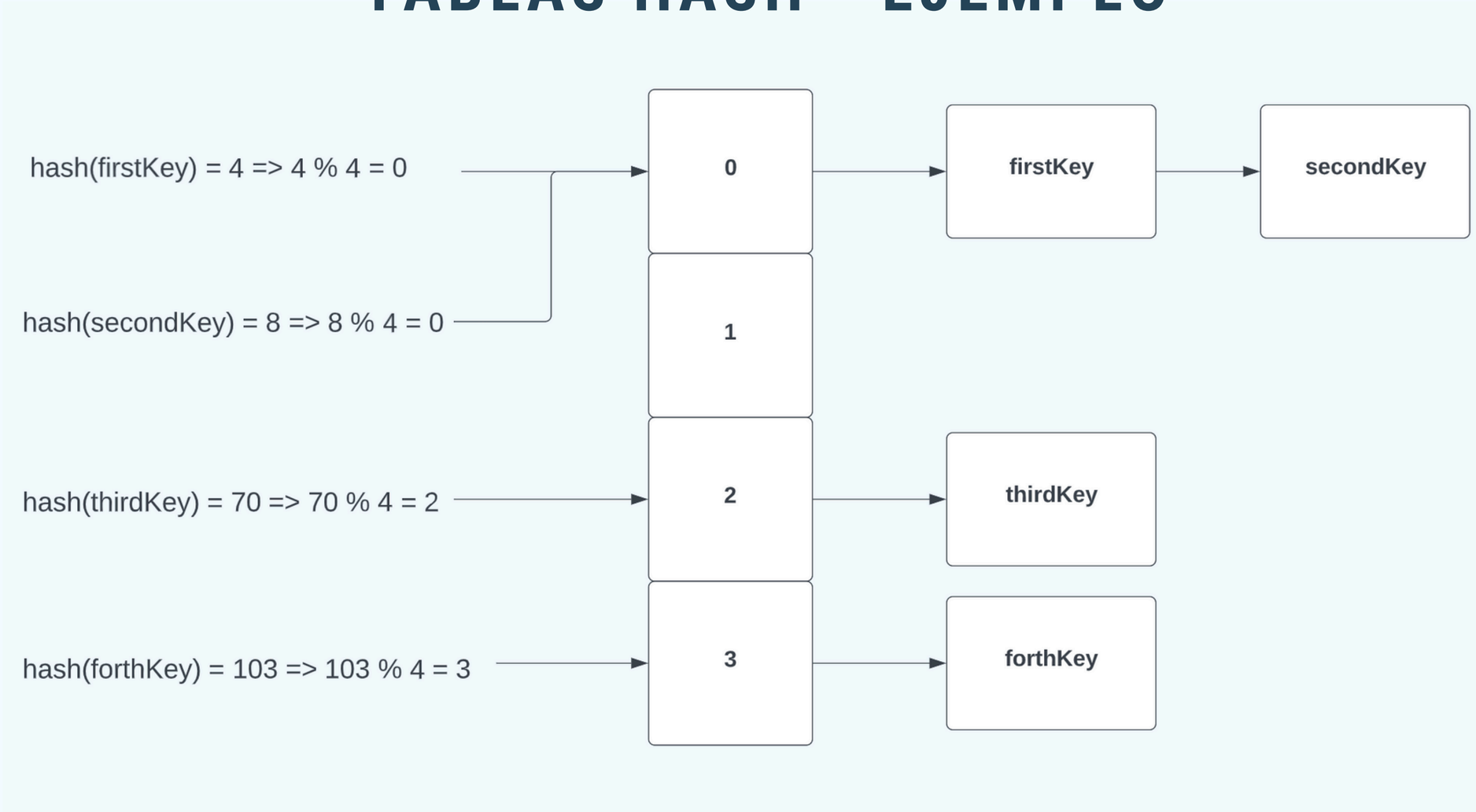
TABLAS HASH - EJEMPLO



TABLAS HASH - EJEMPLO



TABLAS HASH - EJEMPLO



TABLAS HASH - COMPLEJIDAD

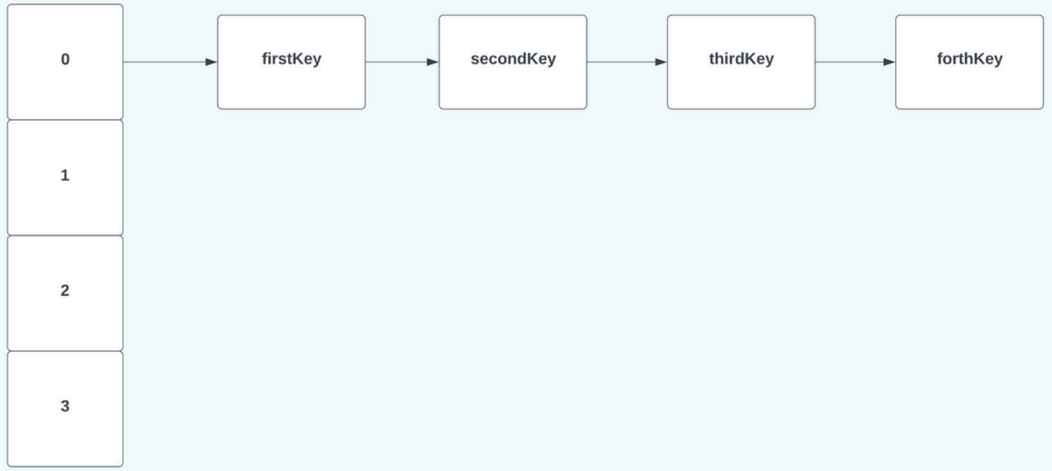
- Inserción
  - **O(1) en el mejor y peor caso**. Muy eficiente en las inserciones.

- Búsqueda:
  - **O(1) en el caso promedio.** Muy eficiente para las búsquedas.
  - **O(N) en el peor de los casos.** Función hash inadecuada.

TABLAS HASH - COMPLEJIDAD

- Inserción
  - **O(1) en el mejor y peor caso**. Muy eficiente en las inserciones.

- Búsqueda:
  - **O(1) en el caso promedio**. Muy eficiente para las búsquedas.
  - **O(N) en el peor de los casos**. Función hash inadecuada.



TABLAS HASH - EJEMPLOS EN LENGUAJES

- Java
  - HashMap
  - Set
- Python
  - Dictionary ({ })
- JavaScript
  - Map

STRING BUILDER

```
String commaSeparatedList(String[] names) {
    String nameList = "";

    for (String name : names) {
      nameList += name + ",";
    }

    return nameList.substring(0, nameList.length() - 1);
}
```

STRING BUILDER

```
String commaSeparatedList(String[] names) {
    String nameList = "";

    for (String name : names) {
      nameList += name + ",";
    }

    return nameList.substring(0, nameList.length( ) - 1);
}
```

1 + 2 + ... + n = n(n+1) / 2 = O(n^2)

