Module 3 Graded Quiz

Due Sep 24, 2021 at 11:59pm **Points** 13
Questions 13
Available Sep 11, 2021 at 12am - Feb 11 at 11:59pm 5 months
Time Limit 120 Minutes

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	13 minutes	12.33 out of 13

Score for this quiz: **12.33** out of 13
Submitted Sep 24, 2021 at 8:41pm
This attempt took 13 minutes.

Question 1

1 / 1 pts

Suppose we have a splay tree of *n* nodes that is as balanced as possible (i.e. the number of nodes at each level i is 2^i) and rooted at node *x*. Which of the following most closely approximates the rank of *x*'s left child?

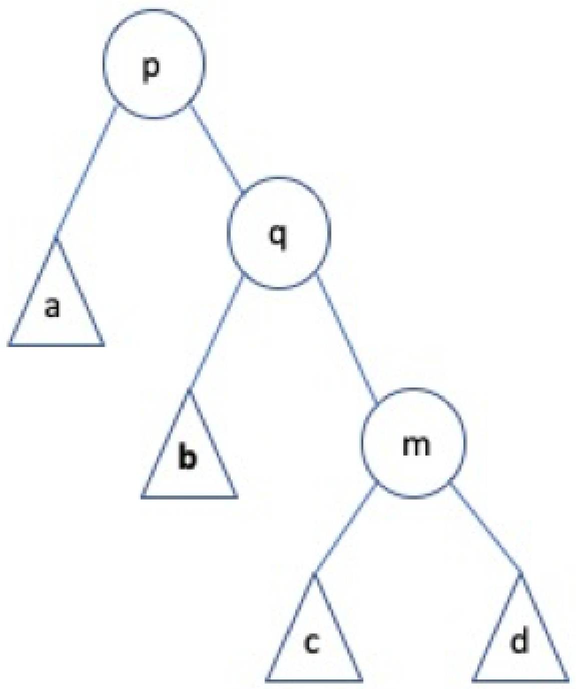
- log(n)
- 1
- n/2
- [✓ log(n-1) - 1]

Correct!

Question 2

1 / 1 pts

Given the following partial splay tree, which of the following represents the smallest upper-bound for the amortized cost of a rollercoaster left-rotation if we assume that the sum of the costs of pointer reassignments for a single rotation is 3? Note: Use only the proof from lecture when formulating your response.



Correct!

[✓ 6+r'(q)+r'(p)-r(m)-r(q)]

3+r'(q)+r'(p)-r(m)-r(q)

3+r'(q)+r'(m)-r(p)-r(q)

6+r'(q)+r'(m)-r(p)-r(q)

Question 3

1 / 1 pts

Suppose we have a very unbalanced splay tree *T* and we perform a search operation on node *x*. If *x* is very deep in the tree, obviously the search operation will be very expensive. Suppose after SPLAY-SEARCH(*T* ,x) is executed, an arbitrary number of other operations is performed on other nodes in the tree. Can we guarantee that the true cost of the next call to SPLAY-SEARCH(*T* ,x) will be relatively inexpensive compared to the first? Explain.

Correct!

[✓ No. Every time an operation is performed on a different node, that node becomes splayed, which pushes x deeper in the tree.]

No. They will have the same costs since the asymptotic time complexity of each splay-tree operation is always O(log(n)) amortized time.

Yes. Since splay trees always become balanced over time, the true cost of the next call to search(x) is at most O(log(n)).

Yes. Node x will be much closer to the root node during the second call to search(x), which makes the operation much less expensive.

Question 4

1 / 1 pts

Suppose you are tasked with performing the aggregate method to determine an amortized bound for an operation *O* whose cost function is defined below:

Let *k* be the iteration index for the number of times *O* has been called.

$$Cost\left(O_k\right) = \{k, \text{ if } k \text{ is a power of } 3, \ 1 \text{ otherwise}$$

What is the total cost *T(n)* of performing 27 iterations?

Correct!

[✓ 63]

27

- ○ 3
- ○ 40

Question 5

1 / 1 pts

From the following list, identify which of the following potential functions are valid.

i.) The number of elements in a linear data structure

ii.) The summation over the depth of each node in a binary heap

iii.) The number of nodes minus the square of the tree-height of a binary search tree

- ○ i., ii., and iii.
- ○ ii. only
- Correct! [✓ i. and ii. only]
- ○ i. only

Question 6

1 / 1 pts

Consider a ternary counter variation of the binary counter, which represents numbers with bits 0,1 and 2. Recall that while using the aggregate method for amortizing the cost of a binary counter increment, we were required to calculate the total cost of *n* increment operations using a geometric series. Repeat this process for the ternary counter. What is the amortized cost of a single increment on the ternary counter?

Correct!

