Part a: Data Structure Design

Imagine that you're designing the data capture software for a large piece of scientific equipment (like the Large Hadron Collider). Data arrives at the system in a burst of a large number of records (we don't know how many will arrive ahead of time), and needs to be stored immediately (via the `List.add(value)` method). It is critical that each record be stored immediately, any delays on a single call to `List.add` will cause data loss (potentially ruining a few million dollars worth of experimental setup). No other access to the data will be required until after the data collection burst ends.

Question 1 [**5 points**]

Which of the `List` data structures that we have discussed so far in class would you select for this use case and (**in at most one sentence**), why?

Answer

A Linked List.
`ArrayList.add()` is *amortized* $O(1)$, but individual calls could still take $O(N)$.

Point Breakdown

- **(3 pt)** Selecting the Linked List
- **(1 pt)** Identifying the $O(1)$ runtime of `LinkedList`'s `add` method as a reason to use it. **OR** Identifying the $O(N)$ runtime of `ArrayList`'s `add` method as a reason not to use it.
- **(1 pt)** Mentioning the fact that the $O(1)$ runtime of `ArrayList` is *amortized.*

Once the burst is finished, the software has an opportunity to reorganize the data into a new data structure for preliminary analysis. This preliminary analysis will require random access to the data (i.e., repeated calls to the List.get method).

Question 2 [**5 points**]

Which of the List data structures that we have discussed in class would you select for this phase and (**in at most one sentence**), why?

Answer

An Array List
`LinkedList.get()` is $O(N)$.

Point Breakdown

- **(3 pt)** Selecting the Array List
- **(2 pt)** Identifying the $O(1)$ runtime of `ArrayList`'s `get` method as a reason to use it **OR** Identifying the $O(N)$ runtime of `LinkedList`'s `get` method as a reason not to use it

Question 3 [**5 points**]

State the tight worst-case runtime bound of generating the data structure in your answer to Question 2 from the data structure in your answer to part Question 1.

Answer

$O(N)$; Every element in the linked list must be copied to the array.

Point Breakdown

- **(5 pt)** The answer is correct for the pair of data structures given above. In general, this means $O(N)$ if the data structures are different, and $O(1)$ if they are the same.

Part B: Complexity

For each of the following formulas, state the Big-O, Big-Ω, and Big-θ bounds (or indicate that the bound does not exist)

Question 1 [**5 points**]

$$f_1(N) = 5N \cdot 2^{\log(N)} + 63\log(N) + N\log(N)$$

$O(f_1)$:

$\Omega(f_1)$:

$\theta(f_1)$:

Answer

- $O(N^2)$, $\Omega(N^2)$, $\theta(N^2)$ for variants A, B
- $O(N^3)$, $\Omega(N^3)$, $\theta(N^3)$ for variants C, D

Point Breakdown

- **(2 pt)** Big-O bound is correct
- **(2 pt)** Big-Ω bound is correct
- **(1 pt)** Big-θ bound is consistent with Big-O and Big-Ω answers.

Question 2 [**5 points**]

$$f_2(N) = \sum_{i=1}^{N} 2^i + 2i$$

$O(f_2)$:

$\Omega(f_2)$:

$\theta(f_2)$:

Answer

- $O(2^N)$, $\Omega(2^N)$, $\theta(2^N)$ for variants A, C
- $O(N^2)$, $\Omega(N^2)$, $\theta(N^2)$ for variants B, D

Point Breakdown

- **(2 pt)** Big-O bound is correct
- **(2 pt)** Big-Ω bound is correct
- **(1 pt)** Big-θ bound is consistent with Big-O and Big-Ω answers.

Question 3 [**5 points**]

$$f_3(N) = \sum_{i=1}^{N} \sum_{j=1}^{i} 6$$

$O(f_3)$:

$\Omega(f_3)$:

$\theta(f_3)$:

Answer

- $O(N^2)$, $\Omega(N^2)$, $\theta(N^2)$ for all variants

Point Breakdown

- **(2 pt)** Big-O bound is correct
- **(2 pt)** Big-Ω bound is correct
- **(1 pt)** Big-θ bound is consistent with Big-O and Big-Ω answers.

Question 4 [**5 points**]

$$f_4(N) = \begin{cases} N^2 & \textbf{if } N \text{ is even} \\ N & \textbf{if } N \text{ is odd} \end{cases}$$

$O(f_4)$:

$\Omega(f_4)$:

$\theta(f_4)$:

Answer

- $O(N^2)$, $\Omega(N)$, no θ bound for variant A
- $O(N^2)$, $\Omega(1)$, no θ bound for variants B, D
- $O(N)$, $\Omega(1)$, no θ bound for variant C

Point Breakdown

- **(2 pt)** Big-O bound is correct
- **(2 pt)** Big-Ω bound is correct
- **(1 pt)** Big-θ bound is consistent with Big-O and Big-Ω answers.

