Part a: Runtime Analysis [25 pt]

For questions in this part, consider the following code. On questions about runtime bounds (Big-O, Big-Ω), provide the tightest provable bounds possible.

```
public class Mystery<T> {
  private ArrayList<T> data = new ArrayList();
  private boolean inserting = true;

  public void add(T elem) {
    if (!inserting) {
      data.reverse();
      inserting = true;
    }
    data.add(elem);
  }

  public T remove() {
    if (inserting) {
      data.reverse();
      inserting = false;
    }
    data.remove(data.size() - 1);
  }

  /* ... */
}
```

Question 1 [**5 points**]

What is the unqualified runtime (O, Ω, and Θ) of `add`? If no Θ bound exists, then write DNE for that part of your answer.

Big-O:

Big-Ω:

Big-Θ:

Answer

Big-O: $O(n)$
Big-Ω: $\Omega(1)$
Big-Θ: DNE

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**]

What is the unqualified runtime (O, Ω, and Θ) of `remove`? If no Θ bound exists, then write DNE for that part of your answer.

Big-O:

Big-Ω:

Big-Θ:

Answer

Big-O: $O(n)$
Big-Ω: $\Omega(1)$
Big-Θ: DNE

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**]

Assume there are n elements already in the data structure `mystery`. What is the tightest possible **unqualified** runtime in terms of n and m of the following snippet of code? If no Θ bound exists, then write DNE for that part of your answer.

```
for (int i = 0; i < m; i++) { mystery.add(i); }
for (int i = 0; i < m; i++) { mystery.remove(); }
```

Big-O:

Big-Ω:

Big-Θ:

Answer

Big-O: $O(n+m)$
Big-Ω: $\Omega(n+m)$
Big-Θ: $\Theta(n+m)$

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**]

Assume there are n elements already in the data structure `mystery`. What is the unqualified runtime in terms of n and m of the following snippet of code? If no Θ bound exists, then write DNE for that part of your answer.

```
for (int i = 0; i < m; i++) {
  mystery.add(i);
  mystery.remove();
}
```

Big-O:

Big-Ω:

Big-Θ:

Answer

Big-O: $O(n \cdot m)$
Big-Ω: $\Omega(n \cdot m)$
Big-Θ: $\Theta(n \cdot m)$

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 5 [**5 points**]

Is `Mystery` an implementation of a `Stack`, `Queue`, or neither?

Answer

Queue

Point Breakdown

5 pt The answer is correct.

Part b: Asymptotic Complexity [15 pt]

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**]

$$f_1(n) = 5\log n + 2n^2 + 4N$$

$f_1 \in O(\quad)$

$f_1 \in \Omega(\quad)$

$f_1 \in \Theta(\quad)$

Answer

- $O(n^2)$, $\Omega(n^2)$, $\theta(n^2)$ for variant A
- $O(2^n)$, $\Omega(2^n)$, $\theta(2^n)$ for variant B
- $O(n^2 \log n)$, $\Omega(n^2 \log n)$, $\theta(n^2 \log n)$ for variant C
- $O(n^3)$, $\Omega(n^3)$, $\theta(n^3)$ for variant 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**]

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

$f_2 \in O(\quad\quad)$

$f_2 \in \Omega(\quad\quad)$

$f_2 \in \Theta(\quad\quad)$

Answer

- $O(n)$, $\Omega(n)$, $\theta(n)$ for variant A
- $O(2^n)$, $\Omega(2^n)$, $\theta(2^n)$ for variant B
- $O(2^n)$, $\Omega(2^n)$, $\theta(n^2)$ for variant C
- $O(\log^2 n)$, $\Omega(\log^2 n)$, $\theta(\log^2 n)$ for variant 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**]

$$f_3(n) = \begin{cases} O(2^n) & \textbf{if } n \mod 3 = 0 \\ O(n^3) & \textbf{if } n \mod 3 = 1 \\ O(1) & \textbf{if } n \mod 3 = 2 \end{cases}$$

$f_3 \in O($ $)$

$f_3 \in \Omega($ $)$

$f_3 \in \Theta($ $)$

Answer

- $O(2^n)$ for variants A, C
- $O(n^3)$ for variants B, C
- $\Omega(1)$ for variants A, B
- $\Omega(n)$ for variants C, D
- Θ does not exist for any variant

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: PA Review [25 pt]

PA1 Review

Consider a Sorted Linked List that is similar to what you implemented in PA1 but with the following changes:

- It only works with ints
- It allocates a new linked list node for every element
- It adds the following method to increment the value of a node by one:

```
public void bump(LinkedListNode<int> node) {
  LinkedListNode<int> ret = this.insert(node.value + 1, node);
  this.remove(node);
  return ret;
}
```

Question 1 [**5 points**]

What is the tight, unqualified worst-case runtime (Big-O) of `bump`?

