CSE 551 Practice Quiz 3 Solutions

Jamison Weber

June 1, 2021

Question 1

Which of the following statements are true? i.) $O(n)$ amortized and $\Theta(n)$ both must indicate average-case time complexities for some input size n. ii.) Amortized analysis can be applied effectively to any algorithm.

1. None of these
2. i. only
3. ii. only
4. i. and ii.

Rationale

Regarding i., it is indeed true that amortized time can be thought of as an average in some sense. Not in terms of expectation of a random distribution, but rather a weighted average of more precisely calculated costs incurred by a set of operations. Big Θ notation, on the other hand, does not have anything to do with average costs. A function is said to be $\Theta(f(n))$ if and only if it is bounded asymptotically from above by $O(f(n))$ and from below by $\Omega(f(n))$. Thus statement i. is false. Regarding ii., in order for amortized analysis to be meaningful, the algorithm generally must be of the form of a set of operations applied to a data structure multiple times. Consider the binary counter, where the data structure is a bit string, and we amortized the cost of multiple increment operations to obtain an average. You will also see in your homework assignment that problem 1 can be thought of as conducting a series of insert operations to a dynamic array, problem two is a set of multiple operations applied to a stack data structure, and problem 3 concerns a set of operations over a heap. Now as an example where amortized analysis makes little sense, consider what it would mean to amortize the cost of a sorting algorithm over an array. Once the sorting algorithm has terminated, the array is sorted. No further calls to sorting are necessary, since this would not change the sorted status of an array. In the case of quick sort, we know that in the worst-case quick sort terminates in $O(n^2)$ time when the elements are in reverse order, but $O(n)$ with the array is already

sorted. But since the input array must be general, we may not assume anything about its structure, whereas, with the binary counter and homework problems, there exist structural elements we may exploit to compute our costs.

2

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.
$\text{Cost}(O_k) = \begin{cases} k, if\ k\ is\ a\ power\ of\ 2 \\ 1, otherwise \end{cases}$
What is the minimum round dollar amount you will have to pay per operation to ensure your bank account never becomes overdrawn?

1. $3
2. $2
3. $4
4. $5

Rationale

Suppose we choose to spend z dollars per operation. Consider the savings incurred from iterations between powers 2^{k-1} and 2^k, and note that each operation in between these powers has a cost of 1. Thus we save a total of $z-1$ dollars per interim operation, and when operation 2^k comes we will have saved $(z-1)(2^k - 2^{k-1} - 1)$ dollars. Now we just have to ensure that this savings is at least 2^k to afford the operation. Find the value of z that satisfies this. Thus we have

$$(z-1)(2^k - 2^{k-1} - 1) \geq 2^k \implies z \geq \frac{2^k}{2^k - 2^{k-1} - 1} + 1$$

We note that the right-hand side is a decreasing function, and so we need only take the limit as k approaches infinite to find the smallest value on the RHS. Then

$$\lim_{k\to\infty} \frac{2^k}{2^k - 2^{k-1} - 1} + 1 = \lim_{k\to\infty} \frac{1}{1 - \frac{1}{2} - \frac{1}{2^k}} + 1 = \frac{1}{1 - \frac{1}{2}} + 1 = 3$$

So we need at least three dollars since k was chosen arbitrarily.

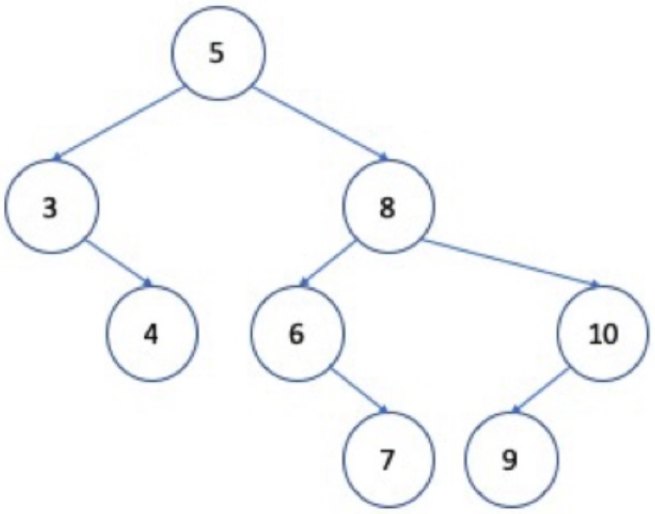
1. ii. only
2. i. only
3. i. and ii.
4. None of these

