Part a: Code Analysis

For questions in this part, consider the following code:

```
class Mystery<T> {
  private Stack<T> in = new Stack<>();
  private Stack<T> out = new Stack<>();

  public void add(T elem) {
    in.push(elem);
  }

  public T remove() {
    if (out.size() == 0) {
      while (in.size() > 0) {
        out.push(in.pop());
      }
    }
    return out.pop();
  }
}
```

Question 1 [**4 points**]

What are the unqualified and amortized runtime bounds of the `add` function when a `LinkedList` based implementation of a `Stack` is used? For the amortized bound, use the most specific bound you can (a Θ bound if it exists, otherwise the tight O bound).

Answer

$O(1)$, $\Omega(1)$, $\Theta(1)$, $Amortized\Theta(1)$

Point Breakdown

- **(+1 each pt)** Correct big-O and big-Ω
- **(+1 pt)** For consistent big-Θ based on their answers for big-O and big-$Omega$
- **(+1 pt)** For correct amortized (Both O and Theta is acceptable)

Question 2 [**4 points**]

What are the unqualified and amortized runtime bounds of the `remove` function when a `LinkedList` based implementation of a `Stack` is used? For the amortized bound, use the most specific bound you can (a Θ bound if it exists, otherwise the tight O bound).

Answer

$O(n)$, $\Omega(1)$, Θ DNE, $Amortized\Theta(1)$

Point Breakdown

- **(+1 each pt)** Correct big-O and big-Ω
- **(+1 pt)** For consistent big-Θ based on their answers for big-O and big-$Omega$
- **(+1 pt)** For correct amortized (Both O and Theta is acceptable)

Question 3 [**4 points**]

What are the unqualified and amortized runtime bounds of the `add` function when an `ArrayList` based implementation of a `Stack` is used? For the amortized bound, use the most specific bound you can (a Θ bound if it exists, otherwise the tight O bound).

Answer

$O(n)$, $\Omega(1)$, ΘDNE, $Amortized \Theta(1)$

Point Breakdown

- **(+1 each pt)** Correct big-O and big-Ω
- **(+1 pt)** For consistent big-Θ based on their answers for big-O and big-$Omega$
- **(+1 pt)** For correct amortized (Both O and Theta is acceptable)

Question 4 [**4 points**]

What are the unqualified and amortized runtime bounds of the `remove` function when an `ArrayList` based implementation of a `Stack` is used? For the amortized bound, use the most specific bound you can (a Θ bound if it exists, otherwise the tight O bound).

Answer

$O(n)$, $\Omega(1)$, Θ DNE, $Amortized \Theta(1)$

Point Breakdown

- **(+1 each pt)** Correct big-O and big-Ω
- **(+1 pt)** For consistent big-Θ based on their answers for big-O and big-$Omega$
- **(+1 pt)** For correct amortized (Both O and Theta is acceptable)

Question 5 [**4 points**]

Does the `Mystery` class above meet the specifications of a `Stack`, a `Queue`, or neither? In at most two sentences, explain your answer.

Answer

`Queue`. When elements are removed, they are removed in the order that they were originally inserted (FIFO ordering).

Point Breakdown

- **(+2 pt)** For recognizing that `Mystery` is an implementation of a `Queue`.
- **(+2 pt)** For reasonable explanation that somehow calls out FIFO ordering, even if not by name directly.

Part B: Asymptotic Analysis

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 [**5 points**]

$$f_1(n) = 10n + n\log(2^n) + 3\log(n)$$

Answer

A: n^2 for all
B: $n\log(n)$ for all
C: n^3 for all
D: $n\log(n)$ for all

Point Breakdown

- **(+2 each pt)** for correct O and Ω
- **(+1 pt)** for Θ that is consistent with O and Ω

Question 2 [**5 points**]

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

Answer

A: n^5 for all
B: n^4 for all
C: n^7 for all
D: n^5 for all

Point Breakdown

- **(+2 each pt)** for correct O and Ω
- **(+1 pt)** for Θ that is consistent with O and Ω

Question 3 [**5 points**]

$$f_3(n) = \sum_{i=1}^{10} i^2$$

Answer

1 for all

Point Breakdown

- **(+2 each pt)** for correct O and Ω
- **(+1 pt)** for Θ that is consistent with O and Ω

Question 4 [**5 points**]

$$f_4(n) = \begin{cases} 2n & \text{if n is prime} \\ 3n^2 & \text{if n is greater than 2 and even} \\ 14\log(n) & \text{otherwise} \end{cases}$$

Answer

A: $O(n^2)$, $\Omega(\log(n))$, Θ DNE
B: $O(n^3)$, $\Omega(n\log(n))$, Θ DNE
C: $O(n)$, $\Omega(1)$, Θ DNE
D: $O(n^4)$, $\Omega(n)$, Θ DNE

