Part a: Trees

For questions in this part, you will use the following values: `[1,2,3,5,7,9,11,16,17,18]`

Question 1 [**5 points**]

Draw a Max Heap (as a tree) containing the values above. It can be any valid Max Heap that contains exactly the above values.

Answer

Must draw a complete binary tree (will have 4 levels, the first 3 are full, the last is filled left to right), such that every parent is greater than or equal to its children. It must contain exactly 10 nodes matching the given values.
A: 1,2,3,5,7,9,11,16,17,18
B: 2,3,4,6,8,10,12,15,16,17
C: 1,2,3,6,8,10,12,16,17,18
D: 2,3,4,5,7,9,11,15,16,17

Point Breakdown

- **(+2 pt)** If the structure is correct (they drew a complete tree)
- **(+2 pt)** If there is a valid heap ordering (either min or max heap)
- **(+1 pt)** If the answer is perfect (it is complete AND a MAX heap specifically)
- **(-1 pt)** Take off a point for each value that is missing, or each extra value added

Question 2 [**5 points**]

Write out the ArrayList representation of the Heap you drew in the previous question.

Answer

Must write out an array with ten elements that is based on their answer to Q1. The root of their tree must be index 0, the two children of the root should be indices 1 and 2, etc.

Point Breakdown

- **(2 pt)** The array they wrote out represents a valid Max Heap, but it is not what they drew for Q1
- **(5 pt)** The array the wrote out exactly represents the tree THEY drew in Q1

Question 3 [**10 points**]

Draw an AVL tree containing exactly the values above. It can be any valid AVL tree that contains exactly the above values.

Answer

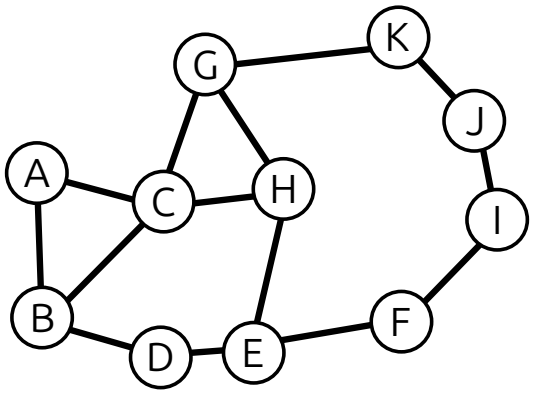
Must draw a valid AVL tree. It must be a binary search tree (each node partitions its descendents into those that are smaller on the left, and larger on the right). Each node in the tree must also have a balance factor of either -1, 0, or 1.

Point Breakdown

- **(+5 pt)** If they drew a valid BST (take off a point if there is only ONE minor error)
- **(+5 pt)** If every node has a balance factor of 0, 1, or -1 (take off a point for each node that does not have a correct balance factor up to a max of -5)

Part B: Graphs

The following questions pertain to the graph below. Where relevant, assume that the `incidentEdges` method returns out-edges in **alphabetical order** of the opposite vertex. e.g., `C.incidentEdges()` would return `A, B, G, H`.



Question 1 [**10 points**]

Perform a **Breadth-First Traversal** of the graph starting at vertex **A** and provide a list of every vertex visited in the order in which they are **removed** from the Queue .

Answer

Variant A: A, B, C, D, G, H, E, K, F, J, I
Variant B: A, B, C, G, H, E, D, F, I, J, K
or A, C, H, E, F, I, J, D, G, B, K
Variant C: H, C, E, G, A, B, D, F, K, I, J
Variant D: D, B, A, C, G, H, E, F, I, J, K
or D, E, H, G, K, J, I, C, A, F, B

Point Breakdown

- **(10 pt)** A completely correct answer.
- **(5 pt)** An answer that starts from the right node and performs a breadth/depth first search in a different order.

Question 2 [**5 points**]

Based on your answer above, is the undirected edge connecting vertices **A** and **C** a *spanning* edge (part of the spanning tree created by the traversal) or a *cross/back* edge (not part of the spanning tree). (Circle One)

Spanning Edge **Cross/Back Edge**

Answer

Variant A: Spanning
Variant B: Spanning
Variant C: Spanning
Variant D: Spanning

Point Breakdown

- **(5 pt)** An answer that is consistent with the answer to Q1.

Question 3 [**5 points**]

Based on your answer above, is the undirected edge connecting vertices **J** and **I** a *spanning* edge (part of the spanning tree created by the traversal) or a *cross/back* edge (not part of the spanning tree). (Circle One)

Spanning Edge **Cross/Back Edge**

Answer

Variant A: Back
Variant B: Spanning
Variant C: Back
Variant D: Spanning

Point Breakdown

- **(5 pt)** An answer that is consistent with the answer to Q1.

Part C: Priority Queue

In class, we designed two priority queues using a linked list, and one using a heap structure. We also discussed several forms of binary search tree.

Question 1 [**5 points**]

Explain **in at most 2 sentences**, or a small block of pseudocode, how you would implement the `add` operation of the priority queue using a binary search tree. You may call (or reference) the three BST operations discussed in class (`find`, `insert`, and `remove`) *without* having to explain how they work.

Answer

- `insert` the element into the BST.

Point Breakdown

- **(5 pt)** An answer that simply calls insert.
- **(3 pt)** An answer that demonstrates awareness of the fact that both a priority queue and a BST require order.

Question 2 [**5 points**]

Provide an upper bound (i.e., Big-O) on the runtime complexity of your `add` operation for (i) A generic binary search tree, (ii) An AVL tree, and (iii) A Red-Black tree. Your answers should be in terms of the number of elements in the priority queue (N).

`Generic BST:`

`AVL Tree:`

`Red-Black Tree:`

Answer

Generic BST: $O(N)$
AVL Tree: $O(\log N)$
Red-Black Tree: $O(\log N)$

Point Breakdown

- **(+2 pt)** $O(N)$ for the generic BST
- **(+2 pt)** $O(\log N)$ for at least one BBST
- **(+1 pt)** $O(\log N)$ for the other BBST

Question 3 [**5 points**]

Explain **in at most 2 sentences**, or a small block of pseudocode, how you would implement the `remove` operation of the priority queue using a binary search tree. You may call (or reference) the three BST operations discussed in class (`find`, `insert`, and `remove`) *without* having to explain how they work.

Answer

- Find the left-most leaf in the BST. $O(d)$
- Remove the value at this leaf. $O(d)$

Point Breakdown

- **(5 pt)** An answer that removes the value at the left-most leaf.
- **(3 pt)** An answer that demonstrates awareness of the fact that both a priority queue and a BST require order.

Question 4 [**5 points**]

Provide an upper bound (i.e., Big-O) on the runtime complexity of your `remove` operation for (i) A generic binary search tree, (ii) An AVL tree, and (iii) A Red-Black tree. Your answers should be in terms of the number of elements in the priority queue (N).

```
Generic BST:          Generic BST:

AVL Tree:               AVL Tree:

Red-Black Tree:     Red-Black Tree:
```

Answer

Generic BST: $O(N)$
AVL Tree: $O(\log N)$
Red-Black Tree: $O(\log N)$

Point Breakdown

- **(+2 pt)** $O(N)$ for the generic BST
- **(+2 pt)** $O(\log N)$ for at least one BBST
- **(+1 pt)** $O(\log N)$ for the other BBST

Question 5 [**5 points**]

Bonus Question Due to a hardware component called the cache, in most practical settings, it can be significantly more expensive to access an object referenced by a pointer (e.g., following a pointer in a linked list node going from one node to another in a tree) than to access a specific index in a relatively small array. Give an asymptotic upper (i.e., Big-O) bound on the number of pointers accessed when calling `add` on a priority queue implemented using (i) The ArrayList-based heap discussed in class, and (ii) an AVL Tree, as you designed above.

`Heap`:

`AVL Tree`:

Answer

A heap requires no pointers: All memory is allocated in the same array. The AVL tree requires chasing pointers.
Heap: $O(1)$
AVL Tree: $O(\log N)$

Point Breakdown

- **(5 pt)** For a correct answer.

Part d: Runtimes

Question 1 [**10 points**]

What is the tight worst-case runtime to find a **specific value** in each of the following data structures?

Sorted ArrayList:

Sorted LinkedList:

Min Heap:

General BST:

Balanced BST:

Answer

Var A and C: O(log n), O(n), O(n), O(n), O(log n)
Var B and D: O(1), O(1), O(1), O(n), O(log n)

Point Breakdown

- **(+2 pt)** for each correct answer

Question 2 [**10 points**]

What is the tight worst-case runtime to remove a **specific value** from each of the following data structures if you do not have a reference to it or know the index of it?

Sorted ArrayList:

Sorted LinkedList:

Min Heap:

General BST:

Balanced BST:

Answer

Var A and B: O(n), O(n), O(n), O(n), O(log n)
Var C and D: O(n), O(1), O(log n), O(n), O(log n)

Point Breakdown

- **(+2 pt)** for each correct answer

Part e: Data Structure Design

Task Scheduler

You are implementing a task scheduler for a new web application. This component will keep track of a large number of tasks that the application needs to perform. When one of the computers supporting the web application is available, it will request the next task to be performed.

The tasks need to be prioritized according to the deadline. The next task to be performed is the one with the earliest deadline. Tasks may arrive in any order.

Question 1 [**5 points**]

Identify the **Abstract Data Type** (*not* the data structure) that best aligns with the needs of the task above.

Answer

Variants A, C: Priority Queue **Variants B, D**: Queue

Point Breakdown

- **(5 pt)** A correct answer
- **(2 pt)** Partial credit for stating PriorityQueue for variants B/D

Question 2 [**5 points**]

Identify the specific **Data Structure** that you would use to implement the task above. In no more than 2 sentences, justify your choice.

Answer

Variants A, C: The **Heap** is the most efficient way to store a priority queue. **Variants B, D**: A **LinkedList** or an **ArrayList** (with a ring buffer) can efficiently implement this queue.

Point Breakdown

- **(5 pt)** One of the answers listed above.
- **(4 pt)** Another answer with equivalent runtime complexity (BST for variants A, C)
- **(5 pt)** If Q1 is wrong and this answer is correct for Q1, or if Q2 is wrong and Q1 is correct for Q2.

Flight Departure Table

You are implementing a system for the 'big board' of flight departures for the coming day at a major airport. Your data structure must be able to efficiently (i) List all flights for a given range of times, and (ii) Move a flight from one departure time to a different departure time.

The airport operates using a fixed number of departure 'slots'. (e.g., one slot for a 10:00 AM departure, one for 10:05 AM, one for 10:10 AM, etc...). A departing flight may only be moved to an open slot.

Question 3 [**5 points**]

Identify the **Abstract Data Type** (*not* the data structure) that best aligns with the needs of the task above.

Answer

Variants A, C: List **Variants B, D**: Binary Search Tree

Point Breakdown

- **(5 pt)** A correct answer; Accept 'List' for variants B, D.
- **(4 pt)** A mostly correct answer (e.g., 'Tree' instead)

Question 4 [**5 points**]

Identify the specific **Data Structure** that you would use to implement the task above. In no more than 2 sentences, justify your choice.

Answer

Variants A, C: ArrayList **Variants B, D**: AVL or Red-Black Tree

Point Breakdown

- **(5 pt)** A correct answer
- **(5 pt)** If Q1 is wrong and this answer is correct for Q1, or if Q2 is wrong and Q1 is correct for Q2.