Rationale

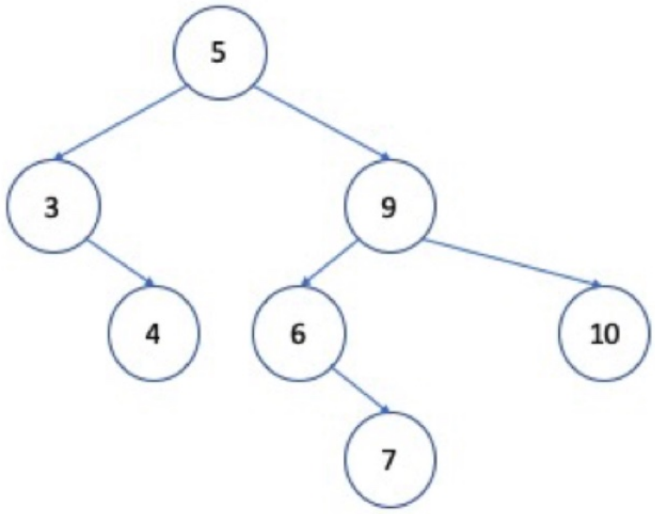
Statement i. is false. The whole point of a potential function is to feed it an operation index and it outputs the credit available for the operation at hand. It does not require evaluating the potential function for all previous steps. Statement ii. is true. A potential function must somehow reflect the configuration of the data structure at each operation index. Otherwise we would not be able to infer anything meaningful.

Question 4

Given the following binary search tree T, identify the configuration of T after the operation TREE-DELETE(T,8). Assume the delete operation uses successors to replace deleted nodes.



Correct Response

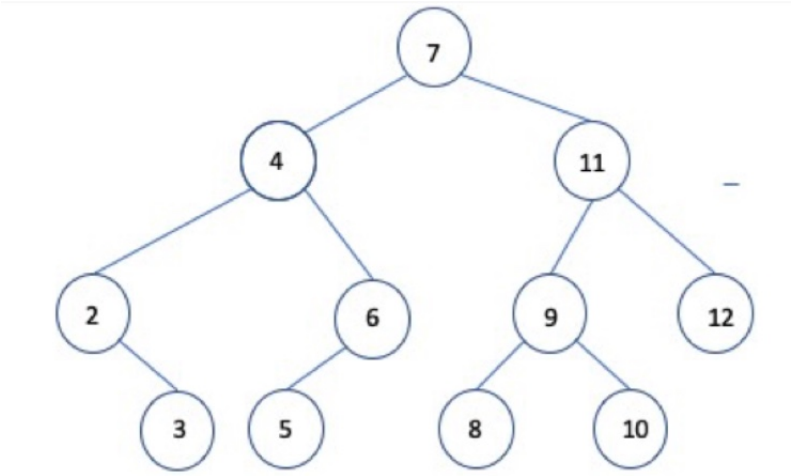


Rationale

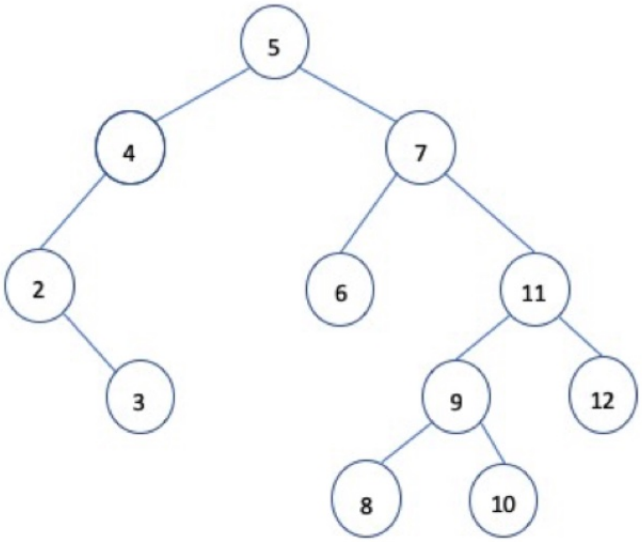
Recall the simple tree delete operation involves simply deleting the target node and replacing it with its successor (or possibly predecessor if defined that way). The successor of 8 in this tree is 9. In general, the successor of a binary search tree is found by taking the right junction until the first left junction is available, and then following the left junctions until no further left junctions are available.

