1 B-Tree

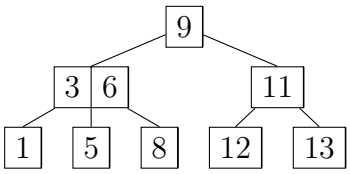
B-Tree Invariants:

- All leaves must be at the same depth from root.
- A non-leaf node with K keys must have exactly $K + 1$ non-null children.

2-3 Tree Invariant:

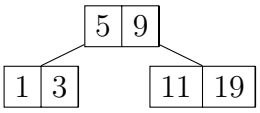
- A node can have at most 2 keys and at most 3 children.

(a) Select all **VALID** 2-3 Tree structures.



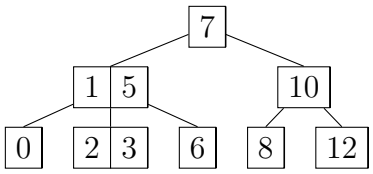
☐

range of values (see 12)

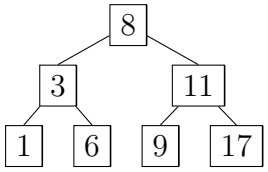


☐

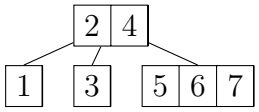
must always have $(K + 1)$ children



■

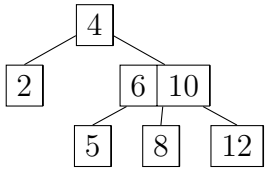


■



☐

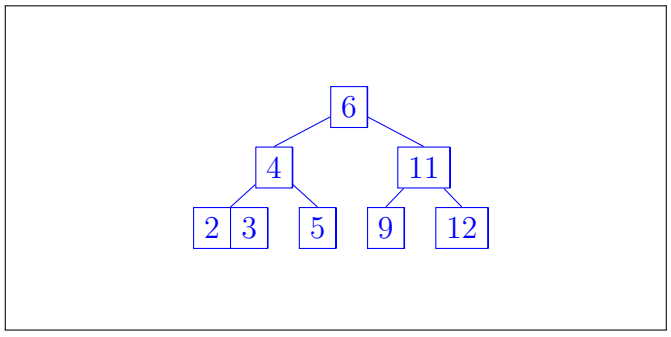
only at most 2 keys per node

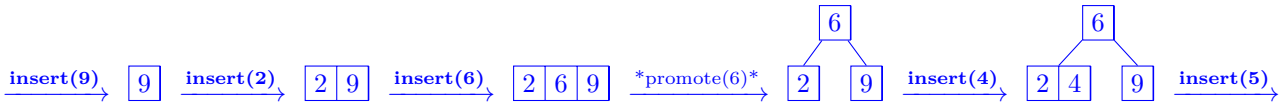


☐

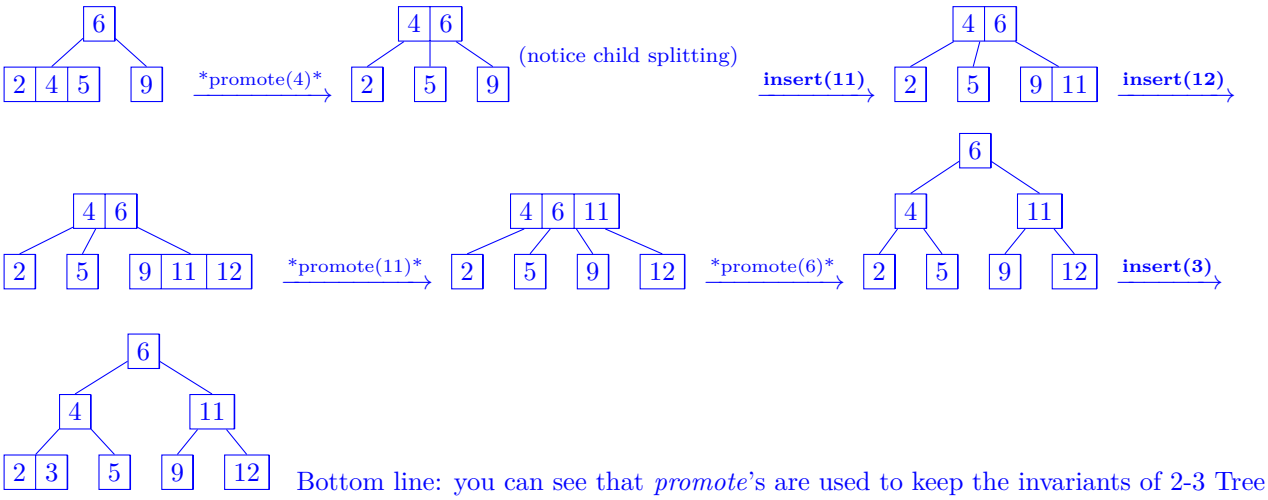
leaves must be at the same depth

(b) Draw the 2-3 tree that results from **inserting** these items in order: **9,2,6,4,5,11,12,3** (hint: try to keep the invariants.)





