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 unless stated otherwise.

Question 1 [**3 points**]

$$f_1(n) = \sum_{i=1}^{n^3} (20n^2 + 5i)$$

Big-O:

Big-Ω:

Big-Θ:

Question 2 [**3 points**]

$$f_2(n) = \sum_{i=0}^{\log(n^5)} \sum_{j=0}^{i-1} 2^j$$

Big-O:

Big-Ω:

Big-Θ:

Question 3 [**3 points**]

$$f_3(n) = \begin{cases} \log(n) + 9n\log(n) + n^2 & \text{if n is odd} \\ n\log(n) + 5\log(n) & \text{if n is even} \end{cases}$$

Big-O:

Big-Ω:

Big-Θ:

Question 4 [**6 points**]

Prove that the following function is in $O(n^4)$:

$$f_4(n) = 39n^2 + 10 + 5n^4$$

Part b: Hash Table Concepts

Question 1 [**12 points**]

Consider a Hash Table that uses chaining to resolve collisions, but instead of each bucket being a LinkedList, each bucket is a Sorted ArrayList. State the expected and tight unqualified runtime for each Hash Table operation discussed in class.

`insert(T elem)`	`contains(T elem)`	`remove(T elem)`
Expected:	Expected:	Expected:
Unqualified:	Unqualified:	Unqualified:

Question 2 [**8 points**]

In class we described an implementation of the Set ADT using a Hash Table. In at most a few sentences, describe what concrete changes we would need to make in order to implement the Bag ADT using a Hash Table. As a reminder, the Bag ADT is unordered (like a Set), but allows for duplicate elements.

Part c: Hash Table Implementation

For all questions in this section, you may assume that $hash_1$ and $hash_2$ are hash functions that return the following values for keys A through F:

	A	B	C	D	E	F
$hash_1$	37	13	93	24	64	58
$hash_2$	60	86	27	81	38	57

Question 1 [**6 points**]

Consider a Hash Table that has 10 buckets, uses $hash_1$ as its hash function, and resolves collisions using **chaining**. You may assume no rehash is required. State which bucket each element will end up in if elements A through F are inserted in alphabetical order.

A: ____ B: ____ C: ____ D: ____ E: ____ F: ____

Question 2 [**6 points**]

Consider a Hash Table that has 10 buckets, uses $hash_1$ as its hash function, and resolves collisions using **open addressing**. You may assume no rehash is required. State which bucket each element will end up in if elements A through F are inserted in alphabetical order.

A: ____ B: ____ C: ____ D: ____ E: ____ F: ____

Question 3 [**6 points**]

Consider a Hash Table that has 10 buckets, uses $hash_1$ and $hash_2$ as its hash functions, and resolves collisions using **cuckoo hashing** exactly as you did in PA3. You may assume no rehash is required. State which bucket each element will end up in if elements A through F are inserted in alphabetical order.

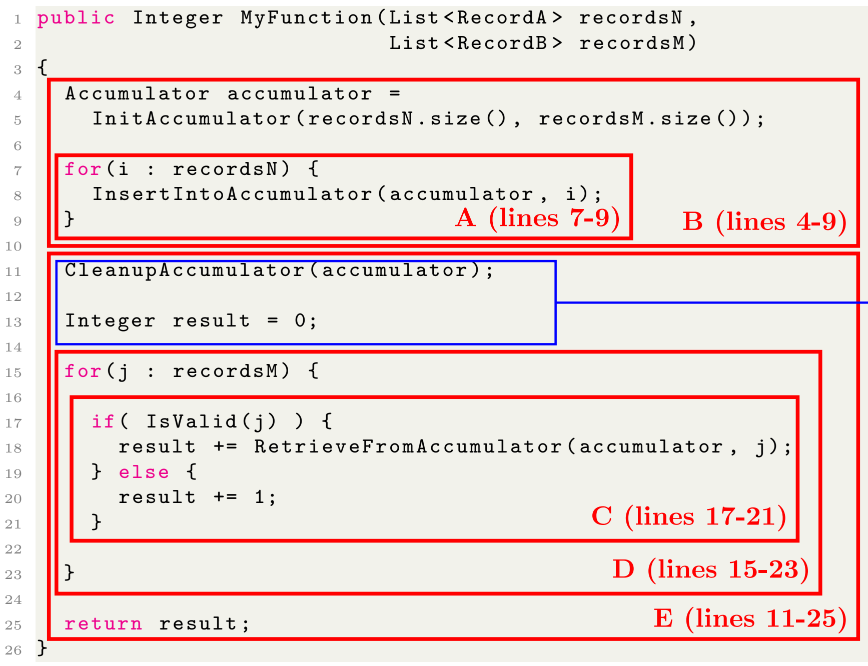
A: ____ B: ____ C: ____ D: ____ E: ____ F: ____

Question 4 [**2 points**]

If the Hash Table with **chaining** described in Question 1 was given a maximum load factor of 0.3 which element would trigger a rehash upon being inserted? If no element would trigger a rehash write “None”.