Question 5

Given the following splay-tree T, perform the operation SPLAY-SEARCH(T,5).

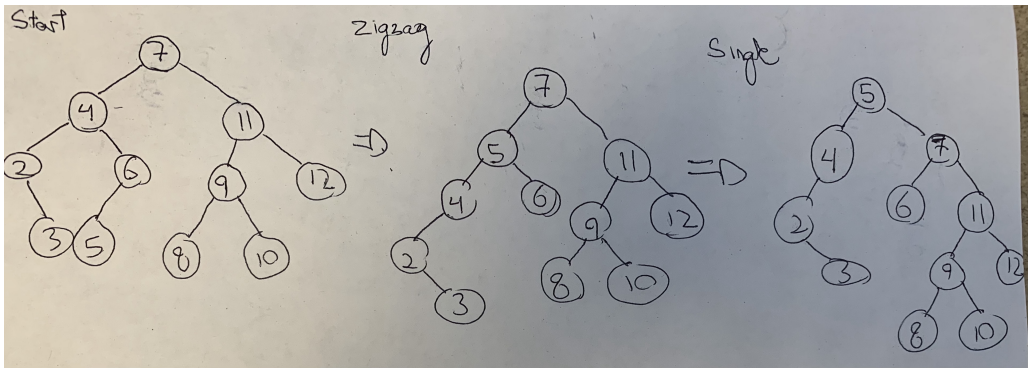


Correct Response



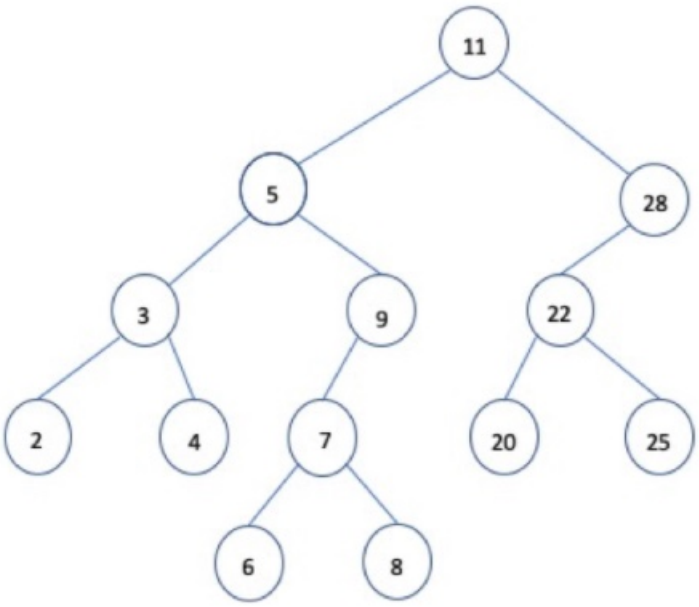
Rationale

Follow the steps in the diagram below:

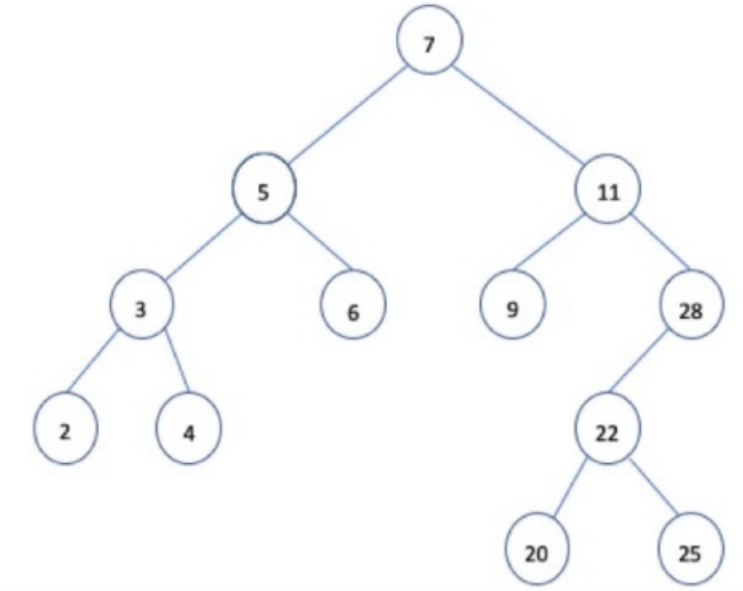


Question 6

Given the following splay-tree T, perform the operation SPLAY-DELETE(T,8). Assume the delete operation uses predecessors to replace deleted nodes.

