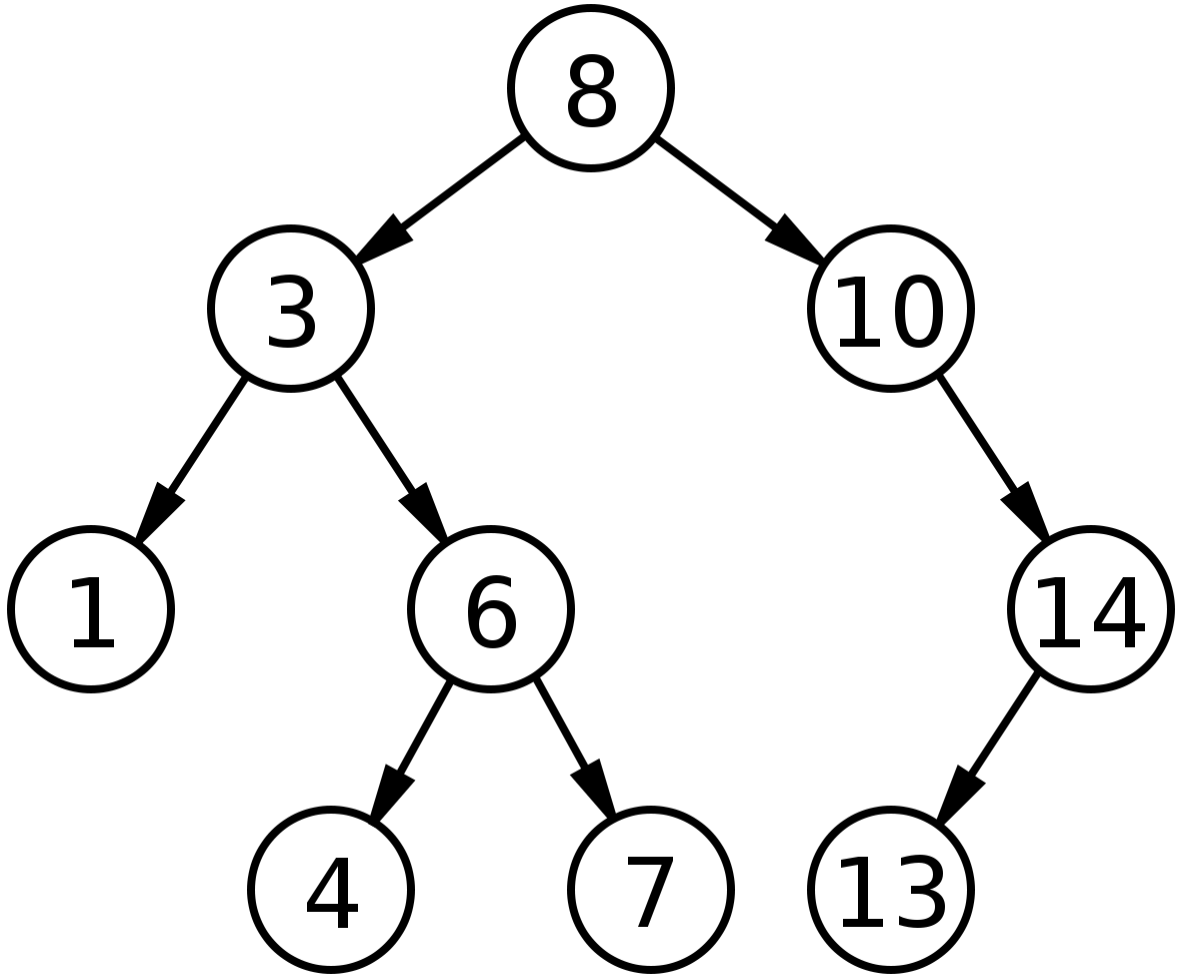
Árboles y hashtables

Estructuras de datos

Binary Search Tree

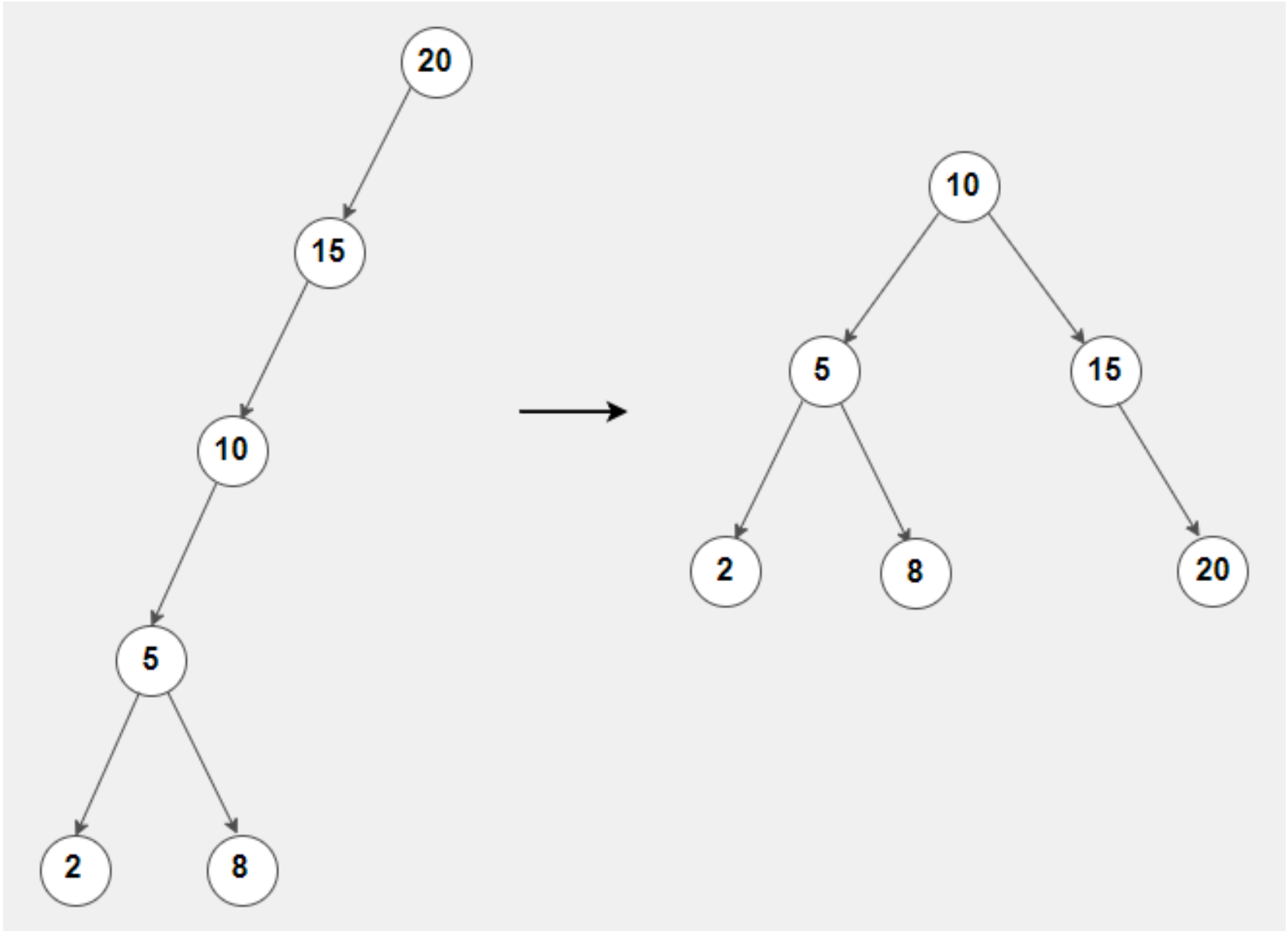
- Una forma eficiente de almacenar elementos de forma ordenada son los árboles.
- En un árbol hay un **nodo raíz** del que cuelgan todos los demás.
- Cada nodo almacena un valor y una referencia al nodo izquierdo y otra al nodo derecho.
- Los nodos que no tienen nodos hijos se denominan nodos hoja.
- Todos los nodos que cuelgan directa o indirectamente a la izquierda de un nodo almacenan valores menores que el del propio nodo.
- Todos los nodos que cuelgan directa o indirectamente a la derecha de un nodo almacenan valores mayores que el del propio nodo.