Part D: Code Runtime

This part of the exam pertains to the following function and labeled boxes. For several questions, you will be asked to provide summations of runtimes labeled with line numbers. See below for an example of a summation for lines 11-13.



```
1  public Integer MyFunction(List<RecordA> recordsN,
2                            List<RecordB> recordsM)
3  {
4    Accumulator accumulator =
5      InitAccumulator(recordsN.size(), recordsM.size());
6
7    for(i : recordsN) {
8      InsertIntoAccumulator(accumulator, i);
9    }
10
11   CleanupAccumulator(accumulator);
12
13   Integer result = 0;
14
15   for(j : recordsM) {
16
17     if( IsValid(j) ) {
18       result += RetrieveFromAccumulator(accumulator, j);
19     } else {
20       result += 1;
21     }
22
23   }
24
25   return result;
26 }
```

Example

$$\underbrace{O(1)}_{\texttt{Line 11}} + \underbrace{O(1)}_{\texttt{Line 13}}$$

The table below presents *tight* bounds on the runtime of the functions called by **MyFunction**. The letters N and M indicate the exact values of `recordsN.size()` and `recordsM.size()` respectively. The expression $|A|$ means the number of times that `InsertIntoAccumulator` has *previously* been called on A (i.e., the size of A). You may assume that iterating over a list is constant-time per element. All bounds are unqualified.

Function	**Big-O**	**Big-Ω**
`InitAccumulator(N, M)`	$O(N+M)$	$\Omega(N)$
`InsertIntoAccumulator(A, i)`	$O(\|A\|)$	$\Omega(\|A\|)$
`CleanupAccumulator(A, i)`	$O(1)$	$\Omega(1)$
`IsValid(j)`	$O(1)$	$\Omega(1)$
`RetrieveFromAccumulator(A, i)`	$O(\|A\|^2)$	$\Omega(\|A\|)$

Question 1 [**5 points**]

For the blocks of code labeled A and B, respectively, do each of the following:

1. Provide a tight upper (Big-O) bound on the runtime of the code in the rectangle as a **summation**.
2. Label the components of your summation that correspond to lines (i) 5, (ii) 7-9, and (iii) 8.
3. Expand the summation and simplify the resulting bound.

Question 2 [**5 points**]

For the blocks of code labeled C, D, and E, respectively, do each of the following:

1. Provide a tight upper (Big-O) bound on the runtime of the code in the rectangle as a **summation**.
2. Label the components of your summation that correspond to lines (i) 11, (ii) 15-21, (iii) 17-21, (iv) 17, and (v) 18, and (vi) 20.
3. Expand the summation and simplify the resulting bound.

Question 3 [**5 points**]

For the blocks of code labeled A and B, respectively, do each of the following:

1. Provide a tight lower (Big-Ω) bound on the runtime of the code in the rectangle as a **summation**.
2. Label the components of your summation that correspond to lines (i) 5, (ii) 7-9, and (iii) 8.
3. Expand the summation and simplify the resulting bound.

Question 4 [**5 points**]

For the blocks of code labeled C, D, and E, respectively, do each of the following:

1. Provide a tight lower (Big-Ω) bound on the runtime of the code in the rectangle as a **summation**.
2. Label the components of your summation that correspond to lines (i) 11, (ii) 15-21, (iii) 17-21, (iv) 17, and (v) 18, and (vi) 20.
3. Expand the summation and simplify the resulting bound.

Question 5 [**5 points**]

Provide the simplified, tight asymptotic upper (Big-O) and lower (Big-Ω) bounds on the runtime of `MyFunction` in terms of the sizes of its two inputs (`recordsN.size()` $= N$; `recordsM.size()` $= M$).

Question 6 [**5 points**]

Provide the simplified tight, *unqualified* simplified asymptotic upper (Big-O) bound on Block **D**, if we additionally provide you with the following two facts:

- $M \gg N$ (M is guaranteed to be much bigger than N, i.e., $N \in O(M)$)
- The *amortized* runtime of `RetrieveFromAccumulator(A, j)` is $O(|A|)$.

Part e: Data Structure Design

For each of the following questions, **circle** the abstract data type **and** the data structure that best fits the requirements of the collection described in the question.

Question 1 [**5 points**]

The streets and intersections of a city for use in a route-finding application.

ADT: List Stack Queue Priority Queue Graph Set Map

Data Structure: Array LinkedList ArrayList Ring Buffer Binary Heap Edge List Adjacency List Adjacency Matrix AVL Tree Red-Black Tree Hash Table

Question 2 [**5 points**]

The waiting list for a class, where students are admitted in the order they joined the waiting list.

ADT: List Stack Queue Priority Queue Graph Set Map

Data Structure: Array LinkedList ArrayList Ring Buffer Binary Heap Edge List Adjacency List Adjacency Matrix AVL Tree Red-Black Tree Hash Table

Question 3 [**5 points**]

The character's inventory in a game. For each item, identified by its name, you need to keep track of how many copies of the item you have.

ADT: List Stack Queue Priority Queue Graph Set Map

Data Structure: Array LinkedList ArrayList Ring Buffer Binary Heap Edge List Adjacency List Adjacency Matrix AVL Tree Red-Black Tree Hash Table

Question 4 [**5 points**]

A checklist of every quest the story's protagonist needs to complete to succeed. We only care about a quest until it is checked off, and so we want to be able to quickly remove tasks from the checklist. Every quest is identified by a magical reference to the quest's entry in the checklist, and may be completed in any order.

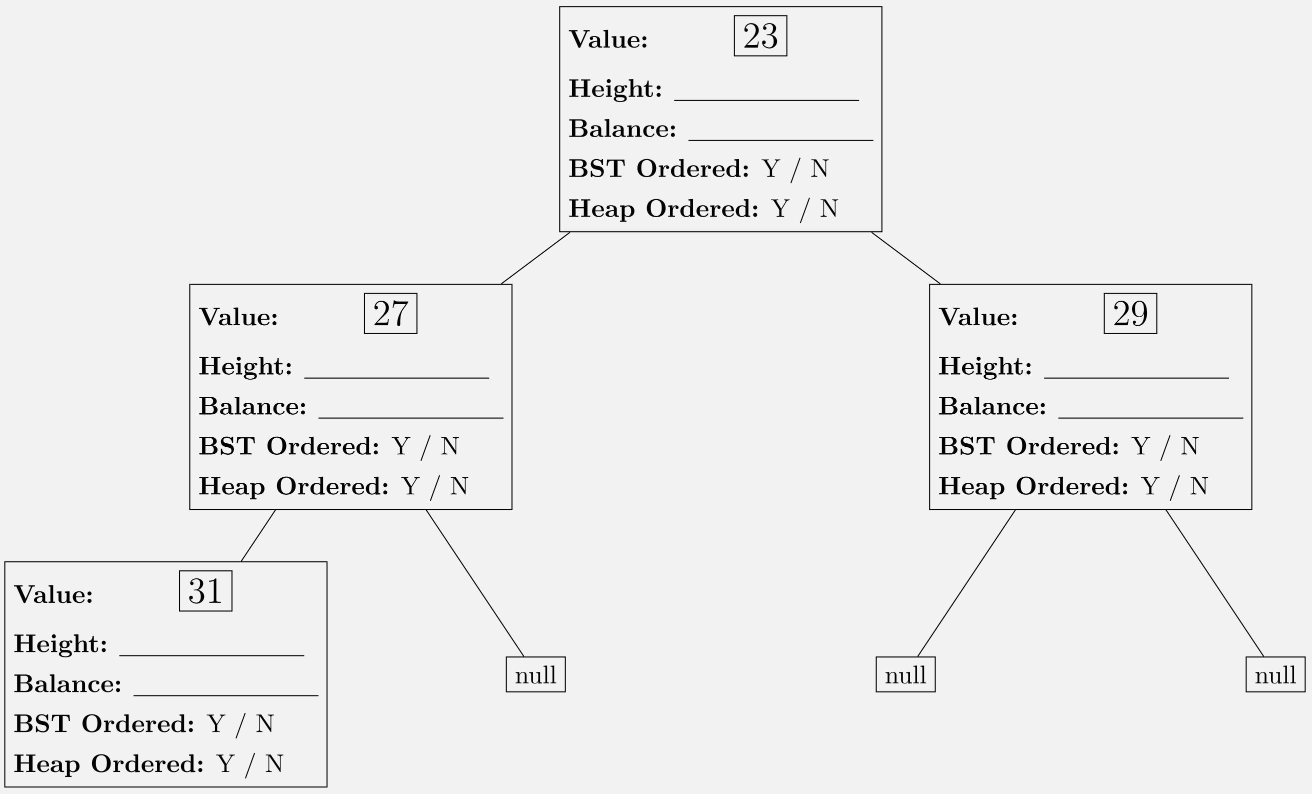
ADT: List Stack Queue Priority Queue Graph Set Map

Data Structure: Array LinkedList ArrayList Ring Buffer Binary Heap Edge List Adjacency List Adjacency Matrix AVL Tree Red-Black Tree Hash Table

Part f: Heaps and Binary Search Trees

Question 1 [**20 points**]

For each node in the tree provided below, mark down (i) the **height** of the node, (ii) the **balance factor** of the node, (iii) whether the node satisfies the **BST ordering property**, and (iv) whether the node satisfies the **min-heap *ordering* property**. Give your answer for each node as if it were the root of the tree.



Part g: Class Participation [Bonus]

Question 1 [**5 points**]

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

Scrap Page

Scrap Page

Scrap Page