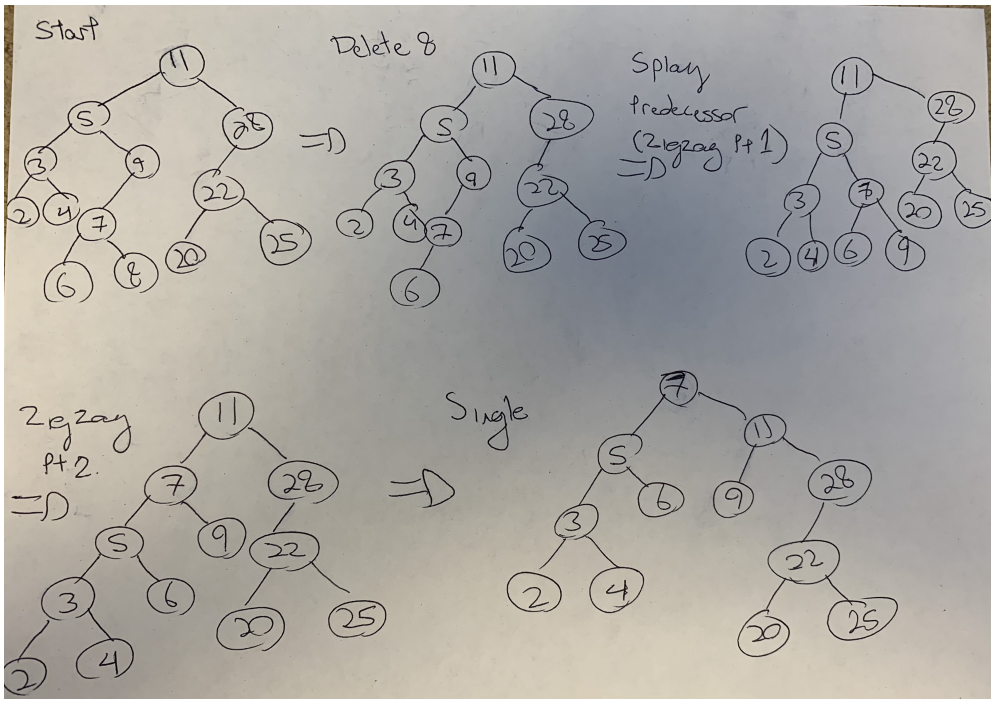


Correct Response



Rationale

Follow the steps in the diagram below:



Question 7

For which kinds of applications would splay trees be best suited?

1. Applications where a relatively small subset of values are accessed more frequently than the rest
2. Storing a large number of cryptographic keys for savings accounts
3. Storing accounting transaction records where the keys are the number of days since the ledger was created

Rationale

It is evident from the behavior of splay trees, the elements accessed frequently will generally be near the root, as they are splayed at each access. Thus the cost of accessing these elements is generally low, thus statement 1 is correct. The idea with statement 2 is that savings accounts are generally not accessed very often, and so it will likely be expensive to access a savings account key in a splay tree, so statement 2 is false. Regarding statement 3, if you insert items into a splay tree in lexicographical order of the keys, then we see that the splay tree just ends up being a line, and therefore a very unbalanced tree, which isn't ideal.

Question 8

Suppose we have a splay tree of n nodes such that the number of nodes at each level i is 2^i and the tree is rooted at node x. Which of the following is the rank of the predecessor of x?