Will you want to pick up your worksheet later? Circle one: Yes / No
How confident do you feel with the material this week? Circle one: 1 / 2 / 3 / 4

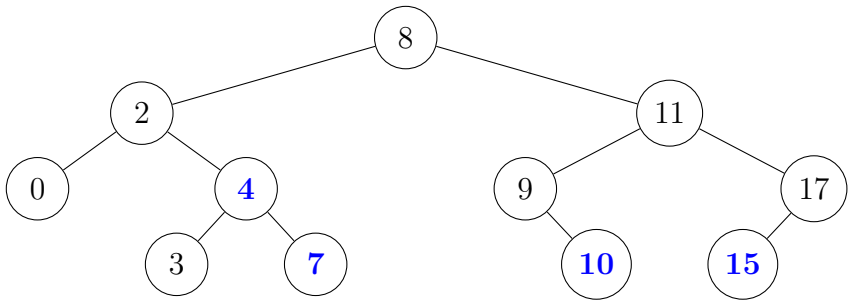


Bottom line: you can see that *promote*'s are used to keep the invariants of 2-3 Tree

2 Binary Search Tree

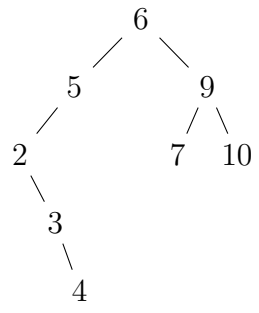
(a) Fill-in the blank nodes of the following binary search tree with valid **integer values**.

One possible answer:



(b) Given the following binary trees, determine if each is a binary search tree, and whether the height of the tree is the same as the height of the optimal binary search tree containing the given elements.

i)

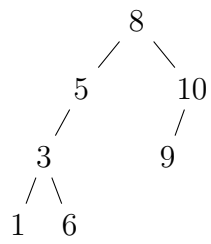


Binary Search Tree: ● Valid ○ Invalid
Height: ○ Balanced ● Unbalanced

Optimal binary search tree with 8 nodes will have height, $h = \lfloor log_2 8 \rfloor = 3$.
The given tree has height of 4.

Will you want to pick up your worksheet later? Circle one: Yes / No
How confident do you feel with the material this week? Circle one: 1 / 2 / 3 / 4

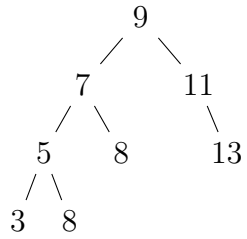
ii)



Binary Search Tree: ◯ Valid ● Invalid (right child of 3 must be in between 3 and 5.)
Height: ◯ Balanced ● Unbalanced

Optimal binary search tree with 7 nodes will have height, $h = \lfloor log_2 7 \rfloor = 2$.
The given tree has height of 3.

iii)



Binary Search Tree: ◯ Valid ● Invalid (right child of 5 must be in between 5 and 7; no duplicates allowed by the definition of binary search tree.)
Height: ● Balanced ◯ Unbalanced

(c) (Optional) Given any insertion order of distinct values $x_1, x_2, ..., x_n$ to insert into BST, we want to build a **balanced BST**. Assume that N is the size of the insertion list.

- `findMedian(list)` runs in $f(N)$ time,
- `insertIntoTree(x)` runs in $g(N)$ time and
- `partition(list, x)` which partitions `list` into two lists, one containing values less than `x` and another containing values more than `x`, runs in $h(N)$ time

Give a recurrence relation modeling the worst-case runtime to build this balanced BST.

$T(N) =$ $2T(\frac{N}{2}) + f(N) + g(N) + h(N)$ for $N > 1$

BST structure depends on the order of insertion. Hence, we want to insert the median value first and split data into two halves. Then, we will recursively insert those two halves.

At each recursive call, we will *find* a median, call `m`, from the list, *insert* `m` into the tree, and *partition* the list, so that the first list contains values less than `m` and the second list contains values more than `m`. Therefore, the non-recursive work is basically $f(N) + g(N) + h(N)$. If the problem size (the size of the list) is N, we know that partitioning the list by its median will cut down the problem size by half. That's why we recursively call two $T(\frac{N}{2})$.

Will you want to pick up your worksheet later? Circle one: Yes / No
How confident do you feel with the material this week? Circle one: 1 / 2 / 3 / 4