Part A: Code Analysis

```
class Mystery<T> {
    private ArrayList<T> data = new ArrayList<>();

    public void add(T elem) {
        data.add(0, elem);
    }

    public T remove() {
        return data.remove(0);
    }

    public T peek() {
        return data.get(0);
    }
}
```

For questions in this part, consider the following code:

Question 1 [**5 points**]

What are the tight, unqualified runtime bounds of `add`? If the Big-Θ bound does not exist, write **DNE**.

Answer

$O(n)$, $\Omega(n)$, $\Theta(n)$

Point Breakdown

- **(+2 each pt)** Correct big-O and big-Ω
- **(+1 pt)** For a big-Θ that is consistent with their big-O and big-Ω

Question 2 [**5 points**]

What are the tight, unqualified runtime bounds of `remove`? If the Big-Θ bound does not exist, write **DNE**.

Answer

$O(n)$, $\Omega(n)$, $\Theta(n)$

Point Breakdown

- **(+2 each pt)** Correct big-O and big-Ω
- **(+1 pt)** For a big-Θ that is consistent with their big-O and big-Ω

Question 3 [**5 points**]

What are the tight, unqualified runtime bounds of `peek`? If the Big-Θ bound does not exist, write **DNE**.

Answer

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

Point Breakdown

- **(+2 each pt)** Correct big-O and big-Ω
- **(+1 pt)** For a big-Θ that is consistent with their big-O and big-Ω

Question 4 [**5 points**]

Does `Mystery` exhibit the behavior of a `Stack`, `Queue`, or neither? In at most 2 sentences, explain your answer.

Answer

Stack. The most recent element inserted will be the first element removed (LIFO).

Point Breakdown

- **(+1 pt)** for a correct answer of Stack
- **(+4 pt)** for an explanation that somehow demonstrates understanding of LIFO ordering.

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. To get full credit your bounds should be as simplified as possible.

Question 1 [**5 points**]

$$f_1(n) = 47n^3 + n^2 + n^2 \log(2^{n^2})$$

Answer

A: n^4 for all
B: n^3 for all
C: $n \log(n)$ for all
D: n^2 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**]

$$f_2(n) = \sum_{i=1}^{15} \sum_{j=1}^{10} 2^i$$

Answer

A: 1 for all
B: n^2 for all
C: 2^n for all
D: n^3 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**]

$$f_4(n) = \begin{cases} 5n^3 & \text{if n is prime} \\ 3n & \text{if n is greater than 2 and even} \\ \log(n) + 100n^2 & \text{otherwise} \end{cases}$$

Answer

A: $O(n^3)$, $\Omega(n)$, Θ DNE
B: $O(n\log(n))$, $\Omega(1$, Θ DNE
C: $O(n^4)$, $\Omega(n\log(n))$, Θ DNE
D: $O(n^5)$, $\Omega(n^2)$, Θ DNE

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**]

Is it possible for a function to be in both $\Theta(n^2)$ and $O(n^3)$? In at most two sentences, explain your answer.

Answer

Yes. If the function is in $\Theta(n^2)$ it is in $\emptyset(n^2)$, which is a subset of $O(n^3)$.

Point Breakdown

- **(+1 point pt)** for a correct answer
- **(+4 points pt)** for a reasonable explanation

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 $g_1(n) = 6n\log(2^n) + \log(n) + 7n$. Prove $g_1(n) \in O(n^2)$.

Answer

Proof for Variant A: Break into pieces:

$$6n\log(2^n) \leq c_1 n^2$$

Using log rules simplifies the above to:

$$6n^2 \leq c_1 n^2$$

The above is true when $c\_1 = 6$ (for example)

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

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

$$7n \leq c_3 n^2$$

The above is true when $c_3 = 7$ and $n \geq 1$ (for example)
Therefore by composition, our original inequality holds true for all $n \geq 1$ when $c = 6 + 1 + 7 = 14$.
Same structure applies for variant B (expected answer for c would be 19 for all $n \geq 1$)
Same structure applies for variant C (expected answer for c would be 13 for $n \geq 1$)
Same structure applies for variant D (expected answer for c would be 28 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 $g_2(n) = n^3 + 7n^2$. Prove $g_2(n) \in \Omega(n^3)$.

Answer

Proof for variant A: Break into pieces:

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

The above is trivially true for $c = 1$ and $n \geq 0$

$$7n^2 \geq 0$$

The above is trivially true for $n \geq 0$
Therefore by composition, our original inequality is true $c = 1$ for all $n \geq 0$.
Same structure applies for variant B, expected value of c is 12.
Same structure applies for variant C, expected value of c is 7.
Same structure applies for variant D, expected value of c is 5.

Point Breakdown

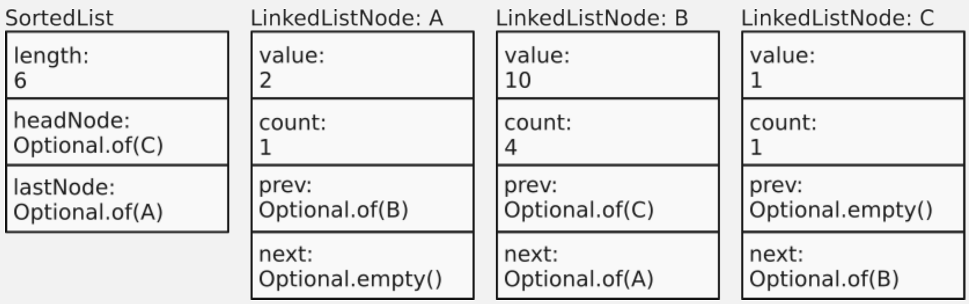
- **(+1 pt)** for a valid c, n_0 as long as there's an attempt to show work
- **(+9 pt)** per detailed work (5 for dominant term, 4 for recognizing other term just needs to be ¿= 0)

Part d: PA1 Review

The following two questions pertain to the `SortedList` data structure you implemented in PA1.

Question 1 [**10 points**]

The diagram below shows the nodes of a nearly valid `SortedList` data structure. There is exactly one error in the structure. Identify the error.



Answer

Variant A: The list (C: 1, B: 10, A: 2) is out of order ($B > A$).
Variant B: The list (C: 1, B: 2, A: 2) contains two entries (B, C) with the same value.
Variant C: The list (C: 1, B: 2, A: 10) has an invalid head pointer (pointing to B instead of C).
Variant D: The list has a length of 6, but a total count of 10.

Point Breakdown

- **(10 pt)** An answer that correctly identifies the problem with the structure.
- **(5 pt)** Partial credit for an answer that demonstrates an understanding of how linked lists work (e.g., by drawing out a diagram).

Question 2 [**10 points**]

Assume that the variable `list` is a `SortedList` containing N integers in the range from 0 to MAX. Suppose the following code has already been run:

```
// Generates a random integer i between 0 and MAX
Random r = new Random()
Integer i = Random.nextInt(MAX)

// Retrieve the node for value i, and save it as a hint.
LinkedListNode<Integer> hint = list.findRefBefore(i)
```

Assuming that `list.length()` is N, give a tight asymptotic upper (Big-O) bound on the runtime of the following block of code:

```
list.insert(i+2, hint)
```

Justify your answer by explaining which `LinkedListNode`s the `insert` operation would need to access in the worst case.

Answer

$O(1)$
Each element in the list has a unique value, and the list contains only integers. Thus, in the worst case, we need to visit the `LinkedListNode` with value i (i.e., `hint`), the node with value $i+1$, and potentially the node with value $i+2$.

Point Breakdown

- **(10 pt)** An answer that correctly identifies the runtime as $O(1)$, and includes a justification that demonstrates an understanding that the number of linked list nodes visited is finite due to the uniqueness constraint.
- **(7 pt)** An answer that demonstrates an understanding that the number of linked list nodes visited is finite due to the uniqueness constraint, but that gives a runtime bound other than $O(1)$.
- **(7 pt)** An answer that correctly indicates the runtime as $O(1)$, but with a justification that solely identifies the hint as a justification (without conveying an understanding that the hint is guaranteed to be finitely many nodes away from the reinserted value).
- **(3 pt)** An answer that correctly identifies the runtime as $O(1)$ but that does not include a meaningful justification.

Part e: Data Structure Design

For each of the following scenarios, noting in particular the bolded text, state the data structure (Array, LinkedList, or ArrayList) you would use. In *at most 2 short sentences*, justify your answer in terms of how the properties of the data structure relate to the (bolded) requirements.

Question 1 [**10 points**]

Smart Watch Faces: You are implementing a 'watch face' manager for a smartwatch, and need a way to store a pointer to the region of memory used to store each watch face's state. Specifically, **you need to store one 8 byte pointer for each watch face**. You need to be able to jump to arbitrary watch faces quickly, so **you need to be able to access the ith pointer in constant time**. Memory on the watch is very limited, so **there will never be more than 19 watch faces open at a time**.

Answer

Variants A, C: Use an array. (i) The size of the array is fixed, and (ii) we need quick access to the ith element.
Variants B, D: Use a linked list. (i) You need quick access to the next/prev elements of the list, and unlike an ArrayList, (ii) allocating new entries is always O(1).

Point Breakdown

- **(10 pt)** An answer that correctly identifies the preferred data structure, and includes a justification that demonstrates understanding of the two features listed above.
- **(8 pt)** An answer correctly identifies both of the two features above as being relevant, but picks the wrong data structure.
- **(8 pt)** An answer that correctly identifies the data structure, but only identifies one of the features above.
- **(5 pt)** An answer correctly identifies one of the two features above as being relevant, but picks the wrong data structure.
- **(3 pt)** An answer that picks the right data structure, but lacks a meaningful justification.

Question 2 [**10 points**]

Intrusion Detection System: You are implementing an intrusion detection system that works in two phases: First, **a large number of event objects are created and need to be stored**. Throughput is important, so it is critical that the **total cost of inserting all of the event objects is linear in the number of objects**. Then, in the second phase, the events are analyzed, requiring **constant-time access to elements by their index.**

Answer

Variants A, B: Use an ArrayList. Since the total cost of inserting all elements needs to be linear, (i) amortized $O(1)$ is sufficient, and (ii) the ArrayList will provide constant-time access to its elements. Also valid solution: Store incoming elements in phase 1 in a linked list, and then copy them to an Array.
Variants C, D: Use a linked list. Amortized $O(1)$ doesn't guarantee constant-time inserts, while (ii) dequeue from the head of a linked list can be done in constant time.

Point Breakdown

- **(10 pt)** An answer that correctly identifies the preferred data structure, and includes a justification that demonstrates understanding of the two features listed above.
- **(8 pt)** An answer correctly identifies both of the two features above as being relevant, but picks the wrong data structure.
- **(5 pt)** An answer correctly identifies one of the two features above as being relevant, but picks the wrong data structure.
- **(3 pt)** An answer that picks the right data structure, but lacks a meaningful justification.

Part f: Bonus

Question 1 [**5 points**]

Suppose you know that the function `foo()` has an **expected runtime** of $O(n)$.
What **guarantees** can you make about the unqualified runtime of the following code:

```
for (int i = 0; i < n; i++) {
  foo();
}
```

Answer

There are no meaningful guarantees that you can make about the *unqualified* runtime based on the information given.
For qualifier X, you can guarantee that the X runtime is n times the X runtime of `foo()`. Given the unqualified runtime of `foo()`, you now have an upper bound on the amortized and expected runtimes of `foo()`. However, this doesn't go in reverse. An expected runtime bound gives you no information about the unqualified runtime.

Point Breakdown

- **(5 pt)** The answer correctly indicates that no unqualified runtime bounds can be inferred from the information given.