Part A: Bounds

For each question in this section, give the unqualified big-O, big-Ω, and big-Θ bounds for the specified function. If the big-Θ bound does not exist, write **DNE**. For this section you are not required to show any work or give a proof.

Question 1 [**3 points**]

$$f_1(n) = \sum_{i=1}^{n} (17n^2 + i)$$

Answer

Variant A: n^3 for all
Variant B: n^2 for all
Variant C: n^6 for all
Variant D: n^4 for all

Point Breakdown

- **(+1 pt)** For correct O
- **(+1 pt)** For correct Omega
- **(+1 pt)** For a consistent Theta

Question 2 [**3 points**]

$$f_2(n) = \sum_{i=1}^{n} \sum_{j=0}^{\log(n)} 2^j$$

Answer

Variant A: n^2 for all
Variant B: n^3 for all
Variant C: n^4 for all
Variant D: n^5 for all

Point Breakdown

- **(+1 pt)** For correct O
- **(+1 pt)** For correct Omega
- **(+1 pt)** For a consistent Theta

Question 3 [**3 points**]

$$f_3(n) = \begin{cases} \log(n) + 16n + 3 & \text{if n is odd} \\ 3n^2 + 17n & \text{if n is even} \end{cases}$$

Answer

Variant A: $O(n^2)$, $\Omega(n)$, Θ DNE
Variant B: $O(n^7)$, $\Omega(n^3)$, Θ DNE
Variant C: $O(n^4)$, $\Omega(n \log(n))$, Θ DNE
Variant D: $O(n^5)$, $\Omega(\log(n))$, Θ DNE

Point Breakdown

- **(+1 pt)** For correct O
- **(+1 pt)** For correct Omega
- **(+1 pt)** For a consistent Theta

Question 4 [**6 points**]

Prove that the following function is in $O(n^3)$:

$$f_4(n) = 16n^2 + 8n^3 + 7$$

Answer

Break into terms
$16n^2 \leq c \cdot n^3$ is true when $c = 16$, $n \geq 0$
$8n^3 \leq c \cdot n^3$ is true when $c = 8$, $n \geq 0$
$7 \leq c \cdot n^3$ is true when $c = 7$, $n \geq 1$
Therefore if $c = 31$ and $n \geq 1$, $16n^2 + 8n^3 + 7 \leq c \cdot n^3$ which means $f_4 \in O(n^3)$

Point Breakdown

- **(+1 pt)** For including each term in the proof in some way
- **(+1 pt)** For a valid choice for c
- **(+1 pt)** For a valid choice for n_0
- **(+1 pt)** For at some point having the definition of big-O

Part B: PA Review

Question 1 [**4 points**]

In the `SortedList` from PA1, duplicate values were stored in a single linked list node (by giving each linked list node a `count` field). Now imagine inserting n elements into the `SortedList`. In at most 2-3 sentences, describe a characteristic of the data that would result in these insertions having **asymptotically better** performance than if the `SortedList` stored every element in its own linked list node.

Answer

If the set of n elements only has a constant number of unique values, then the insertions when duplicat values are in a single node will always have a cost of $O(1)$. When each element has it's own ll node, then insert will have a cost of $O(1)$.

Point Breakdown

- **(4 pt)** If they somehow point out the characteristic of a constant number of unqiue values (or $\log(n)$ unique values).
- **(2 pt)** If they mention that there should be duplicate values but don't give enough detail
- **(2 pt)** If they mention that the data is already in sorted or reverse sorted order

Question 2 [**4 points**]

In PA2 you implemented two graph search algorithms, BFS and Djikstra's. In at most 2-3 sentences, describe **why** you had to use a `PriorityQueue` in your implementation of Djikstra's instead of a `Queue`.

Answer

A queue only orders element based on when they were inserted. However to find the actual shortest path when edges have weights, we need to explore in an ordering that is based on distance from the starting point. PriorityQueues allow us to dequeue by a priority, which in Djikstra's is distance from the start.

Point Breakdown

- **(4 pt)** An explanation that effectively points out that we need to explore in an ordering based on a characteristic of the data, rather than on the order in which they were found.

Question 3 [**4 points**]

In PA3 you implemented a Hash Table in which you used Cuckoo Hashing to resolve collisions. Name **one benefit** and **one drawback** of Cuckoo Hashing compared to the other collision resolution techniques we covered in class.

Answer

Benefit: find and remove are guaranteed O(1)
Drawback: We may have to rehash more often

Point Breakdown

- **(+2 pt)** For a benefit that mentions guaranteed O(1) runtime for find and/or remove
- **(+2 pt)** For a drawback that mentions more rehashing/infinite eviction loops

Part c: Hash Tables

For all questions in this section, you may assume that items A through H have the following hash codes:

hash(A) = 104	hash(E) = 47
hash(B) = 34	hash(F) = 29
hash(C) = 94	hash(G) = 99
hash(D) = 36	hash(H) = 71

Question 1 [**10 points**]

Consider a Hash Table that has 10 buckets, a maximum load factor of 1, and resolves collisions using chaining. During a rehash, the table doubles its number of buckets. Draw the final contents of the Hash Table after inserting the elements A through H in order.

Answer

The hash table should have ten buckets (0-9) with the following elements in each:
Variant A: H in bucket 1 — A,B,C in bucket 4 — D in bucket 6 — E in bucket 7 — F,G in bucket 9
Variant B: H in bucket 3 — D in bucket 4 — A,B,C in bucket 5 — F,G in bucket 7 — E in bucket 9
Variant C: H in bucket 2 — E in bucket 3 — A,B,C in bucket 6 — F,G in bucket 8 — D in bucket 9
Variant D: A,B,C in bucket 1 — H in bucket 2 — D in bucket 5 — F,G in bucket 6 — E in bucket 8

Point Breakdown

- **(+2 pt)** For having the correct number of buckets (10)
- **(+1 pt)** For each element in the correct bucket (based on their number of buckets)

Question 2 [**1 points**]

Now imagine that the Hash Table from the previous question has a maximum load factor of 0.3 instead. Which element would cause a rehash to occur when you try to insert it?

Answer

Variant A: D (and/or G)
Variant B: E
Variant C: F
Variant D: G

Point Breakdown

- **(1 pt)** For the correct answer

Question 3 [**10 points**]

Draw the final contents of the Hash Table from Question 2 after inserting elements A through H in order.

Answer

Variant A should have 40 buckets. Variants B, C, D should have 20 buckets. Elements should be in the following buckets:
Variant A: A: 24 — B: 34 — C: 14 — D: 36 — E: 7 — F: 29 — G: 19 — H: 31
Variant B: A: 5 — B: 15 — C: 15 — D: 14 — E: 9 — F: 7 — G: 17 — H: 13
Variant C: A: 6 — B: 16 — C: 16 — D: 19 — E: 3 — F: 8 — G: 18 — H: 12
Variant D: A: 1 — B: 11 — C: 11 — D: 15 — E: 8 — F: 6 — G: 16 — H: 12

Point Breakdown

- **(+2 pt)** For having the correct number of buckets
- **(+1 pt)** For each element in the correct bucket (based on their number of buckets)

Question 4 [**6 points**]

What are the expected and unqualified worst case runtimes of each of the following operations for the Hash Table above?

`insert(T elem)`	`contains(T elem)`	`remove(T elem)`
Expected: $O(1)$	Expected: $O(1)$	Expected: $O(1)$
Unqualified: $O(n)$	Unqualified: $O(n)$	Unqualified: $O(n)$

Answer

Point Breakdown

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

Question 5 [**6 points**]

Now imagine that the above Hash Table is implemented in such a way that each bucket is a Balanced BST. What are the expected and unqualified worst case runtimes of each of the following operations in this scenario?

`insert(T elem)`	`contains(T elem)`	`remove(T elem)`
Expected: $O(1)$	Expected: $O(1)$	Expected: $O(1)$
Unqualified: $O(n)$	Unqualified: $O(\log(n))$	Unqualified: $O(\log(n))$

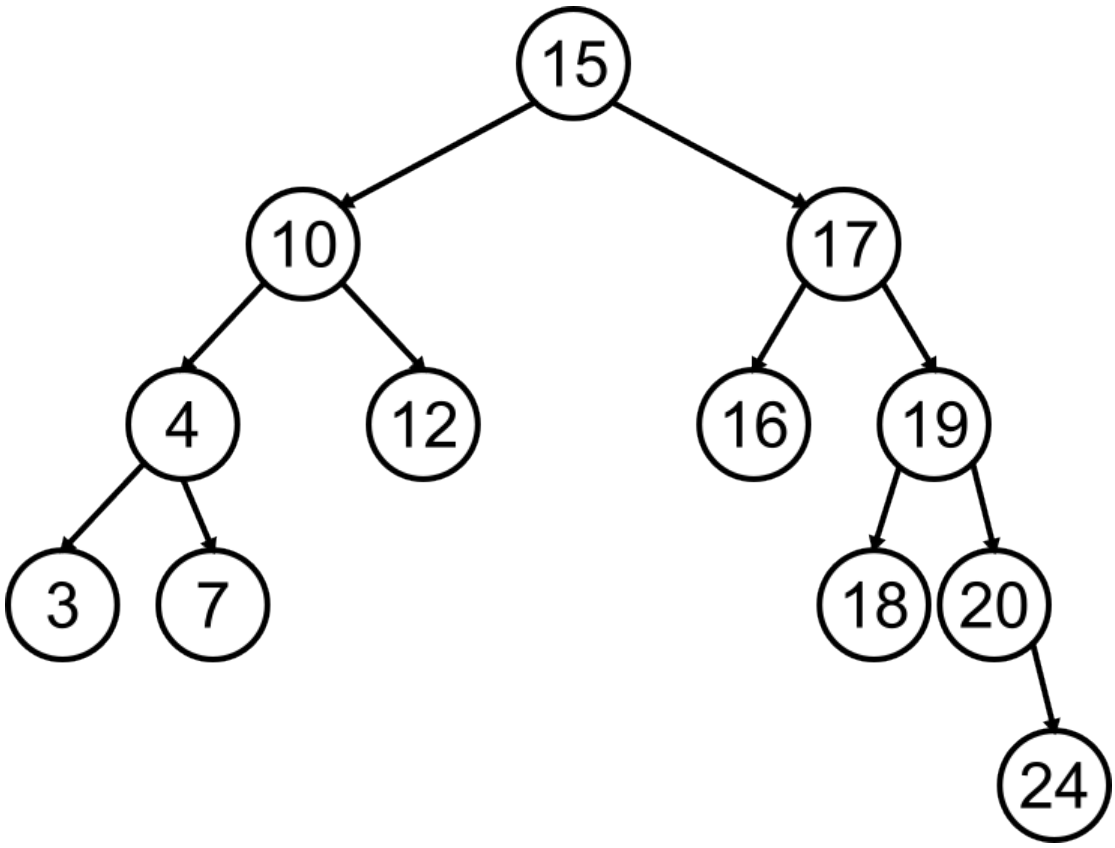
Answer

Point Breakdown

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

Part d: Trees

Answer the following questions about this tree:



ANSWERS WITHOUT EXPLANATION WILL RECEIVE 0 CREDIT

Question 1 [**5 points**]

Is the above tree a Heap? State why or why not. If it is not, be specific (ie name a specific node that causes and issue and why).

Answer

No. It is not a heap. Level 4 is not full, which makes it incomplete. Or name any one of the nodes that is not smaller than it's children.

Point Breakdown

- **(+5 pt)** Correct answer and good explanation
- **(+3 pt)** Correct answer with a vague explanation (ie tree is not complete)

Question 2 [**5 points**]

Is the above tree a BST? State why or why not. If it is not, be specific (ie name a specific node that causes and issue and why).

Answer

Yes. For EVERY node in the tree, the elements in it's left subtree are smaller, and the elements in it's right subtree are larger.

Point Breakdown

- **(+5 pt)** Correct answer and complete explanation
- **(+3 pt)** Correct answer and incomplete explanation (ie one that states elements smaller than the root go left, and larger than the root go right).

Question 3 [**5 points**]

Is the above tree a valid AVL tree? State why or why not. If it is not, be specific (ie name a specific node that causes and issue and why).

Answer

No. One of the nodes in the tree has a balance factor of 2. The inbalanced node is:
Variant A: 17 Variant B: 56 Variant C: 43 Variant D: 27

Point Breakdown

- **(+5 pt)** Correct answer with complete explanation
- **(+3 pt)** Correct answer with incomplete explanation (ie state a balance factor is off, but don't say which one, or the wrong one).

Question 4 [**5 points**]

Is the above tree a valid Red-Black tree? State why or why not. If it is not, be specific (ie name a specific node that causes and issue and why).

Answer

Yes. For EVERY subtree the depth of the shallowest empty tree node is at least the depth of the deepest empty tree node / 2. Alternatively, can give a valid coloring:
Variant A: Black - 15,10,17,12,16,18,20,3,7. Red - 4, 19, 24
Variant B: Black - 27,21,56,26,55,71,73,3,7. Red - 4, 72, 74
Variant C: Black - 27,23,43,26,40,60,73,3,21. Red - 4, 61, 74
Variant D: Black - 15,12,27,14,20,60,73,3,9. Red - 4, 61, 74

Point Breakdown

- **(+5 pt)** For a correct answer and complete explanation.
- **(+3 pt)** For a correct answer and incomplete explanation (ie says it can be colored, but don't give a coloring. Or refer to depths of leaves instead of empty tree nodes, or forget the EVERY part.)

Part e: ADT Implementation

The following two questions require you to describe an implementation of methods in the `PriorityQueue` ADT using an `ArrayList` as the underlying data structure.

- Make sure your methods are consistent with one another (ie they should both considered to be part of the same implementation of `PriorityQueue`)
- Your implementations may call methods of the `ArrayList` class that we have described in class without requiring detailed explanation of how those methods work
- All other operations should be described in detail in either pseudo-code or described in english

You must also give the worst-case runtime bound for your implementation of each method.

- Give the tightest upper bound you can
- Give the amortized runtime if it is smaller than the unqualified runtime

Question 1 [**10 points**]

Algorithm for `void PriorityQueue<T>.add(T elem)`:

Answer

There are 3 acceptable options for this implementation: a lazy approach, a proactive approach, or a heap based approach. Their add method will get points for implementing one of these. The remove will only get points if their chosen method works in conjunction with their add.
Lazy Approach: Just append to the ArrayList. amortized O(1).
Proactive Approach: Iterate through the ArrayList to find the insertion point in order to keep the ArrayList in sorted order, then insert in that spot. O(n).
Heap Approach: Append to the end of the ArrayList, then call fixUp on that element. fixUp will check to see if it must swap with it's parent. If it does, then call fixUp again on the new element. Repeat until the element ends up in the right spot. amortized log(n)

Point Breakdown

- **(+8 pt)** For a good explanation of one of the above approaches.
- **(+4 pt)** For an explanation that is missing some detail (ie don't state how to insert in sorted order, or explain in enough detail how to insert into the correct spot of the heap)
- **(+2 pt)** For the correct runtime
- **(+1 pt)** For the correct unqualified runtime but not mentioning the better amortized runtime.

Question 2 [**10 points**]

Algorithm for `T PriorityQueue<T>.remove()`:

Answer

Remember: They only get points if their approach aligns with their add approach.
Lazy Approach: Iterate through the ArrayList to find the highest priority element, then remove it. O(n)
Proactive Approach: Remove the last element, which will be the highest priority element. O(1)
Heap Approach: Swap the first element with the last element. Remove the last element. Then fixDown on the first elements, which checks to see if it is a lower priority than it's children. If yes, it swaps it with the highest priority child. Continue fixing down until you can't anymore. O(log(n))

Point Breakdown

- **(+8 pt)** For a good explanation of one of the above approachs
- **(+4 pt)** For an explanation that is missing some detail (ie for the proactive approach removing from the wrong end)
- **(+2 pt)** For the runtime of THEIR algorithm

Part f: Data Structure Design

Question 1 [**10 points**]

Imagine a live auction service that allows users to bid on items in real time. Every user has a unique integer ID. You want to store information about users **currently** participating in auctions on the server in order to satisfy the following criteria:

1. Users may log in and log off at any time
2. We need to be able to look up a user by their ID quickly
3. We **cannot** tolerate any expensive spikes in runtime, even if they only happen occasionally

Give a data structure discussed in class that could be used to best solve this problem. Explain why your solution is the best choice. Make sure to address each of the above criteria in your explanation.

Answer

A balanced BST meets all 3 criteria the best. We can add and remove in O(log n) time, we can lookup in O(log n) time. Since the BST never needs to resize, shift elements, and does not rely on expected runtime, it will also not have expensive spikes in runtime – the operations will ALWAYS be at most O(log n).

Point Breakdown

- **(+4 pt)** For balanced BST (or B+ tree would be acceptable)
- **(+2 pt)** For a Hash Table, or a Sorted ArrayList (or another DS that covers 2/3 of the criteria)
- **(+2 pt)** For each criteria that is accurately mentioned in their explanation.

Question 2 [**5 points**]

Would your choice of data structure in the previous problem change if we added the following additional criteria: The number of users currently participating in auctions is expected to be so large that the data will not fit in memory.

If your answer is yes, state what data structure you would use instead and why. If your answer is no, explain why the data structure you used in the previous question also addresses this new criteria.

Answer

Yes. A B+ Tree would have similar asymptotic performance to a balanced BST, but would also minimize the amount of times we have to read from disk.

Point Breakdown

- **(+5 pt)** For a correct answer with an explanation that talks about reducing the number of times we go to disk. Note: If they said B+ Tree for Q1, then an answer of No that explains why would also get full credit.
- **(+2 pt)** For B+ Tree without a good explanation.

Question 3 [**5 points**]

You watch A LOT of movies, and have decided to write a small application that helps you review the movies you watch. At its core, the application must store a TODO list of movies you've watched that still need to be reviewed. Every time you watch a new movie, you add it to the TODO list and every time you review a movie you remove it from the TODO list. In order to give the best reviews you can, you want to review the movies that you've seen **most recently**.

What ADT that we've discussed in class matches the desired behavior of the TODO list? In at most 2-3 sentences explain why.

Answer

A Stack. A stack enforces LIFO order. So every movie watched gets pushed to the top, and then when it is time to review a movie, we pop from the top, which will give us the most recently seen unreviewed movie.

Point Breakdown

- **(+2 pt)** For Stack (or SortedList would also be acceptable).
- **(+3 pt)** For a reasonable explanation.

Part G: Class Participation [Bonus]

Question 1 [**5 points**]

Name any two of the undergraduate TAs for this course (first name only is fine).

Answer

Riad, Amelia, Kartike, DK, Kiki, Marvin, Derek, Brendan, Doniyor, Ethan, Evan, Joy, Marian, Jordan, Chris, Ronan, Alex, Shreyas, Wonwoo, Morgan, Jonathan, Milos, Eric

Point Breakdown

- **(+2.5 pt)** Per correct answer (only count the first two names they wrote if they write a bunch)