Answer

`bump` uses the 'hinted' insert operation. For one of the questions on WA2 you showed that the hinted insert is $O(n)$, since you can't guarantee that the hint will be anywhere close to the actual value.

Point Breakdown

5 pt $O(n)$

Question 2 [**5 points**]

Assume that the list has n elements in it. What is the tight big-O runtime bound (in terms of n) of the following snippet of code? In at most **two** sentences, explain why.

```
for (int i = 0; i < n; i++) {
  node = list.bump(node);
}
```

Answer

Although any individual call to `list.bump` may need to advance the inserted node past all n elements of the list, this successive series of calls to `list.bump` can not advance the inserted element past any individual element more than once. This is an example of an amortized cost.
The total runtime of the snippet is bounded by $O(n)$.

Point Breakdown

5 pt The answer identifies the runtime as $O(n)$, and provides a justification calling out that bump visits/skips each element of the list at most once, or otherwise demonstrates an understanding of bump's amortized behavior.

3 pt **(partial credit)** The answer identifies the runtime as $O(n^2)$, and provides a justification calling out an $O(n)$ operation in a loop.

2 pt **(partial credit)** The answer identifies the runtime as $O(n)$, without justification.

PA2 Review

Question 3 [**5 points**]

Finding the **shortest travel distance** between two intersections in the map of Buffalo is best accomplished by which of the following (check one):

☐ Stack
☐ Queue
☐ Priority Queue

Answer

Point Breakdown

5 pt Priority Queue (Variants A, C); Queue (Variants B, D).

Question 4 [**5 points**]

Finding path between two intersections that goes through the **fewest number of intersections** is best accomplished by which of the following data structures (check one):

☐ Stack
☐ Queue
☐ Priority Queue

Answer

Point Breakdown

5 pt Queue (Variants A, C); Priority Queue (Variants B, D).

PA3 Review

Question 5 [**5 points**]

In PA3, you wrote an algorithm to find approximate matches over two different attributes (Birthday and Zip Code) where some values were missing and there may entries with duplicate keys. Assume you are trying to find **exact** (and **only exact**) matches for four different attributes instead, and that **no values are missing or duplicated**. How many HashTables would you need? In at most **two** sentences, explain why?

Answer

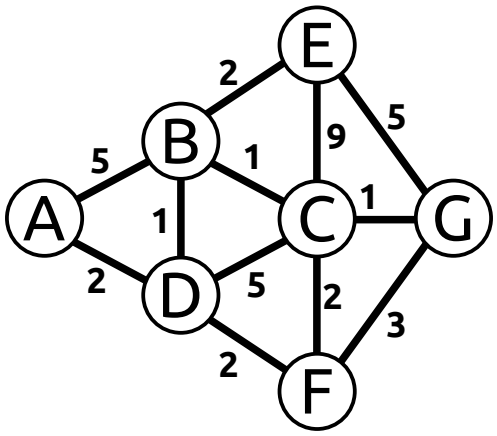
Only a single hash table is required: Load one dataset, using all four attributes jointly as a key. Then scan the elements of the other dataset, looking up their match. Since we are looking up only exact matches, we don't need any of the supplementary hash tables.
As some students implemented PA3 by preloading two sets of hash tables, partial credit is awarded in this case as well.

Point Breakdown

+3 pt Any answer that references the fact that all four attributes can be jointly used as a single key.
+2 pt Any answer that recognizes that no more than a single hash table is necessary, given the above. (+1 partial credit if they use one per dataset).
2 pt **(alternative answer)** An answer that indicates that no hash tables are required if you use nested for loops.

Part d: Graphs [25 pt]

The following questions pertain to the graph below.



Question 1 [**10 points**]

List the sequence of nodes visited by **DFS** in the order in which the algorithm visits them, with **A** as the start node. Assume that the algorithm visits each node's neighbors in alphabetical order.

Answer

Variants A, C: A, B, C, D, F, G, E
Variants B, D: A, B, D, C, E, F, G

Point Breakdown

+5 pt For Variants A, C, the path forms a straight line; For Variants B, C, the path visits nodes according to number of edges from A.
+5 pt The answer is exactly correct, as above.

Question 2 [**5 points**]

Draw, or otherwise mark the spanning tree created by the traversal from Question 1

Answer

As above.

Point Breakdown

- The spanning tree corresponds to the answer given in Question 1.

Question 3 [**5 points**]

Recall that our traversal algorithms all maintain a todo list. For the traversal from Question 1, what graph nodes are on the todo list just before the algorithm visits node C, and in what order.