Point Breakdown

- **(+2 each pt)** for correct O and Ω
- **(+1 pt)** for Θ that is consistent with O and Ω

Question 5 [**5 points**]

$$f_5(n) = \begin{cases} 2n^2 + n & \text{if n is even} \\ 3n^2 & \text{if n is odd} \end{cases}$$

Answer

A: n^2 for all B: n for all C: n^3 for all D: $n\log(n)$ for all

Point Breakdown

- **(+2 each pt)** for correct O and Ω
- **(+1 pt)** for Θ that is consistent with O and Ω

Part c: Bounds Proofs

For each question in this part, you must prove the bound in question by coming up with constants c and n_0 that satisfy the inequalities as defined in class. You must show all work. **Answers given without showing sufficient work will receive no credit.**

Question 1 [**10 points**]

Let $g_1(n) = 3n^2 + 10n + 4$. Prove $g_1(n) \in O(n^2)$.

Answer

Proof for Variant A: Break into pieces:

$$3n^2 \leq c_1 n^2$$

The above is true when $c\_1 = 3$ (for example)

$$10n \leq c_2 n^2$$

The above is true when $c_2 = 10$ and $n \geq 1$ (for example)

$$4 \leq c_3 n^2$$

The above is true when $c_3 = 4$ and $n \geq 1$ (for example)
Therefore by composition, our original inequality holds true for all $n \geq 1$ when $c = 3 + 10 + 4 = 17$.
Same structure applies for variant B (expected answer for c would be 25 for all $n \geq 1$)
Same structure applies for variant C, with a log rule being applied to simplify the last term (expected answer for c would be 17 for $n \geq 1$)
Same structure applies for variant D, with a log rule being applied to simplify the last term (expected answer for c would be 25 for $n \geq 1$)

Point Breakdown

- **(+1 pt)** for a valid c, n_0 as long as there's an attempt to show work
- **(+9 pt)** per detailed work, broken up over each term

Question 2 [**10 points**]

Let $g_2(n) = 10n^2 + n^2 \log(2^n)$. Prove $g_2(n) \in \Omega(n^3)$.

Answer

Proof for variant A: Break into pieces:

$$10n^2 \geq 0$$

The above is trivially true for $n \geq 0$

$$n^2 \log(2^n) \geq cn^3$$

By applying log rules:

$$n^3 \geq cn^3$$

The above is true when $c = 1$ for all $n \geq 0$ (for example)
Therefore by composition, our original inequality is true $c = 1$ for all n.
Same structure applies for variant B, expected value of c is 3.
Same structure applies for variant C but no need for log rules. Expected value of c is 4.
Same structure applies for variant D but no need for log rules. Expected value of c is 3.

Point Breakdown

- **(+1 pt)** for a valid c, n_0 as long as there's an attempt to show work
- **(+9 pt)** per detailed work, broken up over each term

Question 3 [**10 points**]

For this question you may refer to work done in either of the previous two questions if needed.
Let $g_3(n) = 10n^2 + n^2 \log(2^n)$. Prove $g_3(n) \in \Theta(n^3)$.

Answer

For variant A: Need to show $g_3 \in O(n^3)$ AND $g_3 \in \Omega(n^3)$
We already showed the big-Omega bound in the previous question.
For big-O, break into pieces:

$$10n^2 \leq c_1 n^3$$

The above holds true when $c_1 = 10$ for $n \geq 1$

$$n^2 \log(2^n) \leq c_2 n^3$$

Apply log rules:

$$n^3 \leq c_2 n^3$$

The above is true when $c_2 = 1$
By composition, our original inequality for big-O is true when $c = 10 + 1 = 11$ for $n \geq 1$.
Therefore $g_3 \in O(n^3)$ AND $g_3 \in \Omega(n^3)$, so it is also in $\Theta(n^3$
Same structure applies to B, expected value for c is 7.
Same structure applies to C, except the big O will have already been proven in q1 of this part so here they'll prove Omega. Expected value for c is 4.
Same structure applies to D, except the big O will have already been proven in q1 of this part so here they'll prove Omega. Expected value for c is 18.

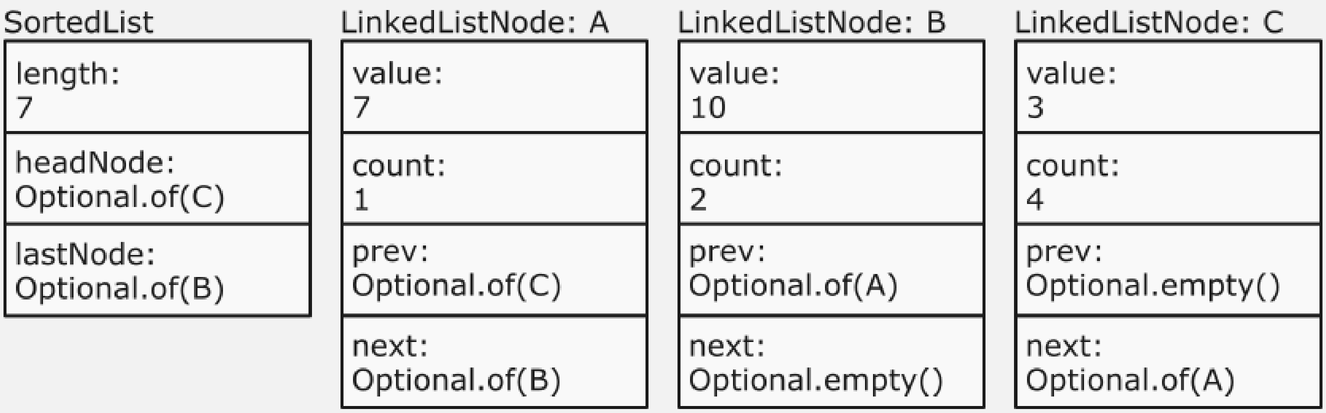
Point Breakdown

- **(+5 pt)** for big O proof (using similar point breakdown as q1, or if relevent skipping the proof because they did it in q1)
- **(+5 pt)** for big Omega proof (using similar point breakdown as q2, or if relevent skipping the proof because they did it in q2)

Part D: PA1 Review

Each of the following questions depicts a single `SortedList` comprised of three `LinkedListNode` objects based on the specs of PA1. Each diagram contains up to two bugs. For each question state what the bugs are, or if there are no bugs, state that the diagram represents a valid example of a `SortedList` from PA1.

Question 1 [**5 points**]



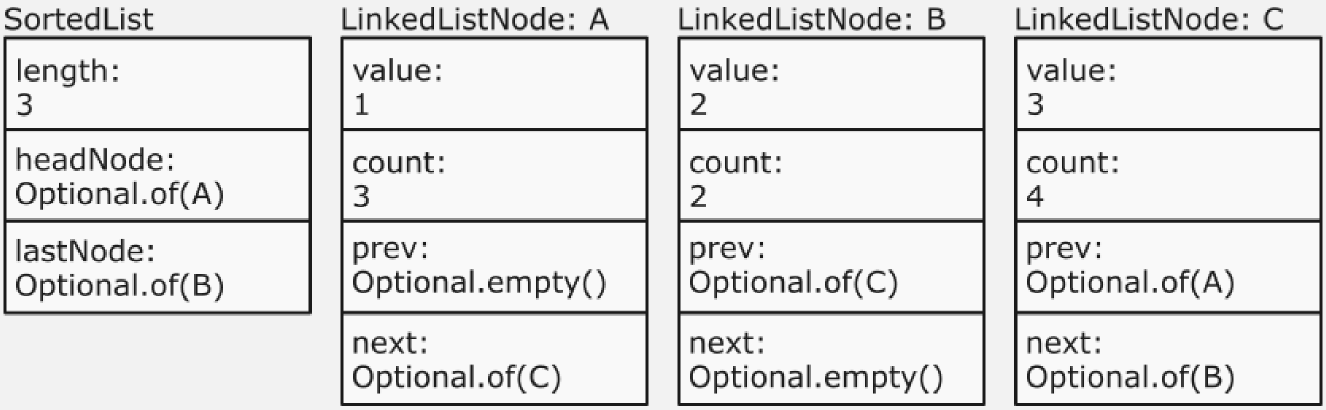
Answer

Variant A: No bugs, valid list Variant B: Bug 1 - list is not sorted/out of order/incorrect tail, Bug 2 - incorrect length/incorrect counts/counts and length don't match Variant C: No bugs, valid list Variant D: Bug 1 - list is not sorted/out of order/incorrect tail, Bug 2 - incorrect length/incorrect counts/counts and length don't match

Point Breakdown

- **(5 pt)** For recognizing a valid list or if both bugs are identified
- **(3 pt)** If only one of two bugs is found

Question 2 [**5 points**]



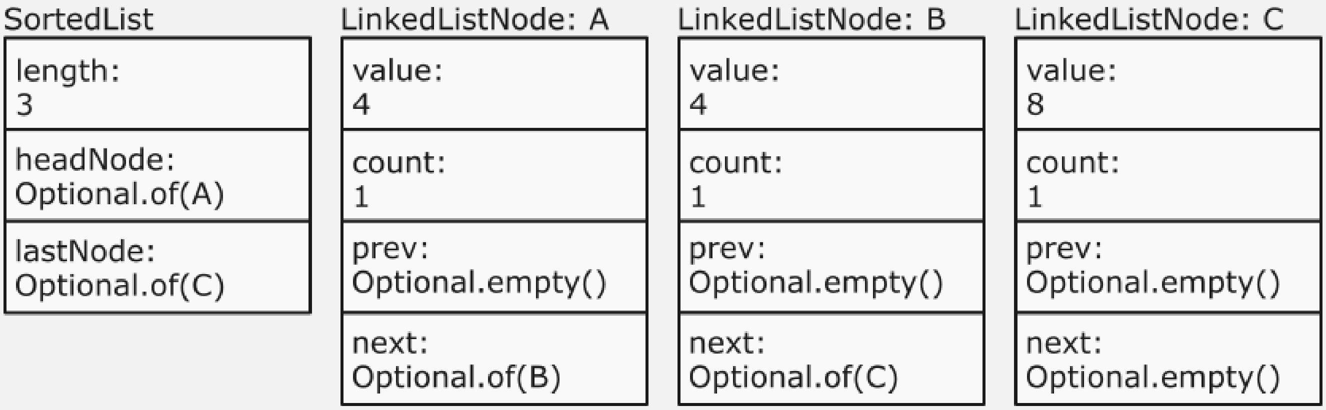
Answer

Variant A: Bug 1 - list is not sorted/out of order/incorrect tail, Bug 2 - incorrect length/incorrect counts/-counts and length don't match Variant B: Bug 1 - duplicate values are supposed to be in the same node, Bug 2 - incorrect prev links Variant C: Bug 1 - list is not sorted/out of order/incorrect tail, Bug 2 - incorrect length/incorrect counts/counts and length don't match Variant D: Bug 1 - duplicate values are supposed to be in the same node, Bug 2 - incorrect prev links

Point Breakdown

- **(5 pt)** For recognizing a valid list or if both bugs are identified
- **(3 pt)** If only one of two bugs is found

Question 3 [**5 points**]



Answer

Variant A: Bug 1 - duplicate values are supposed to be in the same node, Bug 2 - incorrect prev links Variant B: No bugs, valid list Variant C: Bug 1 - duplicate values are supposed to be in the same node, Bug 2 - incorrect prev links Variant D: No bugs, valid list

Point Breakdown

- **(5 pt)** For recognizing a valid list or if both bugs are identified
- **(3 pt)** If only one of two bugs is found

Part e: Data Structure Design

Question 1 [**10 points**]

You are tasked with designing a data structure which will manage players playing an online game. Each player is represented by a `Player` object, and you must store these `Player` objects in a data structure. The game allows for drop-in/drop-out play meaning that players can join or leave a game at any time. The data structure you design must efficiently allow for this without any unexpected spikes in wait time when a player joins or leaves the game. In the space below you must answer the following questions:

1. What data structure will you use as the basis of your design?
2. What key operations must be implemented efficiently according to the above description?
3. What data, if any, must be stored in the `Player` object to implement those operations efficiently?
4. How you will implement each of the key operations you identified?
5. What is the runtime of each of the key operations you identified?

Answer

We should use a LinkedList. We must efficiently implement add and remove. Add can just add to the end, remove must be able to remove any element. The Player objects should store a reference to the LinkedList node they are stored in (so we can remove by reference, instead of searching the list). For add, we can just append to the end (or beginning of the linked list). For remove, we have the reference to the linked list node in the Player object so we just need to remove it from the linked list by updating the pointers of the neighboring nodes. Both operations are $\Theta(1)$ in this case since they are essentially add and remove by reference.

Point Breakdown

- **(+2 pt)** For choosing LinkedList, +1 for ArrayList
- **(+1 pt)** Each for recognizing add and remove
- **(+2 pt)** For recognizing that we need to store a reference to more efficiently remove
- **(+1 pt)** Each for giving a resonable implementation (should be add and remove by reference, but if they failed to store a reference in the Player object, then a linear search for removal is fine)
- **(+1 pt)** Each for each correct runtime based on THEIR implementation

Part f: Bonus

Question 1 [**5 points**]

Consider a new implementation of the `List` ADT that stores data in an `ArrayList` of `LinkedList`s. Each element of the `ArrayList` is a `LinkedList` that holds the actual elements stored in the list. Each `LinkedList` is constrained to hold at most a fixed number of elements, L, and only the last `LinkedList` in the array is allowed to have fewer than L elements. What is the unqualified worst-case runtime of the `.get(...)` method for this implementation? In at most two sentences explain why.

Answer

Runtime is constant (O(1), Theta(1), O(L), Theta(L) would all be acceptable ways to state this).
We can look up position in the array in constant time, and then we have to search through the LinkedList, but the LinkedList has a constant number of elements. So no matter how big our list gets, the cost to lookup never changes.

Point Breakdown

- **(+1 pt)** For correct answer
- **(+2 pt)** For pointing out that we can get the array position in constant time
- **(+2 pt)** For also recognizing that searching through the list takes constant time