[✓ 3/2 = O(1)]

2 = O(1)

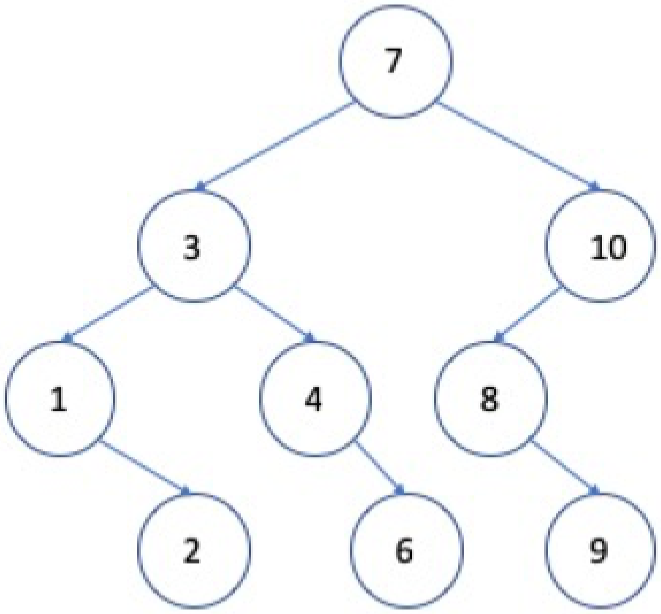
2/3 = O(1)

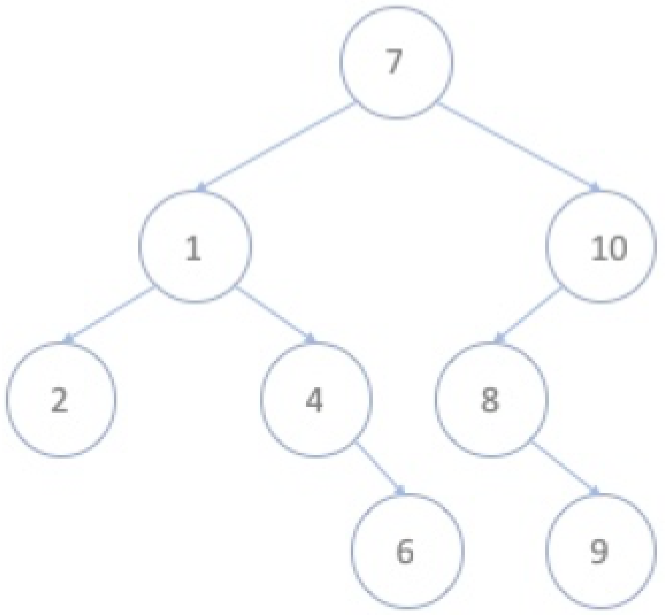
3 = O(1)

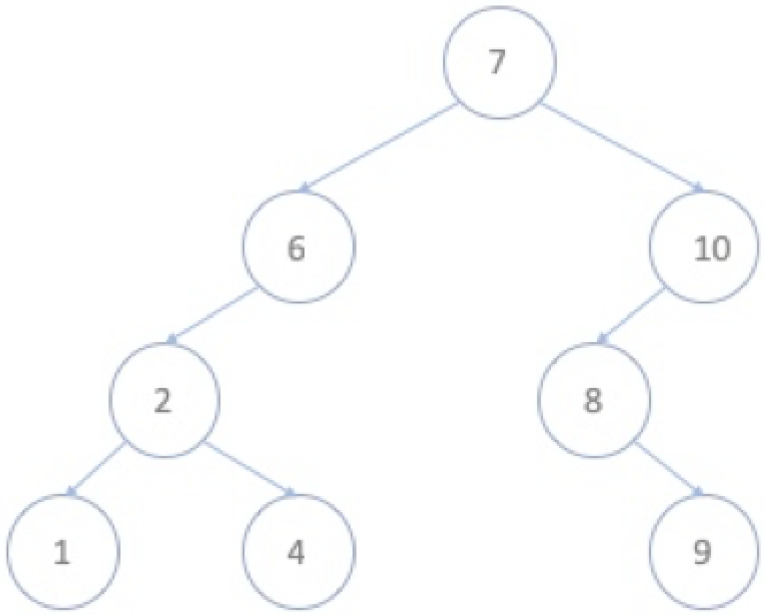
Question 7

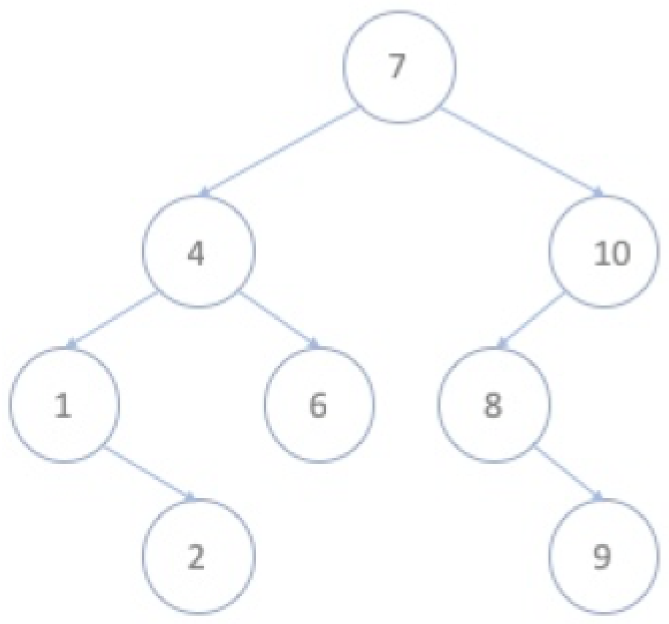
1 / 1 pts

Given the following binary search tree *T*, identify the configuration of *T* after the operation TREE-DELETE(*T*,3). Assume the delete operation uses *predecessors* to replace deleted nodes. Identify the correct resultant configuration.

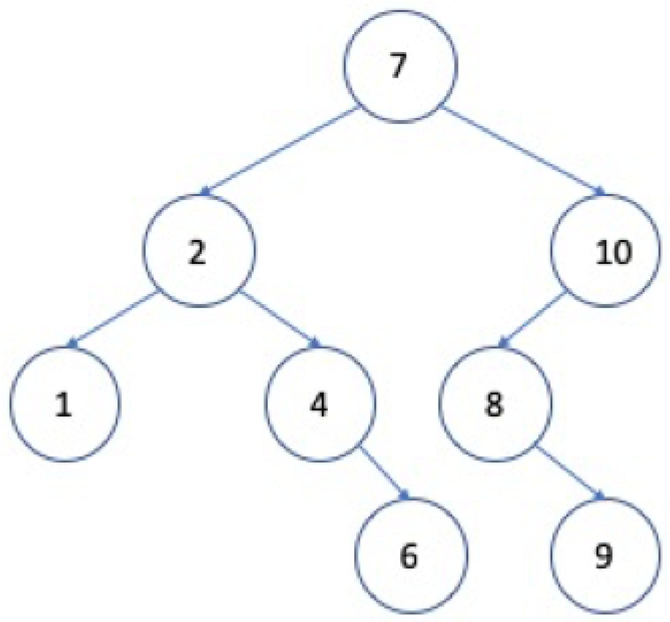



7
1
10
2
4
8
6
9


7
6
10
2
8
1
4
9

○ 

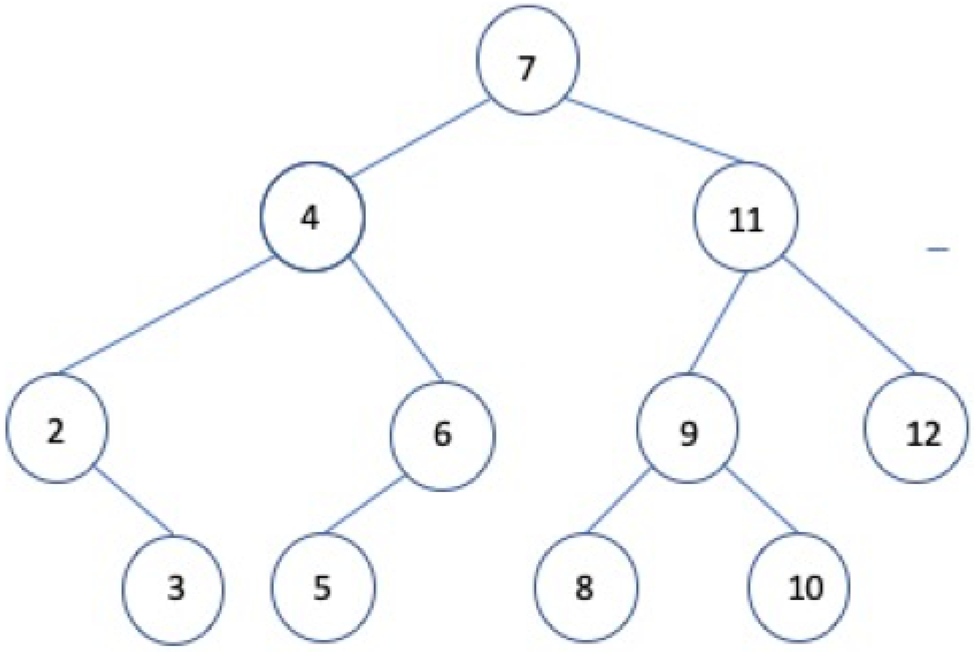
Correct!

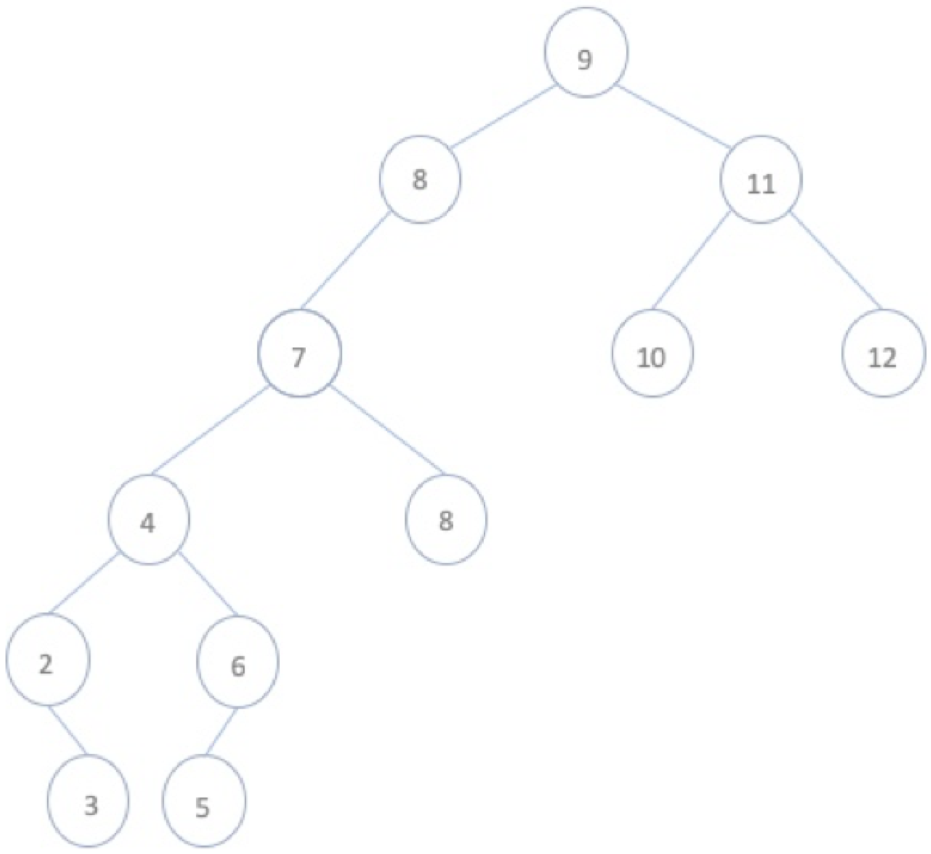
● 

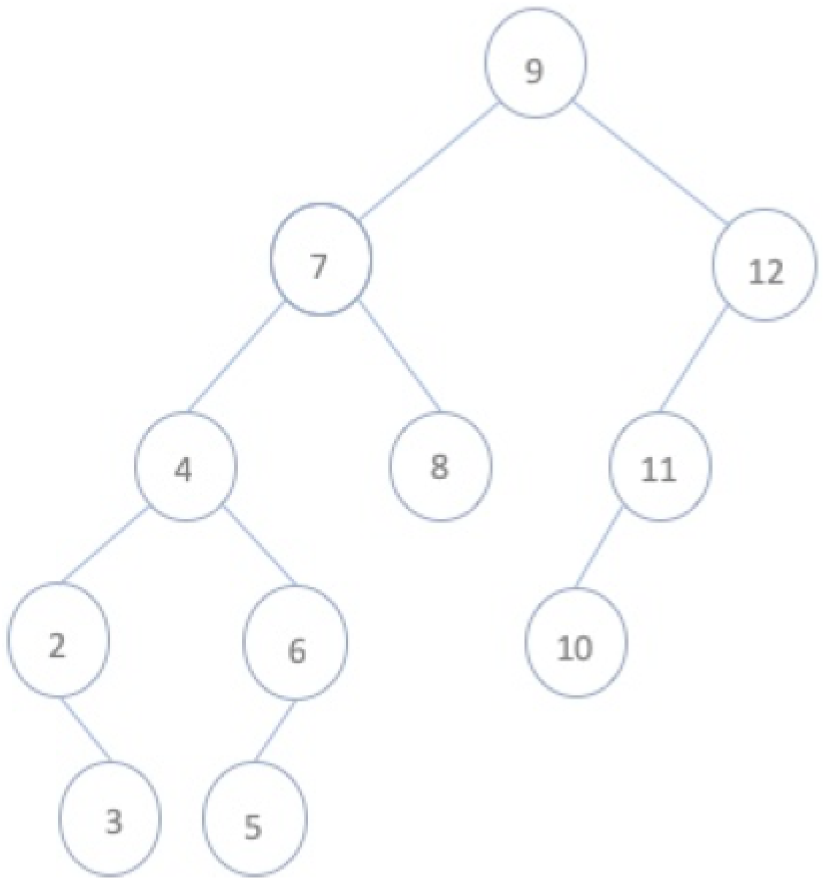
Question 8

1 / 1 pts

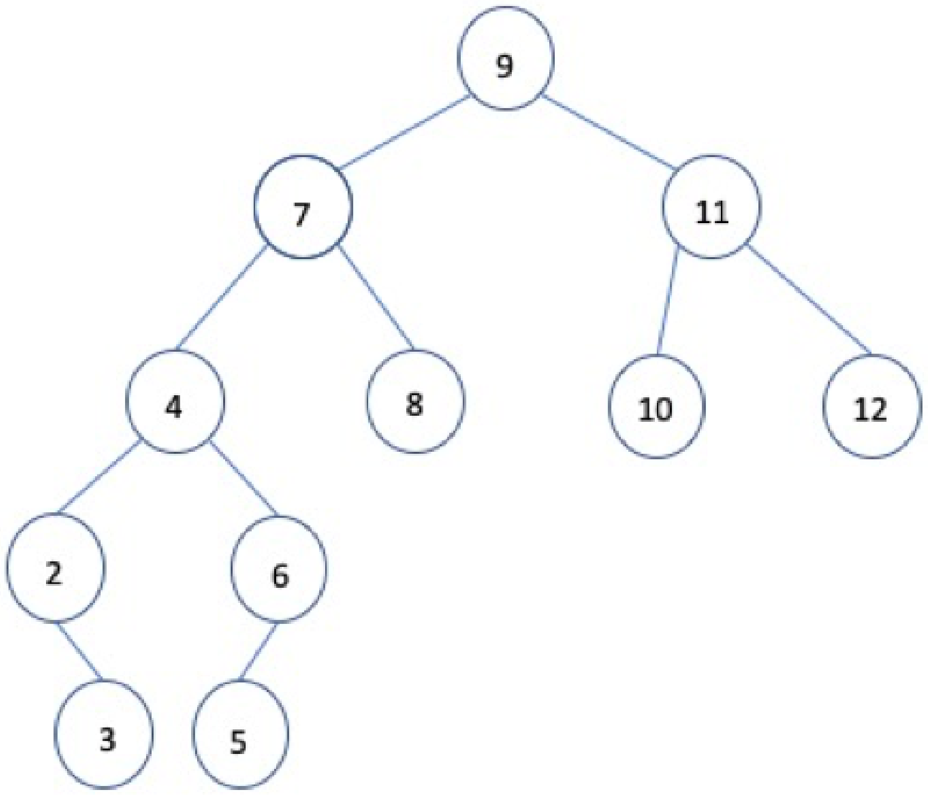
Given the following splay-tree *T*, perform the operation SEARCH(*T*,9). Identify the correct resultant configuration.

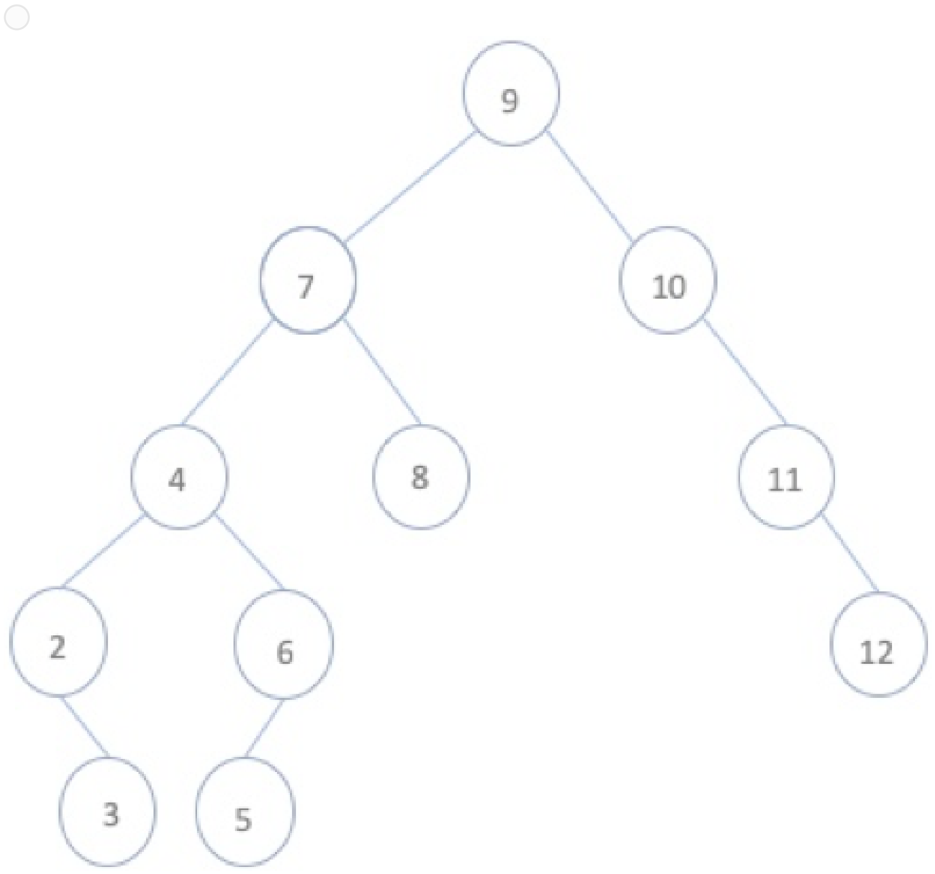

7
4
11
2
6
9
12
3
5
8
10


9
8
11
7
10
12
4
8
2
6
3
5

○ 

Correct!

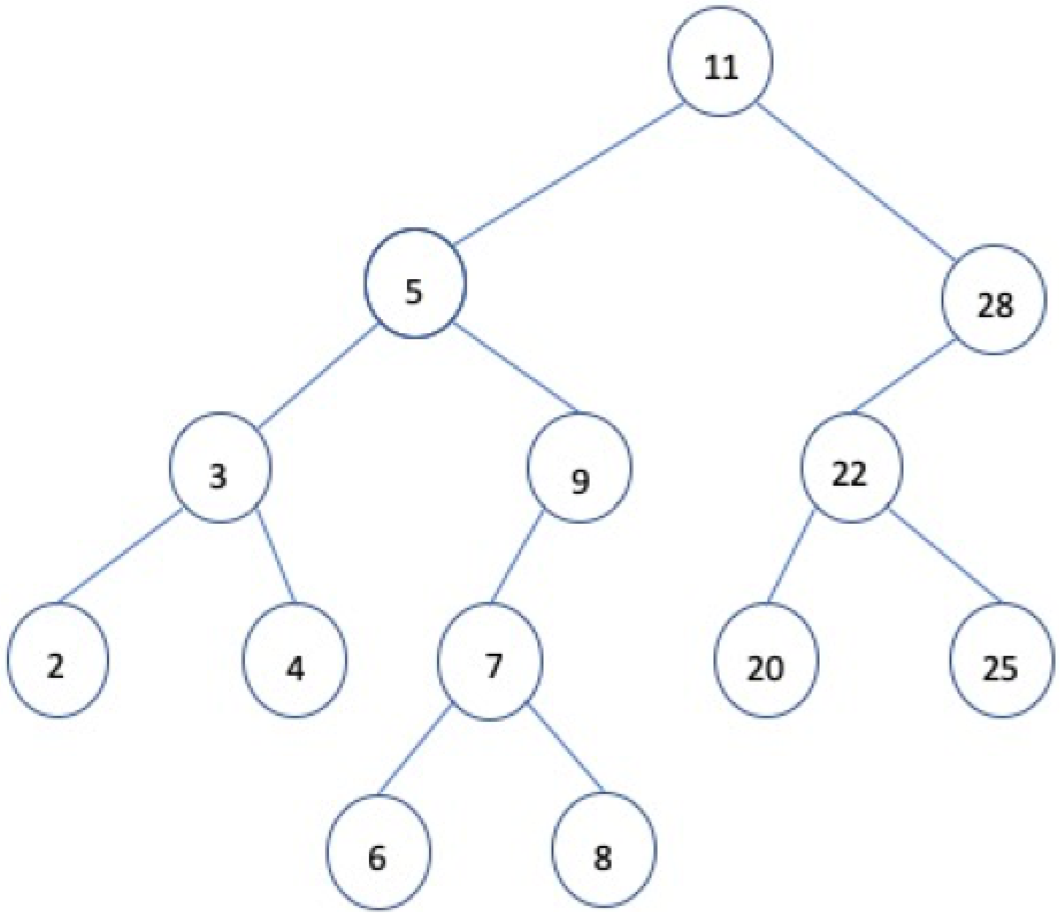
[✓] 

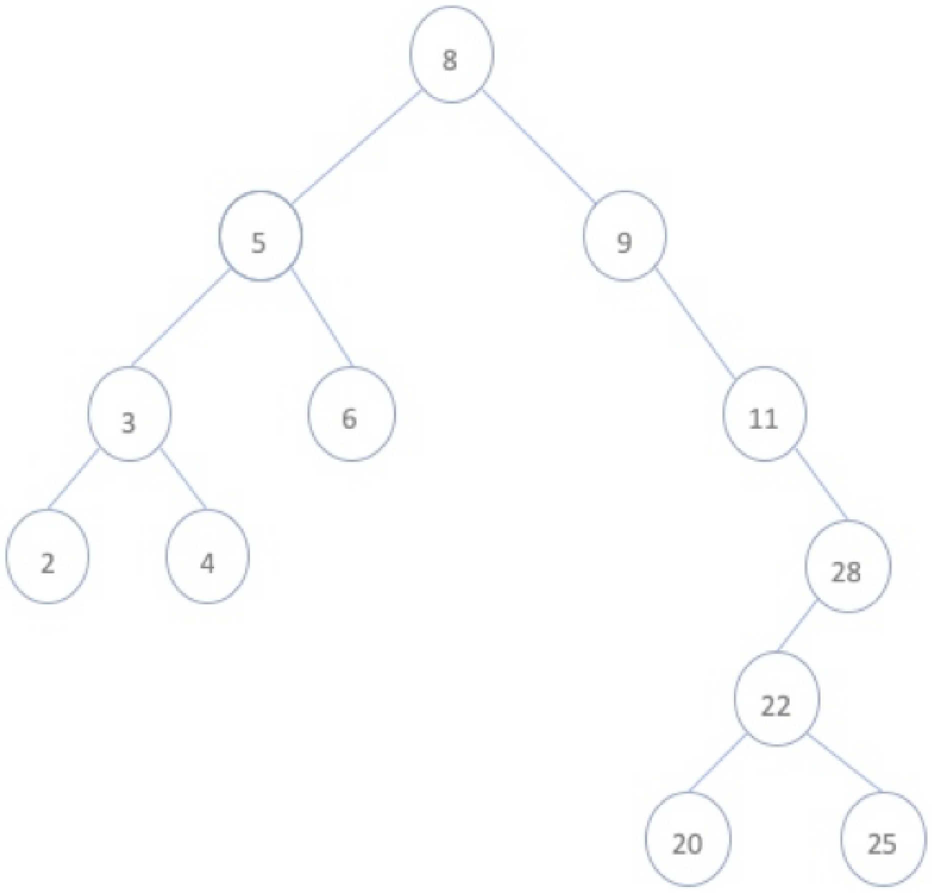


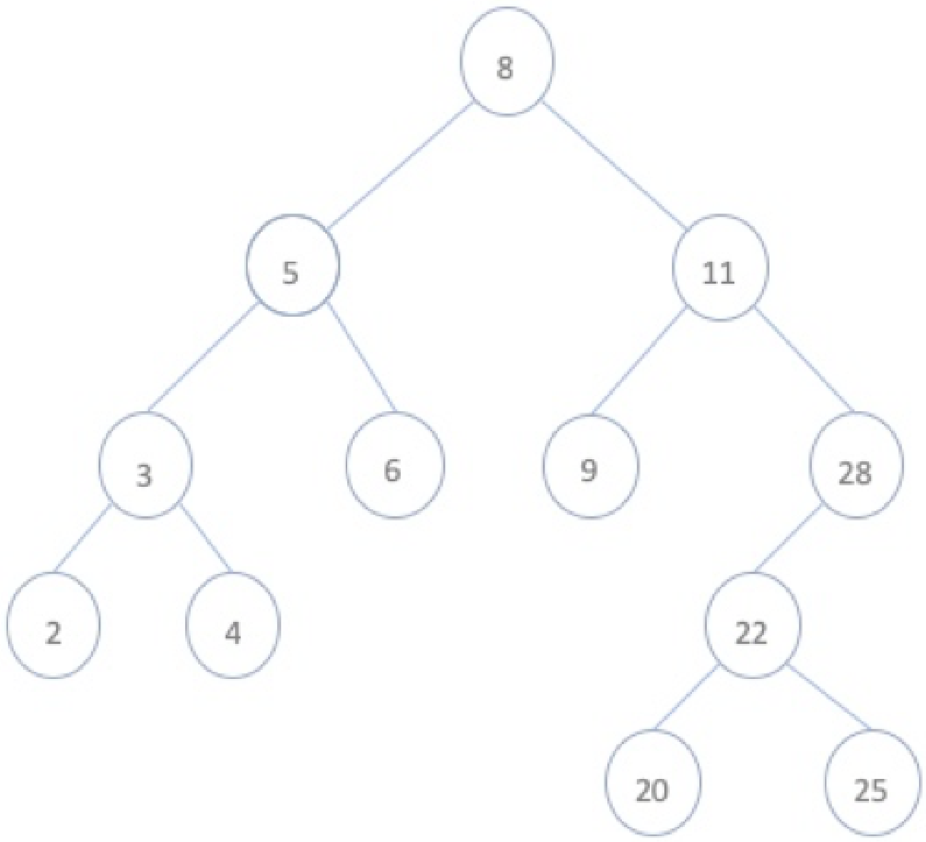
Question 9

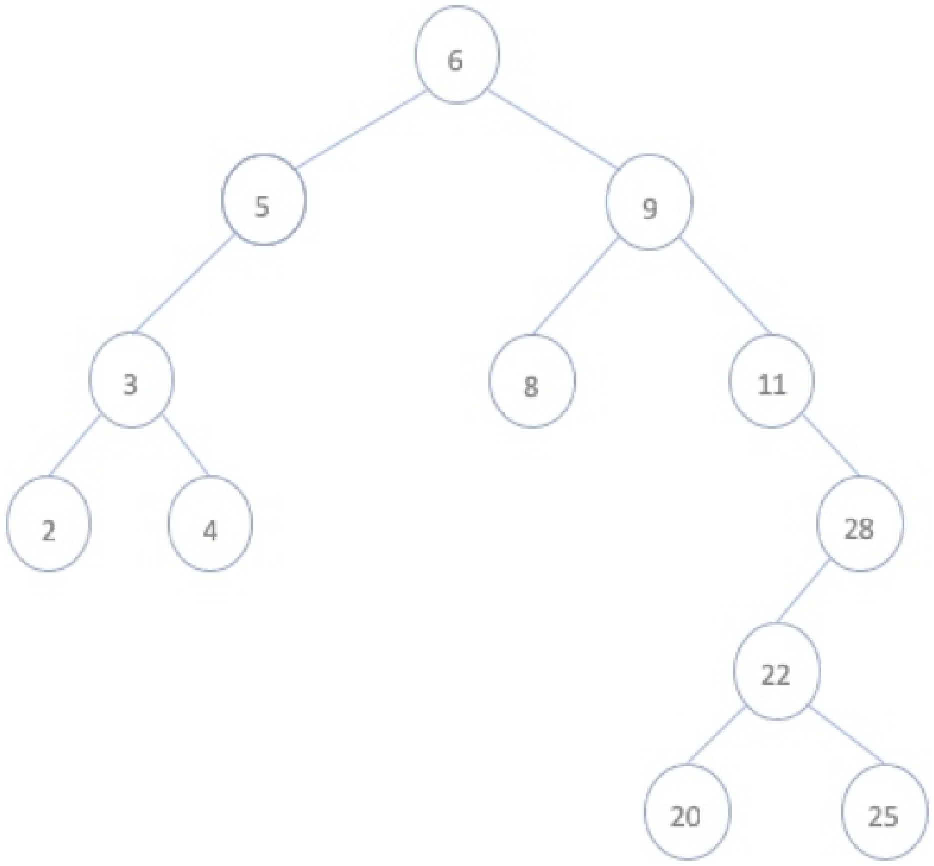
1 / 1 pts

Given the following splay-tree *T*, perform the operation DELETE(*T*,7). Recall that the SPLAY-DELETE operations splays the *predecessor* of the deleted node. Identify the correct resultant configuration.


11
5
28
3
9
22
2
4
7
20
25
6
8

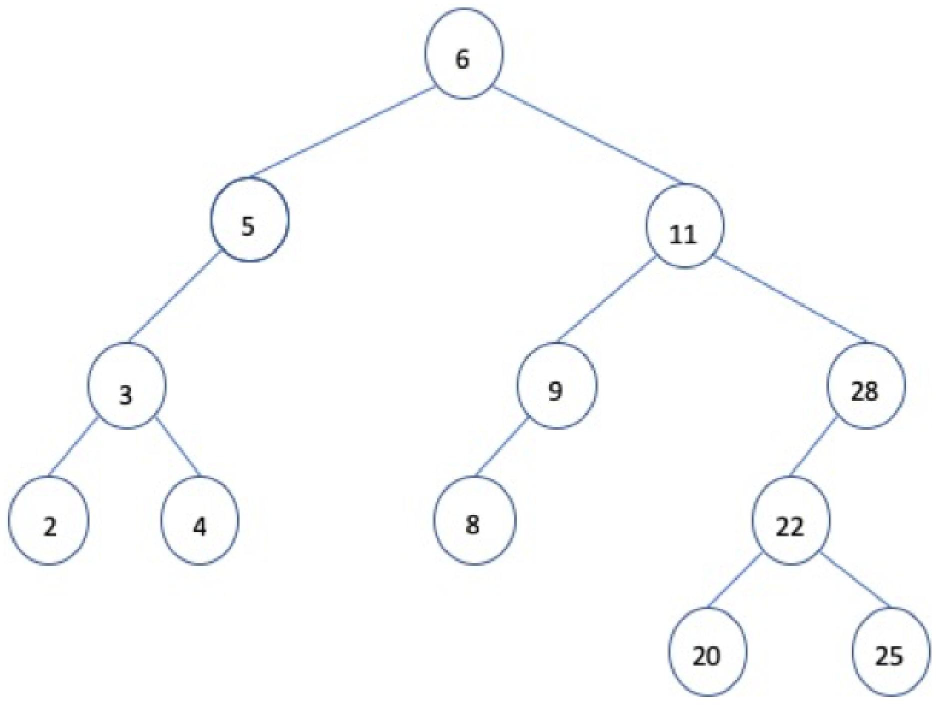

8
5
9
3
6
11
2
4
28
22
20
25


8
5
11
3
6
9
28
2
4
22
20
25


6
5
9
3
8
11
2
4
28
22
20
25

Correct!

[✓]



Question 10

1 / 1 pts

Why would one want to bother using splay trees when operations on data structures such as AVL trees and red-black trees have guaranteed logarithmic time complexity?

- The maintenance routines for AVL and red-black trees require logarithmic time, so operations on frequently accessed nodes will be needlessly expensive.
- For some instances, splay-trees can execute operations in guaranteed sub-logarithmic time.

Correct!

- [✓ There exist many real-world cases wherein a relatively small proportion of total data is accessed most frequently.]

○ Because red-black and AVL trees require variables to keep track of colors and subtree heights respectively, the asymptotic space complexity for splay-trees is smaller.

Question 11

1 / 1 pts

We introduce an operation called SPLAY-KEY-INCREASE(T, ℓ, r), that takes a splay tree T of integer-labelled nodes, locates the node with label ℓ and increases it's label by integer r. The algorithm works as follows: We begin at the root of T, walk down to a node with label ℓ, increase the label by value r, then splay this node, which now has label $\ell + r$. If a node label ℓ does not exist, then the algorithm simply splays the leaf node it ends its walk on. Can we use the amortized bounds seen in lecture to conclude that the amortized cost of SPLAY-KEY-INCREASE is $O(\log n)$ where n is the number of nodes in T? Explain.

○ Yes. Since if the label given in the function arguments exists in T, then the node with that label is splayed.

○ No. The amortized bounds for splay-tree operations are $\Omega(n \log n)$.

○ No. The amortized bounds for splay tree operations seen in lecture only applies to the basic BST operations search, insert and delete.

Correct!

[✓ Yes. The bounds of the theorem described in lecture applies to any operation wherein we walk from the root node of the splay tree down to some node which is then splayed. Since the work performed on the target node requires constant time, the amortized cost of the operation is $O(\log n)$.]

Question 12

1 / 1 pts

Decide whether the following statement is true or false. The process of amortized analysis is applicable only to algorithms that contain randomized elements, can be described as computing the expected running time of such algorithms with respect to their random elements.

Correct!

[✓ False]

True

Question 13

0.33 / 1 pts

Of the following statements regarding our analysis of splay trees seen in lecture, mark all that are **true** regarding a splay tree on n nodes.

orrect Answer

☐ The amortized cost of a splay operation is $O(n \log n)$.

ou Answered

☑ The largest possible rank of a node v is n, which is the case whenever v is the root node.

☐ The smallest possible rank of any node is 1.

Correct!

☑ The minimum height of any splay tree is $\Omega(\log n)$.

Correct!

☑ The amortized cost of a splay operation is $O(\log n)$.

Quiz Score: **12.33** out of 13