Binary Search Tree



Binary Search Tree

- La **profundidad de un árbol** es el número máximo de niveles que hay entre el nodo raíz y las hojas.
- La **altura de un nodo** es la distancia entre el mismo y el nodo raíz.
- Para comprobar si un elemento existe en el árbol habrá que realizar, como máximo, tantos pasos como la profundidad del árbol.
- Se dice que un árbol está **desbalanceado** cuando la diferencia de altura entre la hoja más baja y la más alta es mayor que 1.
- Cuando un árbol está **balanceado** podemos calcular su profundidad aplicando la fórmula $\mathbf{log_2(n)}$.
- Sin embargo, cuando está **desbalanceado** la profundidad puede llegar hasta **n**.


20
15
10
5
2
8
10
5
15
2
8
20

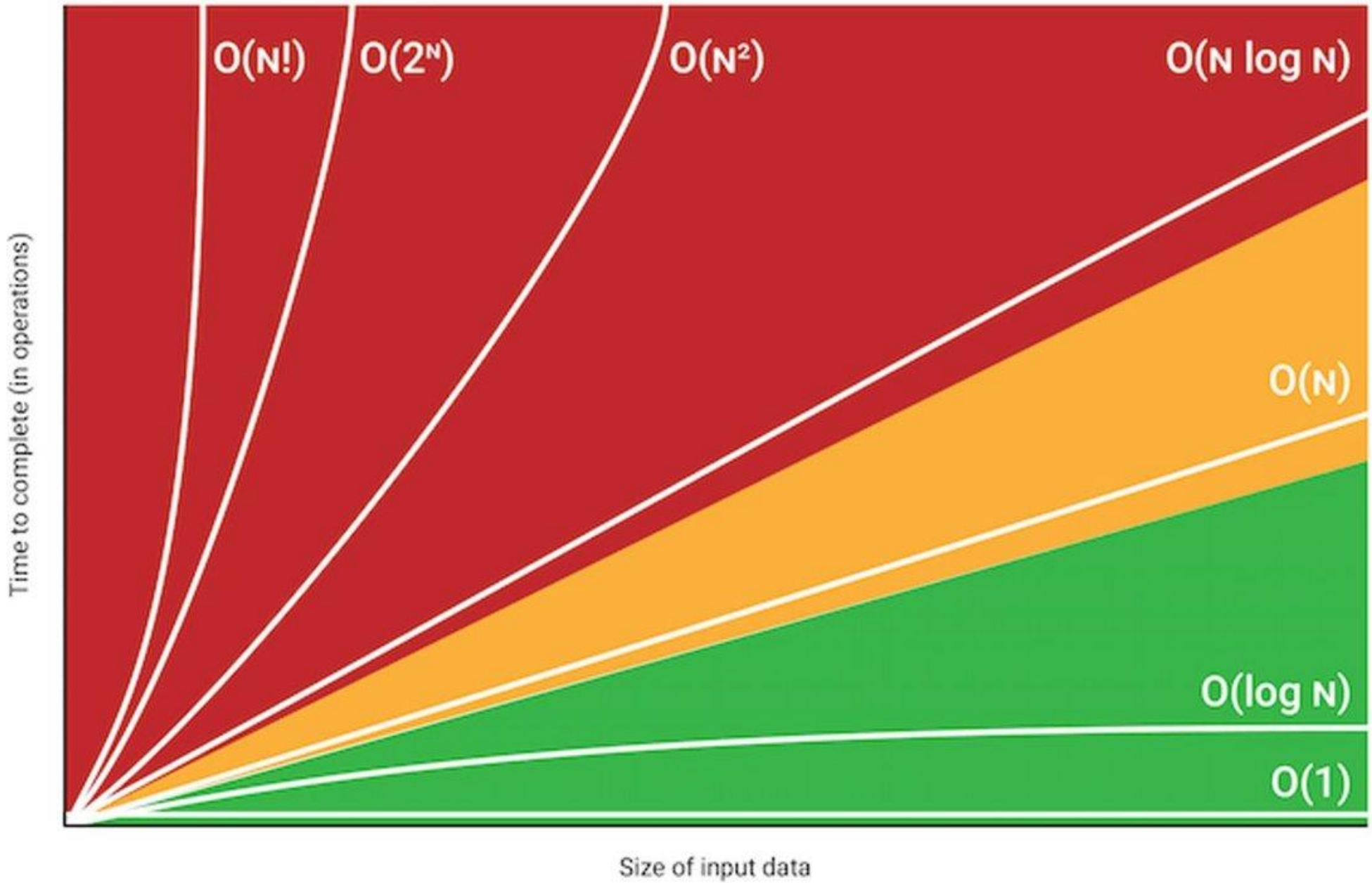
Binary Search Tree

- Si queremos buscar un elemento en un árbol, tendremos que dar como máximo el mismo número de pasos que la profundidad del árbol.
