CSE 373 20au: Exam I

Note: This exam was originally offered via Gradescope. Free response fields are omitted in this PDF reproduction.

1. Instructions

- This exam is written to be completed in 1-2 hours. To maximize flexibility, it will be available for 48 hours. **It is due strictly a 8:30 am PDT on Wednesday, October 28.** You may not use late days on this exam.
- You may work in multiple sessions, and submit as many times as you'd like (we only grade your last one).
- You may work alone, or in groups of up to 6 (and submit a single exam). Group submission instructions:
  - First, have one person click "Save Answer" on any question. Then, scroll to the very bottom and click "Submit & View Submission".
  - You can now add your group members in the upper right corner, just like a programming assignment.
  - Then, you can click "Resubmit" to edit your answers as many times as you want, with the same group.
  - Be careful not to have two people edit answers at the same time (Gradescope may lose something). We recommend coordinating through a different platform and having one person submit for the group.
- This test is open-note and open-internet. However, you are not permitted to share these questions or get help from anybody not enrolled in 373 20au, such as former students.
- During the exam, you may ask clarification questions on Ed or in office hours. Course staff will not answer questions about course concepts or give hints on specific exam questions.
- Sentence estimates are just to clarify our expectations. You will not be penalized for writing more or less.
- Each question is annotated with approximate corresponding learning objectives from lecture. This is purely extra information and only intended to make the link between what you've learned and what we're testing clearer: don't read too much into them or worry whether you've adequately demonstrated the skill! :)
- Any significant exam clarifications will be posted on Ed.

This problem asks you to affirm you and your group have read all of these policies and those on the website. Failure to fill this question out may result in your exam not being marked for credit.

2. ADTs and Design Decisions

Learning Objectives: Compare the runtime of ArrayList and LinkedList (LEC02), Distinguish the List ADT from its implementations (LEC02), Describe the Stack, Queue, and Map ADTs (LEC03), Compare the runtime of Stack, Queue, and Map operations on a resizable array vs. linked nodes (LEC03).

Scenario: Congrats! You just got hired as an intern to be the fourth developer at InnerSloth, the company that made the popular game, 'Among Us'. Don't worry, you don't need to know much about the game to work on this problem, we will explain anything you need.

The game is normally played with at most 10 players, where some of them are imposters amongst the crewmates. One of the game's central mechanics is calling a meeting where you discuss and decide which player to vote out because they are "sus" (short for "suspicious", since no one can spell that). The player that receives the most votes is voted off and ejected from the spaceship. There may be multiple meetings called in the game, but for this problem we are only considering modeling a single meeting.

Normally, the game of 'Among Us' is played with at most 10 players in a lobby. Your first job as intern is to lift that cap and allow a boundless number of n players play the game at once. We don't know in advance how many players will be in the game and players will be allowed to join mid-game. We will define a `MeetingVotes` ADT that keeps track of the votes cast during a single meeting to track who should get voted off. The ADT is designed to only store votes for a single meeting, so once another meeting starts, we will use a new instance of an implementation of the ADT. In other words, you don't need to worry about votes carrying over between meetings.

The `MeetingVotes` ADT defines the following operations. For each operation, you can assume that the player names are unique and that any player is only allowed to cast a vote at most once:

- `castVote(String player1, String player2)` – Casts a vote that `player1` thinks `player2` is "sus". This will count as one more vote towards `player2`'s votes received.
- `int numVotes(String player)` – Returns the number of votes the given `player` has received the meeting.

In the following problems, we will describe various implementations of this ADT and you will describe the runtimes of the operations in the implementation. When doing analysis for these problems, let the variable n refer to the total number of votes cast. You should be able to express all of your bounds in terms of n. *For analyzing any arrays in the worst case, you may **not** assume that resizing doesn't happen.*

2.1. Linked Nodes Implementation

Suppose we implement this ADT with a singly-Linked List of contributions, where each contribution is represented by a Pair object containing the player who cast the vote and who they cast it for. *This singly linked list only has a pointer to the front of the list and each node only has a pointer to the next node in the list*. Casting a vote is done by inserting a new Pair at the *END* of the list (i.e. index n). For each method, give a simplified worst-case Big-Theta bound in terms of n (the number of total votes) for how the method would need to be implemented, or write "N/A" if one doesn't exist. You don't need to justify your answers.

(a) `castVote` worst case Big-Theta.

(b) `numVotes` worst case Big-Theta.

2.2. Array Implementation

Now, suppose we implement this ADT with an ArrayList of contributions, again where each contribution is represented by a Pair object containing the user and text. As before, adding votes is done by inserting a new Pair at the *END* of the list (i.e. index n). For each method, give a simplified worst-case Big-Theta bound in terms of n (the number of total contributions) for how the method would need to be implemented, or write "N/A" if one doesn't exist. You don't need to justify your answers.