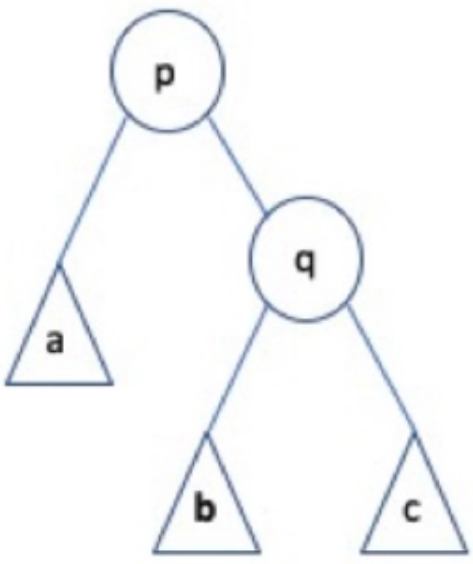
1. 0
2. 1
3. $\log(n-1)$
4. $\log n$

Rationale

Recall that the rank of a node v is defined as the floor of the log of the number of nodes in the subtree rooted at v. But if v is the predecessor of the root, and each level of the tree is full, then v necessarily has no children, and thus the rank of v is $\lfloor \log 1 \rfloor = 0$.

Question 9

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



1. $3 + r'(q) - r(q)$
2. $3 + r'(p) - r(p)$
3. $3 + 3(r'(q) - r(q))$
4. $3 + 3(r'(p) - r(p))$

Rationale

We consider the bounds given for the single rotation in the proof of the amortized cost of splay-tree operations, and select the lowest bound of the answers available in the multiple choice question. The following are the calculated bounds for a single rotation:

$$a_t = 1 + r'(v) + r'(y) - r(v) - r(y) \leq_{(*)} 1 + r'(v) - r(v) \leq_{(**)} 1 + 3(r'(v) - r(v))$$

And for your reference, here is the tree from the proof.

Case 1: Single Rotation

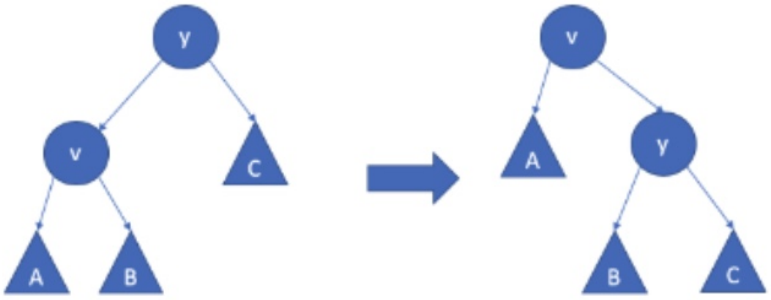


Figure 1: Figure 1: Single right-rotation at node v

Of the four answer choices, we seek the lowest bound on $1 + r'(v) + r'(y) - r(v) - r(y)$. We just need to map the vertices v, y to p, q. Nodes v and q begin at the bottom, so v maps to q, and so y maps to p, as both of these are the nodes that end up at the bottom after the rotation. Finally, since the cost of a single rotation is assumed to be 3, we replace 1 with 3. Substituting nodes and values gives us:

$$3 + r'(q) = r(q)$$

Question 10

Suppose we introduce a new operation for a splay tree T called SPLAY-INCREASE-KEY(T,node x, v). This operation will access a node x, increase its key by amount v, then splay x. If the SPLAY-INCREASE-KEY routine reaches a leaf node without finding a node x, then it simply splays the leaf it landed on to the root. Respond True or False for both of the following statements in the form (response_stmt_1,response_stmt_2) and explain.
1.) The amortized cost of this new operation is $O(log(n))$.
2.) T is still guaranteed to be a valid binary search tree.

1. (True,False) This new operation follows the required steps necessary to guarantee logarithmic amortized time complexity, but altering the key of x may cause a violation of the BST property.

2. (True,True) This new operation follows the required steps necessary to guarantee logarithmic amortized time complexity, and the rotations executed during the splay will ensure the BST property is always satisfied.

3. (False,True) We cannot guarantee an amortized logarithmic time complexity for this new operation because the height of the splay tree is n in

the worst case, and the rotations executed during the splay will ensure the BST property is always satisfied.

4. (False,True) We cannot guarantee an amortized logarithmic time complexity for this new operation because the height of the splay tree is n in the worst case, and the rotations executed during the splay will ensure the BST property is always satisfied.

Rationale

Answer choice one is correct. Remember that in order for the amortized splay bounds to apply according to the proof seen in lecture, each operation must walk down from the root and select some node x and it necessarily *must* splay x. This algorithm does that, however, it is true that altering the key may not necessarily preserve the BST property, for example, if the key increase makes the key larger than any other key in the tree and after the splay operation, the root still has a right-child.