- **Los árboles desbalanceados tienen una mayor profundidad** de la necesaria y, por tanto, reducen la eficiencia de las operaciones sobre los mismos.
- Un árbol balanceado realizará un máximo de $\log_2(n)$ pasos para encontrar un elemento, mientras que uno desbalanceado realizará hasta n pasos.
- Por ejemplo, si en el árbol hay 16 elementos deberemos realizar un máximo de 4 pasos (balanceado) u 16 pasos (desbalanceado)

Self Balanced Trees

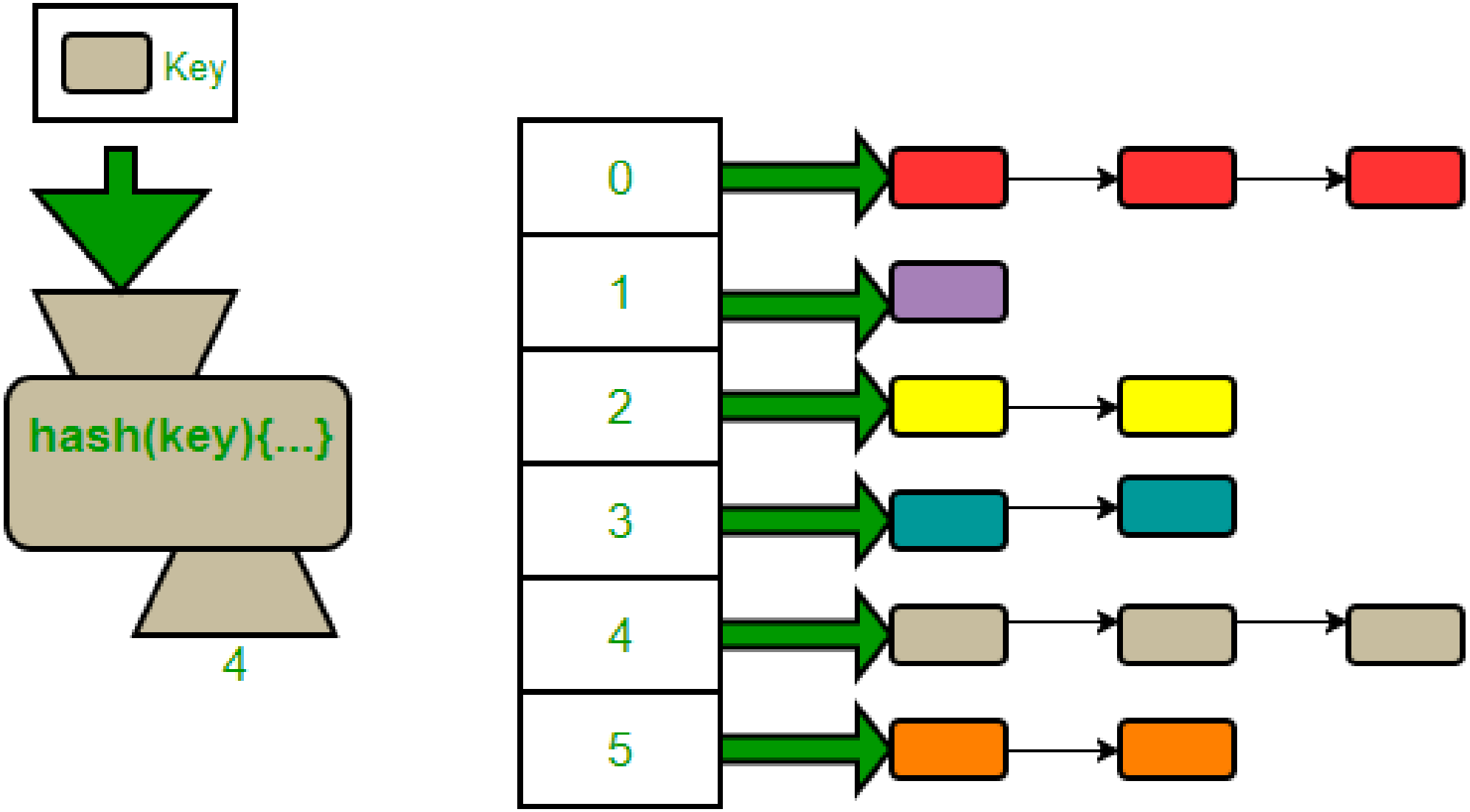
- Un árbol puede acabar desbalanceado dependiendo del orden en el que se insertan / eliminan los elementos.
- Por ejemplo, cuando insertamos los elementos de mayor a menor (o viceversa) el árbol queda completamente desbalanceado.
- Para evitar este problema los árboles suelen incorporar algún algoritmo de rebalanceo como parte de su funcionalidad de inserción y eliminación.
- Los árboles auto balanceados más conocidos son **AVL** (Adelson-Velskii), **Red-Black Tree** y **B-Tree**.

Tree type		**Average**	**Worst**
Binary search tree	Space	Θ(n)	O(n)
	Insert	Θ(log n)	O(n)
	Search	Θ(log n)	O(n)
	Delete	Θ(log n)	O(n)
AVL tree	Space	O(n)	O(n)
	Insert	O(log n)	O(log n)
	Search	O(log n)	O(log n)
	Delete	O(log n)	O(log n)


O(N!)
O(2^N)
O(N^2)
O(N log N)
O(N)
O(log N)
O(1)
Time to complete (in operations)
Size of input data

Hashtables

- Almacena los elementos en una tabla en forma de array.
- Cada elemento se almacena en una "fila" de esta tabla. Cada una de estas "filas" se denominan **buckets**.
- Dentro de cada bucket se almacena una lista de elementos.
- No es posible almacenar dos elementos iguales. Si se insertan dos iguales uno reemplazará al otro.
- Para decidir en qué bucket se almacena cada elemento se le aplica una función de **hashing al valor** (hashCode en Java). Una vez obtenido el hash se establece a qué bucket le corresponde el hash.
- Cada bucket contiene una lista con los elementos asignados al mismo.

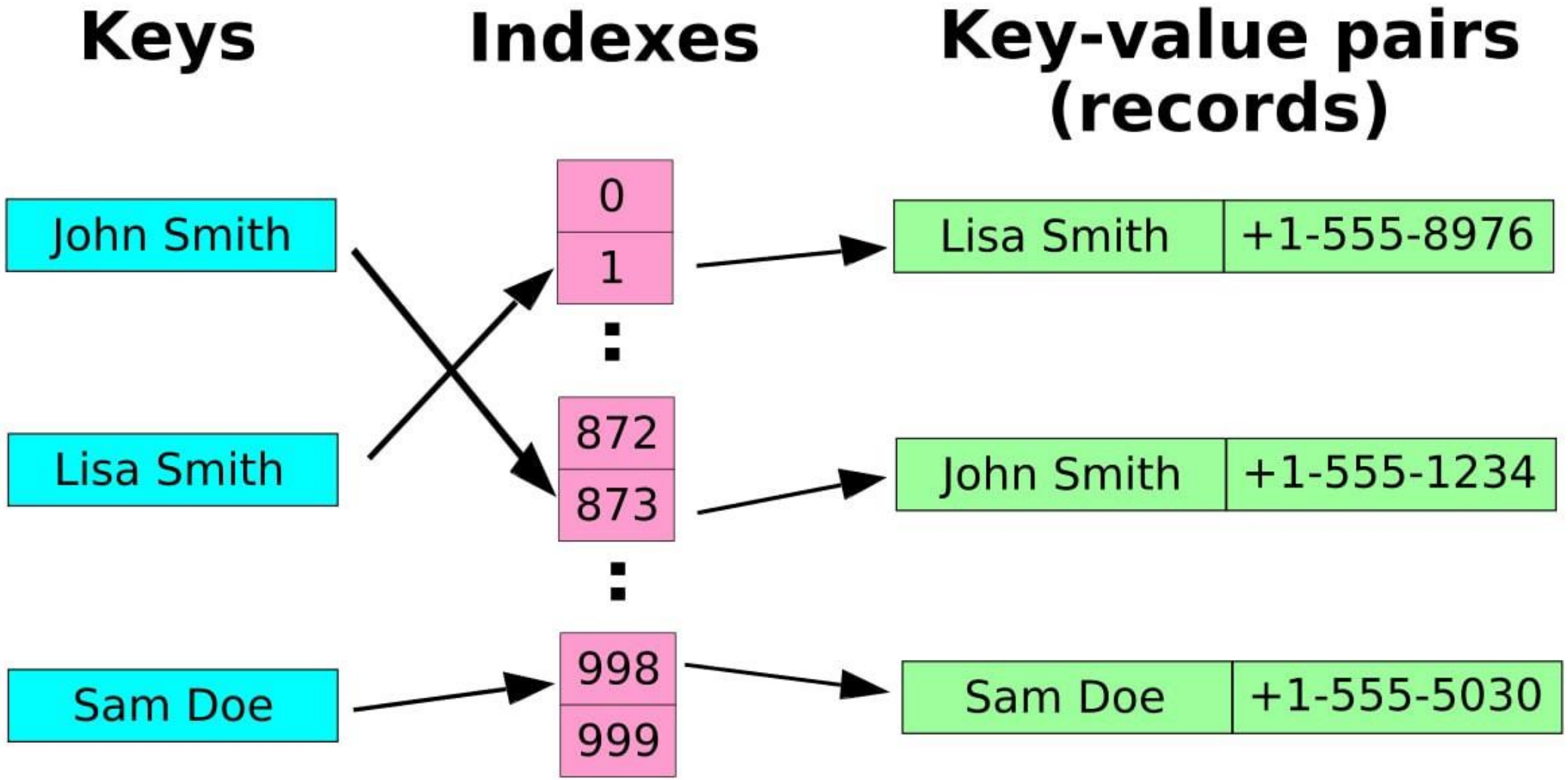

Key
hash(key){...}
4
0
1
2
3
4
5

Hashtables

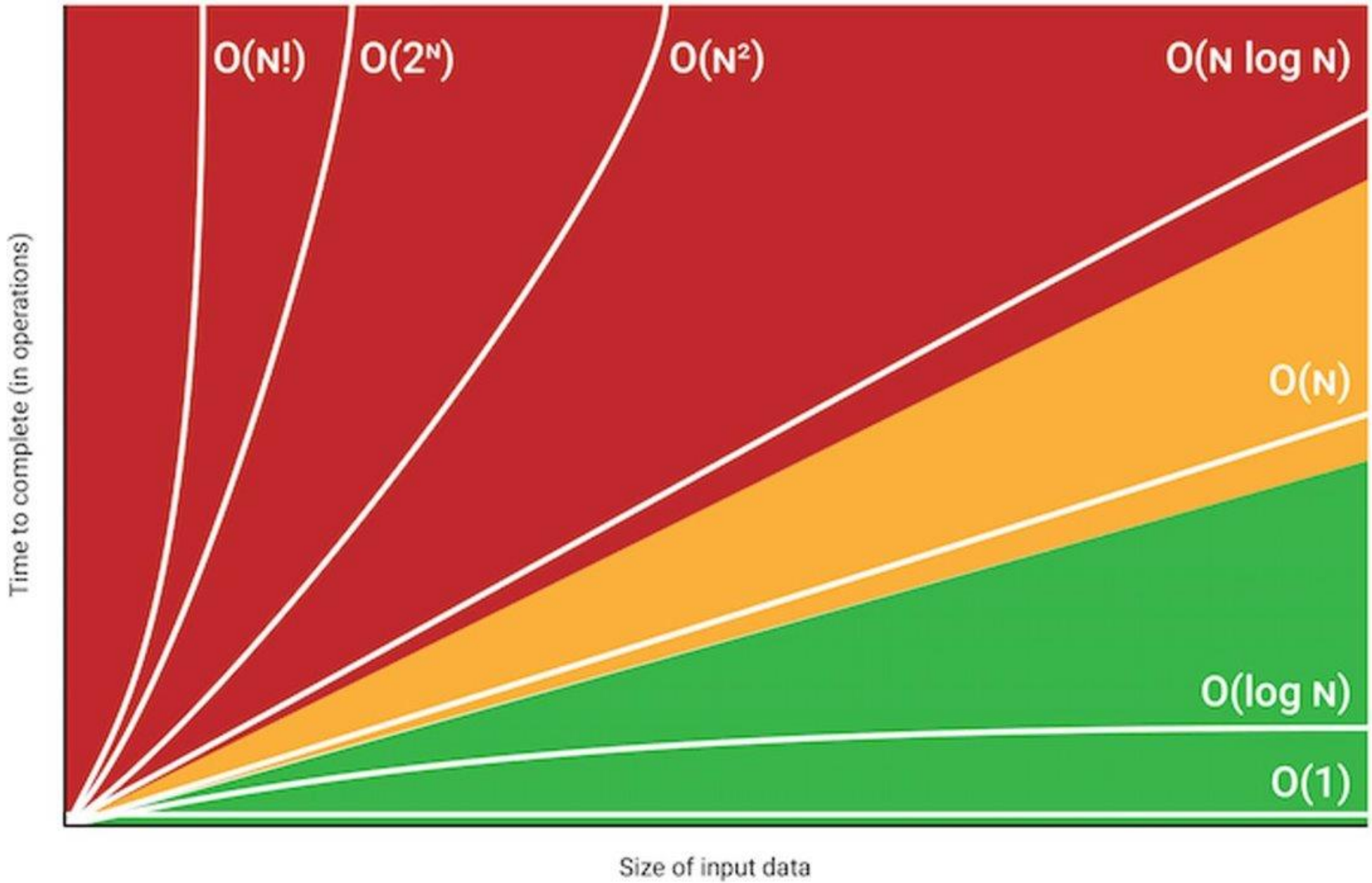
- Las hashtables **no permiten almacenar los elementos en orden**.
- Están **optimizadas** para el **acceso, inserción y eliminación** de datos.

Hashtables asociativas

- Existe una variante muy utilizada de las hashtables que implementa un array asociativo. Esta variante de las hashtables almacena pares clave-valor (records) y son conocidas con nombres como **diccionario** o **hashmap**.
- La **clave** es la que se utiliza para la **selección del bucket**, aplicandose la funcion de hashing únicamente a ella.
- La lista de elementos contiene los **pares clave-valor (records)**.
- No puede haber dos elementos con la misma clave.


Keys
Indexes
Key-value pairs (records)
John Smith
Lisa Smith
Sam Doe
0
1
872
873
998
999
Lisa Smith
+1-555-8976
John Smith
+1-555-1234
Sam Doe
+1-555-5030

Algorithm	Average Case	Worst Case
Space	O(n)	O(n)
Search	O(1)	O(n)
Insert	O(1)	O(n)
Delete	O(1)	O(n)


O(N!)
O(2^N)
O(N^2)
O(N log N)
O(N)
O(log N)
O(1)
Time to complete (in operations)
Size of input data