En cada iteración **se crea una nueva String** de tamaño nameList.length() + (name + ",").length() y se copian todos los caracteres.

STRING BUILDER

```
String commaSeparatedList(String[] names) {
    String nameList = "";

    for (String name : names) {
      nameList += name + ",";
    }

    return nameList.substring(0, nameList.length() - 1);
}
```

O(names)

1 + 2 + ... + n = n(n+1) / 2 = O(n^2)

En cada iteración **se crea una nueva String** de tamaño nameList.length() + (name + ",").length() y se copian todos los caracteres.

STRING BUILDER

```
String commaSeparatedList(String[] names) {
    String nameList = "";

    for (String name : names) {
      nameList += name + ",";
    }

    return nameList.substring(0, nameList.length( ) - 1);
}
```

O(names * n^2)

O(names)

1 + 2 + ... + n = n(n+1) / 2 = O(n^2)

En cada iteración **se crea una nueva String** de tamaño nameList.length() + (name + ",").length() y se copian todos los caracteres.

STRING BUILDER

```
String commaSeparatedList(String[] names) {
    StringBuilder nameList = new StringBuilder();

    for (String name : names) {
      nameList.append(name).append(",");  O(1)
    }

    return nameList.toString().substring(0, nameList.length() - 1);
}
```

StringBuilder usa un **array redimensionable => O(names)**