(a) `castVote` worst case Big-Theta.

(b) `numVotes` worst case Big-Theta.

2.3. Custom Implementation

Come up with your own implementation of the ADT that optimizes for both the `castVote` and `numVotes` runtimes. You may use any data structures implementing the List, Stack, Queue, Deque, and Map ADTs covered in the course, and you may use any number of data structures (or combine them). Your only restriction is that you will want to choose an implementation that makes the operations as (asymptotically) fast as possible. Describe your implementation below, then give a worst case Big-Theta bound for each method. There may be many correct answers to this question.

(a) Describe your implementation. For full credit, we'll look for: (1) you explicitly mention the names of any data structures (and their corresponding ADTs) that you use, (2) your chosen implementation demonstrates you've thought about how to do these tasks reasonably efficiently, and (3) you give a high-level description of what happens in each method that is specific enough to come up with Big-Theta bounds. (2-3 sentences)

(b) `castVote` worst case Big-Theta.

(c) `numVotes` worst case Big-Theta.

3. Oh/Omega/Theta Bounds

Learning Objectives: Identify whether Big-Oh (and Big-Omega, Big-Theta) statements about a function are accurate (LEC04), Explain why we can throw away constants when we compute Big-Oh bounds (LEC04), Differentiate between Big-Oh, Big-Omega, and Big-Theta (LEC05), Describe the difference between Case Analysis and Asymptotic Analysis (LEC05).

3.1. Big-Omega

Suppose we know that $f(n)$ **is a function in** $\Omega(n^2)$. Given that that's all we know about $f(n)$, for each of the following statements, indicate if it is ALWAYS true, NEVER true, or SOMETIMES true (more information about $f(n)$ would be needed to determine if it's true). You do not need to justify your answers.

3.2) $f(n)$ is in $\Omega(n^3)$.

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.3) $f(n)$ is in $\Omega(n)$.

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.4) There exists some constant n_0, such that for all $n \geq n_0$, $f(n) \geq n^2$.

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.5) $f(n)$ is in $\mathcal{O}(n^3)$.

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.6) $f(n)$ is in $\mathcal{O}(n)$.

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.7) $f(n)$ has a Big-Theta bound.

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.8) $f(n)$ is $\Theta(n^3)$

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.9) $f(n)$ describes a runtime of a best-case of case analysis

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.10. Big-Theta

Suppose we know that $g(n)$ **is a function in** $\Theta(n^3)$. Given that that's all we know about $g(n)$, for each of the following statements, indicate if it is ALWAYS true, NEVER true, or SOMETIMES true (more information about $g(n)$ would be needed to determine if it's true). You do not need to justify your answers.

3.11) $g(n)$ is in $\mathcal{O}(n^3)$.

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.12) $g(n)$ is in $\Theta(n)$.

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.13) $g(n)$ is in $\Omega(n)$

◯ ALWAYS ◯ NEVER ◯ SOMETIMES (Need more information)

3.14 Big-Oh

This problem will ask you to think about the relationships of functions and their complexity classes. Suppose we had three functions $f(n), g(n), h(n)$ and that **we know that $f(n)$ is a function in $\mathcal{O}(g(n))$ and $g(n)$ is a function in $\mathcal{O}(h(n))$**. Given that that's all we know about the functions, select all of the following that are guaranteed to be true. You do not need to justify your answer or give any proofs.

Hint: Think carefully about the definition of Big-Oh. If you are able to find any specific examples of $f(n), g(n), h(n)$ that makes a statement below false, then it can't possibly always be true.

- ☐ $f(n)$ is $\mathcal{O}(h(n))$
- ☐ $h(n)$ is $\Omega(f(n))$
- ☐ $f(n) + g(n)$ is $\mathcal{O}(g(n))$
- ☐ $f(n) \cdot g(n)$ is $\mathcal{O}(h(n))$
- ☐ If we also know that $f(n)$ is in $\Theta(h(n))$, then it must be true that $g(n)$ is also in $\Theta(h(n))$

4. Algorithmic Analysis

Learning Objectives: Come up with Big-Oh, Big-Omega, and Big-Theta bounds for a given function (LEC05), Perform Case Analysis to identify the Best Case and Worst Case for a given piece of code (LEC05).

Scenario: The following methods are implementations of an algorithm that looks at the difference between any two numbers in an array of numbers, and reports whether or not all pairwise differences are less than some parameter `maxDiff`. For this problem, we will assume the given array is non-empty.

For example, if we called `constrainedDiffs([1, 2, 3, 4], 2)`, it would return `false` because there is a pair of numbers (1 and 4) whose difference exceeds the target value 2.

The following are two different implementations of this method. Each problem will ask you to analyze one implementation. **When asked to analyze the runtime, do so in terms of n, the length of the array of numbers.**

4.1. