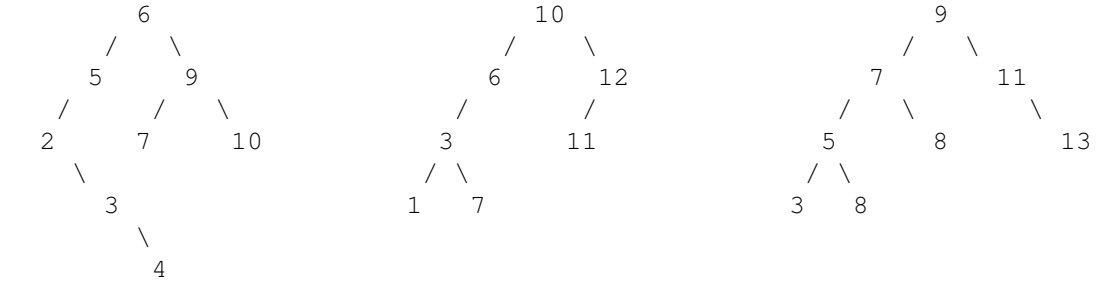
1 Balanced and Valid BSTs

Given the following binary trees, determine if each is a valid BST, and whether or not it has minimum-BST-height. A minimum height BST has the same height as the optimal binary search tree containing the same elements.



Valid:	T	F	Valid:	T	F	Valid:	T	F
Min-Height:	T	F	Min-Height:	T	F	Min-Height:	T	F

Suppose we know the height `H` and number of nodes `N` of a BST. Can we determine whether or not this BST is minimum-BST-height without having to check the values of each node? Why or why not?

2 Ding Dong Design Discussion: A Better Hash Map?

Ding has a wild idea to improve hash maps: Instead of placing items via modulo, use the faster absolute value operation (`bin = Math.abs(key.hashCode())`), and preemptively expand the size of the hash map as necessary (so the bin exists). Ding's twin brother, Dong, believes Ding's idea is boloney, as a hash map always runs in constant time anyways, and thus doesn't need improvement. Enumerate the pros and cons of Ding's modified implementation, and reject or justify the validity of Dong's reaction.

3 Coco's Guerrilla Section Conundrum

Coco doesn't know sign language, but she wants to learn by associating each letter with a number. Help her hash `(Character,Integer)` entry pairs into a `HashMap<Character,Integer>`. Assume that the hash map starts with 10 buckets and resizes with load factor 0.65. Draw the results of inserting four items: `('a',9) ('u',2) ('b',5) ('a',5)`. Determine `C`, the total number of hash collisions, and `F`, the current load factor after the four insertions. The hash code for `'a'` is 97, `'b'` is 98, and so on.

4 Heaps of fun

(a) Assume that we have a binary min-heap (smallest value on top) data structure called `Heap` that stores integers, and has properly implemented `insert` and `removeMin` methods. Draw the heap and its corresponding array representation after each of the operations below:

```
Heap h = new Heap(5); // Creates a min-heap with 5 as the root
h.insert(7);
h.insert(3);
h.insert(1);
h.insert(2);
h.removeMin();
h.removeMin();
```

(b) Your friend Sahil Finn-Garng challenges you to quickly implement an integer max-heap data structure - "Hah! I'll just use my min-heap implementation as a template to write max-heap.java", you think to yourself. Unfortunately, two Destroyer Penguins manage to delete your `MinHeap.java` file. You notice that you still have `MinHeap.class`. Can you still complete the challenge before time runs out? Hint: you can still use methods from MinHeap.

5 Extra for Experts: LRU Cache

LRU stands for Least Recently Used. When removing from the LRU cache, we always remove the least "recently used" item. An item is "recently used" if it was recently added (`add()`) or accessed (`get() contains()`). For example, if we add three items, A, B, and C, and then check `.contains(B)`, calling remove three times would return A, C, and then B. Describe how to implement an LRU cache as efficiently as possible. Then give the tightest Big-O runtimes of `add() get() remove()` and `contains()`. Hint: You are not limited to one classic data structure.