Part c: Asymptotic Bounds

For each of the following claims, use the inequality definition of Big-O, Big-Ω, or Big-θ to either prove or disprove the claim. For each question, your answer must show, using the inequalities equivalent to the claim, and the rules given in the cheat sheet at the front of the exam, one of the following:

- ... that there exists a constant (write down such a constant) for which the inequality(ies) must hold for a sufficiently large N; **OR**
- ... that the inequality(ies) does not hold for any constant and large values of N (e.g., by reducing the inequality to an invalid inequality $e.g., c > N$).

Question 1 [**5 points**]

$$2^N + 5N + 1 \in \Omega(N^2)$$

Answer

Variants A, B
We need to show

$$2^N + 5N + 1 \overset{?}{>} c \cdot N^2$$

(pick $a + b + d = c > 0$)

$$2^N + 5N + 1 \overset{?}{>} (a + b + d) \cdot N^2$$

It suffices to show:
- $2^N \overset{?}{>} a \cdot N^2$
- $5N \overset{?}{>} b \cdot N^2$
- $1 \overset{?}{>} d \cdot N^2$

Pick $a = 1$

$$2^N > N^2$$

Given (True for $N > 4$); Pick $b = 0$

$$5N > 0 \cdot N^2 = 0$$

Given (True for $N > 1$); Pick $d = 0$

$$1 > 0 \cdot N^2 = 0$$

Given (True); Since $a + b + d > 0$, we have

$$2^N + 5N + 1 > 1 \cdot N^2$$

QED.

Variants C, D
We need to show

$$2^N + 5N + 1 \overset{?}{<} c \cdot N^2$$

We know $2^N < 2^N + 5N + 1$, so if $2^N > c \cdot N^2$, then we have a contradiction.

$$2^N > c \cdot N^2$$

Given. QED.

Point Breakdown

- **(1 pt)** Valid selection of constants; Correct counter-example picked.
- **(4 pt)** Work shown to derive the constants/counter-example.

Question 2 [**5 points**]

$$N^2 + 12N + \log(N) \in O(N)$$

Answer

Variants A, B
We need to show

$$N^2 + 12N + \log(N) \overset{?}{<} c \cdot N$$

We know $N^2 < N^2 + 12N + \log(N)$, so if $N^2 > c \cdot N$, then we have a contradiction.

$$N^2 > c \cdot N$$

Given. QED.
Variants C, D
We need to show

$$N^2 + 12N + \log(N) \overset{?}{>} c \cdot N$$

(pick $a + b + d = c > 0$)

$$N^2 + 12N + \log(N) \overset{?}{>} (a + b + d) \cdot N$$

It suffices to show:

- $N^2 \overset{?}{>} a \cdot N$
- $12N \overset{?}{>} b \cdot N$
- $\log(N) \overset{?}{>} d \cdot N$

Pick $a = 1$

$$N^2 > N$$

Given (True for $N > 1$); Pick $b = 0$

$$12N > 0 \cdot N = 0$$

Given (True for $N > 1$); Pick $d = 0$

$$\log(N) > 0 \cdot N^2 = 0$$

Given (True); Since $a + b + d = 1 > 0$, we have

$$N^2 + 12N + \log(N) > 1 \cdot N^2$$

QED.

Point Breakdown

- **(1 pt)** Valid selection of constants; Correct counter-example picked.
- **(4 pt)** Work shown to derive the constants/counter-example.

Question 3 [**5 points**]

$$\left(\begin{cases} N^2 & \textbf{if } n \text{ is even} \\ N & \textbf{if } n \text{ is odd} \end{cases}\right) \in \theta(N)$$

Answer

To show that the function is not in $\theta(N)$ or $\theta(N^2)$, we need to show that one of the Big-O or Big-Ω bounds.

Variants A, C

We can show that the function is not in $O(N)$

$$\left(\begin{cases} N^2 & \textbf{if } N \text{ is even} \\ N & \textbf{if } N \text{ is odd} \end{cases}\right) \leq c \cdot N$$

$$\left(\begin{cases} N & \textbf{if } N \text{ is even} \\ 1 & \textbf{if } N \text{ is odd} \end{cases}\right) \leq c \cdot 1$$

This is false for any even value of $N > c$.

Variants B, D

We can show that the function is not in $\Omega(N^2)$

$$\left(\begin{cases} N^2 & \textbf{if } N \text{ is even} \\ N & \textbf{if } N \text{ is odd} \end{cases}\right) \geq c \cdot N^2$$

$$\left(\begin{cases} N & \textbf{if } N \text{ is even} \\ 1 & \textbf{if } N \text{ is odd} \end{cases}\right) \geq c \cdot N$$

This is false for any odd value of $N > c$.

Point Breakdown

- **(1 pt)** Valid selection of constants; Correct counter-example picked.
- **(4 pt)** Work shown to derive the constants/counter-example.

Part D: Project Review

Recall the `SortedList` structure that you completed for **PA1**. Consider the following new data structure, consisting of

- A `SortedList` (over strings that store names)
- A 26-element array named `hints`.

`hints[0]` contains the first node of the sorted list that starts with an 'A' or `Option.empty()` if no nodes start with 'A'. `hints[1]` does the same for nodes starting with 'B', `hints[2]` for 'C', etc...

Whenever we want to search for or insert a new name into the list, we first check `hints` to see if a node starting with the same letter as our target exists. If it does, we use that element as the hint to our search/insert. Otherwise, we use the closest non-empty element of the `hints` array, or the unhinted version if it is empty. If the operation was an insert, we also update `hints` as needed.

Question 1 [**5 points**]

What is the asymptotic complexity of updating and retrieving `hints` from the `hints` array?

Answer

$O(1)$

Point Breakdown

- **(5 pt)** The answer is $O(1)$

Question 2 [**5 points**]

What is the unqualified worst-case complexity of finding an element in the list when using a hint retrieved from `hints`. You may assume that the hint in `hints` was not `Option.empty()` and that the hinted version of the search is used.

Answer

$O(\frac{N}{26}) = O(N)$

Point Breakdown

- **(5 pt)** The answer is $O(N)$

Part e: Data Structure Performance

Think about a scrolling list on your phone (e.g., Instas, Emails, Piazza Posts). This list is populated 'lazily'; that is, as you scroll down, the system automatically fetches the next batch of entries in the list (e.g., posts) and appends them to one of the data structures we've discussed in class. Once it loads an element, it is never forgotten: the app keeps all of the entries around forever (or at least until you close the app).

Users complain about the app, that scrolling down is usually smooth, but occasionally the interface freezes for about a second.

Question 1 [**5 points**]

Name one of the data structures that we've discussed in class that the app is probably using to store the list entries. Explain **in at most two sentences**, why the app is probably lagging.

Answer

The symptoms suggest that `List.add(item)` is occasionally very slow. Although `ArrayList.add(item)` is *amortized* $O(1)$, individual calls to the method may still be $O(N)$.

Point Breakdown

- **(2 pt)** The answer correctly identifies the ArrayList.
- **(3 pt)** The answer correctly identifies the *amortized* runtime of `add(item)` as the problem.

Part f: Algorithm Runtimes

In Java, `addAll` is a method that combines two lists by adding the elements of one of the lists to the end of the other list. For example, if `l1` is a list containing the elements `[1,4,7]`, and `l2` is a list containing `[2,2,6]`, then calling `l1.addAll(l2)` would result in `l1` being a list containing the elements `[1,4,7,2,2,6]`, and `l2` a list containing `[2,2,6]`.

Now let's say you want to define a variant of this method called `takeAll` which also removes all of the elements from the second list. So in the above example, after calling `l1.takeAll(l2)`, `l1` would contain `[1,4,7,2,2,6]` and `l2` would be empty.

Question 1 [**3 points**]

Describe an algorithm to efficiently implement `takeAll` when both `l1` and `l2` are `ArrayLists`.

Answer

```
l1.addAll(l2)
l2.clear()
```

Point Breakdown

- **(1 pt)** The algorithm ends with all elements of `l2` appended to `l1`.
- **(1 pt)** The the algorithm ends with `l2` empty.
- **(1 pt)** The above two requirements are both met, and the algorithm runs in $O(|\texttt{l2}|)$.

Question 2 [**2 points**]

State the runtime of your algorithm by giving the unqualified tight upper (Big-O) bound.

Answer

$O(|\texttt{l2}|)$ or $O(N)$

Point Breakdown

- **(2 pt)** The answer is correct for the algorithm described in the answer to Question 1.

Question 3 [**3 points**]

Describe an algorithm to efficiently implement `takeAll` when both `l1` and `l2` are `LinkedList`s. You may assume that there is a reference to the tail of each list.

Answer

1. Redirect the next pointer of `l1`'s tail element to the head element of `l2`.
2. Set the tail pointer of `l1` to the tail element of `l2`
3. Set the head and tail pointers of `l2` to `null`.

```
l1.addAll(l2)
l2.clear()
```

Point Breakdown

- **(1 pt)** The algorithm ends with all elements of `l2` appended to `l1`.
- **(1 pt)** The the algorithm ends with `l2` empty.
- **(1 pt)** The above two requirements are both met, and the algorithm runs in $O(1)$.

Question 4 [**2 points**]

State the runtime of your algorithm by giving the unqualified tight upper (Big-O) bound.

Answer

$$O(1)$$

Point Breakdown

- **(2 pt)** The answer is correct for the algorithm described in the answer to Question 1.