Answer

Variants A, C: (bottom) [D, E] (top) or...
- C may optionally appear at the top of the stack.
- A duplicate D may optionally appear above the E, below the C if present.

Variants B, D: (head) [E, F] (tail) or...
- C may optionally appear at the head of the queue.
- A duplicate C may optionally appear between the E and F.

Point Breakdown

5 pt One of the correct variants, as above.
-1 pt The two main nodes in the list appear out of order.
-2 pt Every extra/missing node in the list.

Question 4 [**5 points**]

List the sequence of nodes visited by Dijkstra's algorithm in the order in which the algorithm visits them.

Answer

A, D, B, C, F, B, G

Point Breakdown

5 pt The correct answer, as above.
3 pt One moved, swapped, or missing node away from the correct answer.
1 pt Two moved, swapped, or missing nodes away from the correct answer.

Part e: Data Structure Selection [10 pt]

Each of the following scenarios describes a type of data and several operations that need to be performed efficiently on the data elements. For each scenario, identify one or more data structures discussed in class that could be used to organize the data, and in at most **three** sentences per scenario, explain i) what each data structure will store (e.g., what is a collection element, and what is the element's key if applicable), and (ii) how each data structure would be used to efficiently perform the operations indicated above.

- You may give answers in (short, high-level) pseudocode instead of prose.
- If we discussed an algorithm or technique in class, you should simply reference it instead of re-implementing it.
- State any assumptions you make about the implementation of the data structures you identify
- List any supplemental information you are keeping as well.

Question 1 [**6 points**]

Imagine you are designing a massively multiplayer video game. You can assume that each player's record consists of an integer PlayerID and a position on a 2-dimensional map at an (x,y) coordinate pair, where x and y are floating point numbers. You want to be able perform the following operations as efficiently as possible:

- List (enumerate) all players that are within a specific circular region.
- Obtain the location of a specific player based on their PlayerID

Answer

The first objective calls for a spatial index (a 2DMap) data structure. We discussed two in class: a K-D Tree (2-D tree) and the Quad Tree. The key here is the x and y coordinates, and accessing an element requires a tree traversal, starting at the center of the circular region, and moving up the tree omitting branches that are exclusively outside of the region. Referencing the (k-)nearest neighbor algorithm discussed in class is sufficient.
Achieving the second objective calls for a Map data structure with the PlayerID as a key. We discussed both a HashMap (Expected $O(1)$ lookup), and a Tree Map (Unqualified $O(\log n)$ lookup).

Point Breakdown

+1 pt The answer identifies the need for more than one base data structure.
+2 pt The answer identifies the need for a two-dimensional index (e.g., Spatial Index", K-D Tree", or "Quad Tree").
+2 pt The answer identifies the need for a Map (Hash or Tree/BBST) or similar lookup structure (Sorted Array) with comparable lookup costs.
+1 pt The answer makes a reasonable argument for the choice of lookup structure (e.g., Hash Map because it has (expected) $O(1)$ lookups, or Tree Map/Sorted Array because Hash Map doesn't have a guaranteed lookup time.)

Question 2 [**4 points**]

Imagine you are building a sensor monitoring system, which needs to have a collection of sensor readings. You can assume that each sensor reading is a data record consisting of a timestamp and an integer value. We want to support the following operations:

- `insert(ts, v)`: Add a new record with timestamp `ts` and value `v`. We can assume that `ts` will be newer than any previously inserted timestamp. The function should be performed in constant time.
- `expire(ts)`: Expire all records with timestamps older than `ts`, removing them from the collection. If k records are expired, this function should be performed in $\Theta(k)$ time.
- `sum()`: Compute the total value of all records. This function should be performed in constant time.

Answer

Providing the sum of all collection elements in constant time requires us to keep the sum pre-computed. This means we need to ensure that `insert` and `expire` can perform the maintenance.
`insert` in constant time limits our choices to a linked list or hash table (or an array if we are comfortable with amortized constant).
Meeting the constraints on `expire` requires us to to be able to take the lowest element out of the collection in constant time. This is a regular queue. (partial credit may be given for a priority queue).

Point Breakdown

+2 pt The answer identifies, even if obliquely, the FIFO nature of the insert/expire operation and/or states the need for a queue.
+1 pt The answer specifically identifies the linked list as the base queue data structure.
+1 pt The answer identifies the need to store a precomputed sum value.

Part f: Class Participation [Bonus: 5 pt]

Question 1 [**5 points**]

Let's go _____.

Answer

"Gamer"
The phrase was used regularly in recitation reviews for Midterm 2 and the Final, and all students were encouraged to write the phrase on their cheat sheets for good luck.

Point Breakdown

5 pt Let's go gamer. (light variations accepted)