Part A: Stacks and Queues

For questions in this part, consider the following code:

```
MysterySequence seq = new MysterySequence();
seq.addSomething("S");
seq.addSomething("C");
System.out.print(seq.removeSomething());
seq.addSomething("A");
seq.addSomething("R");
System.out.print(seq.removeSomething());
seq.addSomething("E");
System.out.print(seq.removeSomething());
System.out.print(seq.removeSomething());
seq.addSomething("M");
System.out.print(seq.removeSomething());
```

Question 1 [**5 points**]

What is printed if `MysterySequence` is a `Queue`, and `addSomething` and `removeSomething` are `enqueue` and `dequeue` respectively? What are the contents of the `Queue` after the code is run?

`Printed:`

`Contents:`

Answer

Printed: VARIANT A: SCARE, B: ALERT, C: TACOS, D: LEECH
Contents: VARIANT A: M, B: Y, C: S, D: R

Point Breakdown

- **(+4 pt)** For correct printed string
- **(+1 pt)** For correct remaining contents

Question 2 [**5 points**]

What is printed if `MysterySequence` is a `Stack`, and `addSomething` and `removeSomething` are `push` and `pop` respectively? What are the contents of the `Stack` after the code is run?

`Printed:`

`Contents:`

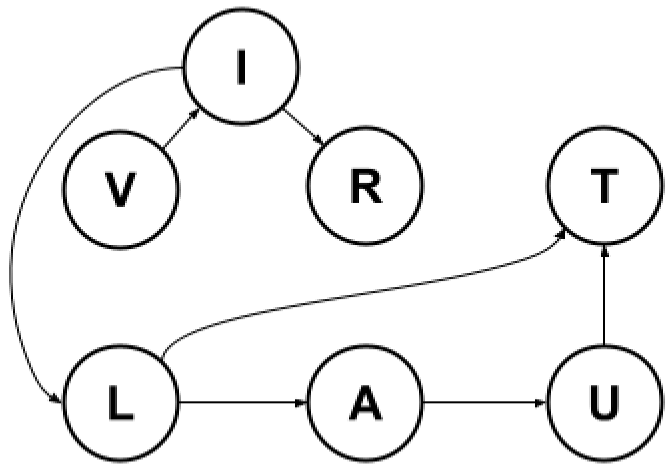
Answer

Printed: VARIANT A: CREAM, B: RELAY, C: COATS, D: CHEER
Contents: VARIANT A: S, B: T, C: S, D: L

Point Breakdown

- **(+4 pt)** For correct printed string
- **(+1 pt)** For correct remaining contents

Part B: Graphs



Imagine we are doing a Breadth-First Search of the above graph starting from node `V`.

Question 1 [**5 points**]

What implementation of the `Graph` ADT should we use to store the graph in order to get the most efficient runtime for our BFS?

Answer

AdjacencyList

Point Breakdown

- **(5 pt)** Correct answer given

Question 2 [**5 points**]

What ADT should we use to store the TODO list of vertices we still need to explore in order to guarantee we search the graph in the correct order?

Answer

Queue

Point Breakdown

- **(5 pt)** Correct answer given

Question 3 [**5 points**]

What is the most efficient runtime of BFS as described in class?

Answer

$O(n + m)$ or $O(|V| + |E|)$

Point Breakdown

- **(5 pt)** Correct answer given

For the next two questions, you may assume that if you ever have to add more than one vertex at a time to our TODO list, that we add them in alphabetical order. You may also assume that vertices are marked as `VISITED` right after we add them to our TODO list. The starting vertex of the search is `V`.

Question 4 [**5 points**]

At the point in the search where vertex `T` is marked as `VISITED`, which other vertices will already have been marked as `VISITED`?

Answer

VARIANT A: V, I, L, R, A
VARIANT B: V, S, R, E, C
VARIANT C: V, E, G, I, N
VARIANT D: V, O, L, E, G

Point Breakdown

- **(5 pt)** For a correct answer. Deduct 2 points for each extra vertex listed, and each vertex missing. Ignore T if they included it. Order doesn't matter.

Question 5 [**5 points**]

At the point in the search where vertex `T` is marked as `VISITED`, which vertices (including `T`) will be in our TODO list? Make sure to write out the contents of the TODO list in order.

Answer

VARIANT A: R A T
VARIANT B: R C T
VARIANT C: N T
VARIANT D: L G T

Point Breakdown

- **(5 pt)** For a perfect answer in perfect order.
- **(4 pt)** For all of the correct vertices, but in the wrong order.
- **(3 pt)** Correct vertices in correct order, but with one extra or one missing vertex.
- **(2 pt)** Correct vertices in wrong order, but with one extra or one missing vertex.

Part c: PA2 Review

In PA2 we used a `PriorityQueue` to store intersections ordered by distance from the starting intersection. For example, if we push `("A", 42)` to the `PriorityQueue` it means the intersection labeled `"A"` is at a distance of `42` from the starting vertex. You can assume that the implementation of `PriorityQueue` is the array-based `Heap` we discussed in lecture.

Question 1 [**5 points**]

At what point during the execution of Djikstra's algorithm are we able to consider a particular intersection `VISITED`?

Answer

We can consider a vertex to be VISITED when it is dequeued/popped/removed from the Heap

Point Breakdown

- **(5 pt)** Answer is correct
- **(3 pt)** States that it can be marked as VISITED when it is the closest vertex to the starting point (or something similar), but fails to point out at what point of the algorithm this is known.

Question 2 [**5 points**]

If multiple intersections get pushed with the same priority, can we assume they will also be removed from the `Heap` in the same order that they were pushed? In at most 2 or 3 sentences explain why or why not?

Answer

No. Consider pushing (”C”,42),(”A”,42)(”T”,42) in that order. They will be added to the array in that order, but when the (”C”,42) entry is removed, it will be replaced by the (”T”,42) as the next item to be removed.

Point Breakdown

- **(5 pt)** Correct answer (no) with a good explanation that either explains that when removing from the Heap, the last element in the Array is pulled up to the top (and then fixed down), which could potentially perturb the order of removal for multiple items with identical prio OR gives a specific example that shows when order would not be preserved.
- **(1 pt)** Correct answer (no) without an explanation or with an incorrect explanation.

Part D: Applied Runtimes

For each of the following operations, give the worst-case runtime **in terms of** n for the listed data structures. In situations where the amortized runtime is smaller than the unqualified runtime, give the amortized runtime.

You may assume that sorted lists are sorted from smallest to largest. You may assume you have pointers to the tail of your LinkedLists.

Question 1 [**5 points**]

Find the smallest element in a...

`Sorted ArrayList:`

`Sorted LinkedList:`

`Min Heap:`

`General BST:`

`Balanced BST:`

Answer

$O(1)$,$O(1)$,$O(1)$,$O(n)$,$O(\log(n))$

Point Breakdown

- **(+1 pt)** For each correct answer

Question 2 [**5 points**]

Remove the smallest element (without having a prior reference to it) from a...

`Sorted ArrayList:`

`Sorted LinkedList:`

`Min Heap:`

`General BST:`

`Balanced BST:`

Answer

$O(n)$,$O(1)$,$O(\log(n))$,$O(n)$,$O(\log(n))$

Point Breakdown

- **(+1 pt)** For each correct answer

Question 3 [**5 points**]

Find the largest element in a...

`Sorted ArrayList:`

`Sorted LinkedList:`

`Min Heap:`

`General BST:`

`Balanced BST:`

Answer

$O(1)$,$O(1)$,$O(n)$,$O(n)$,$O(\log(n))$

Point Breakdown

- **(+1 pt)** For each correct answer

Question 4 [**5 points**]

Remove the largest element (without having a prior reference to it) from a...

`Sorted ArrayList:`

`Sorted LinkedList:`

`Min Heap:`

`General BST:`

`Balanced BST:`

Answer

$O(1)$,$O(1)$,$O(n)$,$O(n)$,$O(\log(n))$

Point Breakdown

- **(+1 pt)** For each correct answer

Question 5 [**5 points**]

Insert an arbitrary new element into a...

`Sorted ArrayList:`

`Sorted LinkedList:`

`Min Heap:`

`General BST:`

`Balanced BST:`

Answer

$O(n)$,$O(n)$,$O(\log(n))$,$O(n)$,$O(\log(n))$

Point Breakdown

- **(+1 pt)** For each correct answer

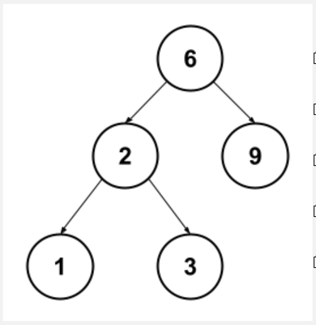
Part e: Trees

For each of the following trees, indicate which (if any) of the listed constraints apply. In each case, mark **all** that apply.

Question 1 [**5 points**]

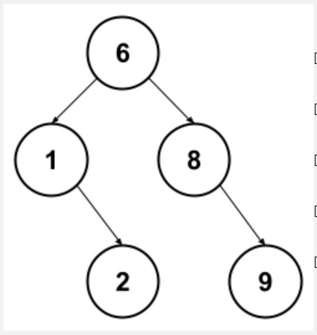
Answer

A



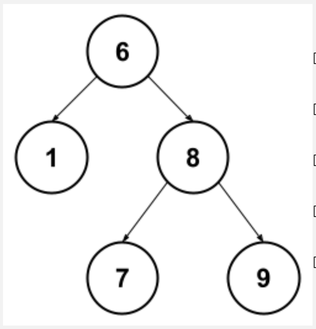
- [✓ BST]
- ☐ Min Heap
- [✓ Complete]
- [✓ AVL balance]
- [✓ RB colorable]

B



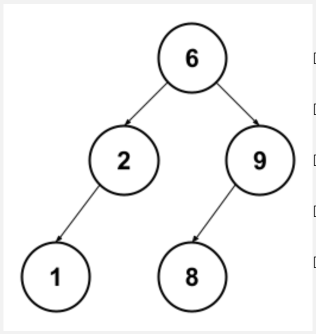
- [✓ BST]
- ☐ Min Heap
- ☐ Complete
- [✓ AVL balance]
- [✓ RB colorable]

C



- [✓ BST]
- ☐ Min Heap
- ☐ Complete
- [✓ AVL balance]
- [✓ RB colorable]

D



- [✓ BST]
- ☐ Min Heap
- ☐ Complete
- [✓ AVL balance]
- [✓ RB colorable]

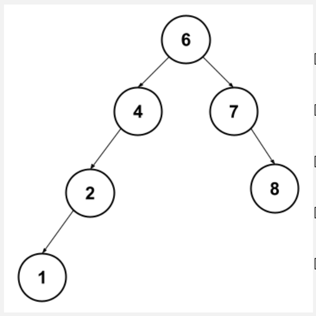
Point Breakdown

- **(5 pt)** If perfect. Deduct 2 per extra check and 2 per missing check.

Question 2 [5 points]

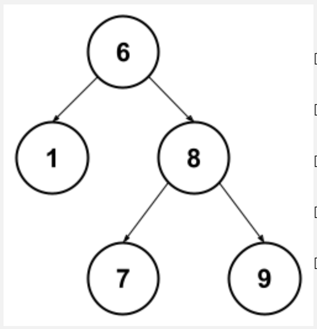
Answer

A



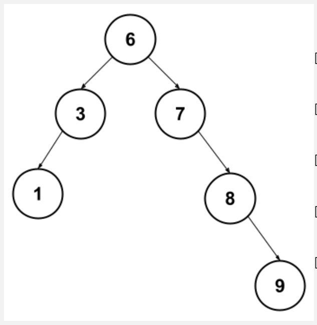
- [✓] BST
- ☐ Min Heap
- ☐ Complete
- ☐ AVL balance
- ☐ RB colorable

B



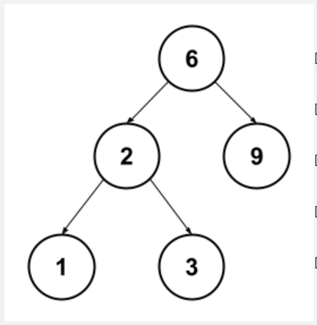
- [✓] BST
- ☐ Min Heap
- ☐ Complete
- [✓] AVL balance
- [✓] RB colorable

C



- [✓] BST
- ☐ Min Heap
- ☐ Complete
- ☐ AVL balance
- ☐ RB colorable

D



- [✓] BST
- ☐ Min Heap
- [✓] Complete
- [✓] AVL balance
- [✓] RB colorable

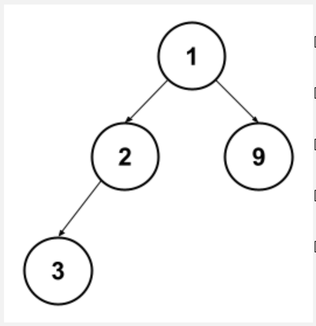
Point Breakdown

- **(5 pt)** If perfect. Deduct 2 per extra check and 2 per missing check.

Question 3 [5 points]

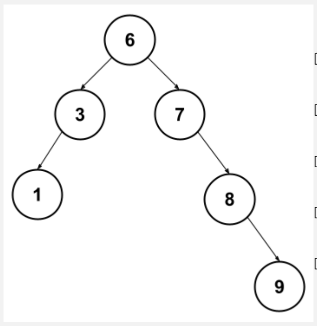
Answer

A



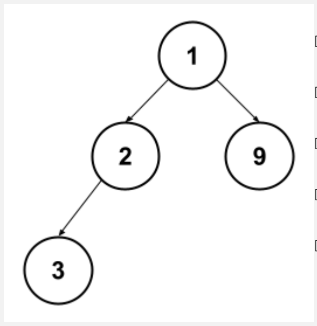
- ☐ BST
- ✓ Min Heap
- ✓ Complete
- ✓ AVL balance
- ✓ RB colorable

B



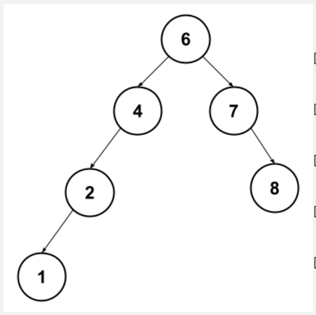
- ✓ BST
- ☐ Min Heap
- ☐ Complete
- ☐ AVL balance
- ☐ RB colorable

C



- ☐ BST
- ✓ Min Heap
- ✓ Complete
- ✓ AVL balance
- ✓ RB colorable

D



- ✓ BST
- ☐ Min Heap
- ☐ Complete
- ☐ AVL balance
- ☐ RB colorable

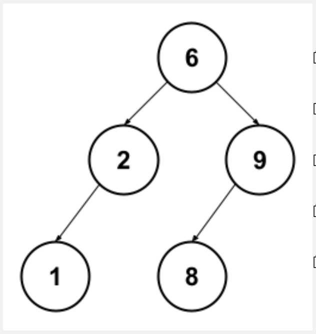
Point Breakdown

- **(5 pt)** If perfect. Deduct 2 per extra check and 2 per missing check.

Question 4 [5 points]

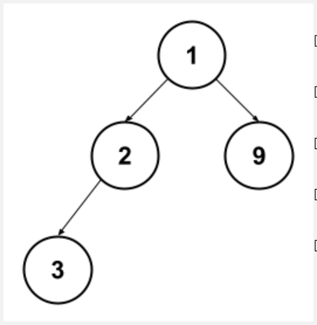
Answer

A



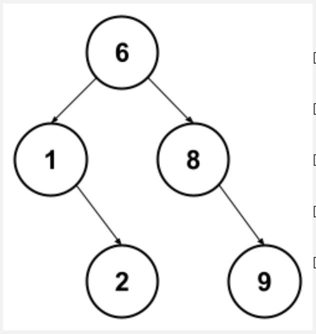
- ✓ BST
- ☐ Min Heap
- ☐ Complete
- ✓ AVL balance
- ✓ RB colorable

B



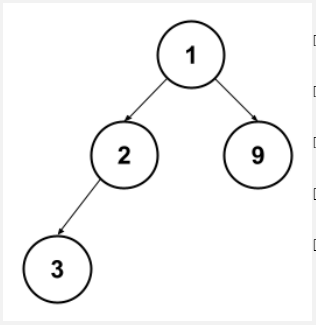
- ☐ BST
- ✓ Min Heap
- ✓ Complete
- ✓ AVL balance
- ✓ RB colorable

C



- ✓ BST
- ☐ Min Heap
- ☐ Complete
- ✓ AVL balance
- ✓ RB colorable

D



- ☐ BST
- ✓ Min Heap
- ✓ Complete
- ✓ AVL balance
- ✓ RB colorable

Point Breakdown

- **(5 pt)** If perfect. Deduct 2 per extra check and 2 per missing check.

Part f: Miscellaneous

Question 1 [**5 points**]

Does DFS always find the shortest paths in an **unweighted tree**? In at most one sentence explain why or why not?

Answer

Yes. In a tree there is only one path between each pair of vertices, and DFS will find that path.

Point Breakdown

- **(5 pt)** For correct answer (yes) and an explanation that somehow identifies that the shortest path is the only path.
- **(1 pt)** For just a correct answer (yes) with no explanation or an incorrect explanation.

Question 2 [**5 points**]

In class we saw that we could store a Heap with n elements in an `ArrayList` of size $O(n)$. Can we apply the same technique to store general BSTs in an `ArrayList` of size $O(n)$? In at most 2 or 3 sentences explain why or why not?

Answer

No. An arbitraty BST with n elements may have up to n levels, which would require 2^n space to store using the same scheme as we used for Heaps.

Point Breakdown

- **(5 pt)** For correct answer (no) and an explanation that somehow identifies that the BST would require 2^n space, or that the amount of empty array elements would be $> O(n)$.
- **(4 pt)** For a correct answer (no) with an explanation that hints at the fact that there would be "a lot" of empty space (ie they point out that there could be n levels), but does not quite get all the way there.
- **(1 pt)** for just a correct answer (no) with an explanation, or with a bad explanation.
- **(1 pt)** for an incorrect answer (yes), but with an explanation that demonstrates an understanding of how the tree would be stored, but fails to recognize that it would take more than $O(n)$ space.

Part g: Bonus

Question 1 [**5 points**]

Draw a non-empty tree that meets all requirements of both a Max Heap and a BST.

Answer

A drawing of a single node tree.
OR
A drawing of a two node tree where the left child is smaller than the root.

Point Breakdown

- **(5 pt)** A correct tree is drawn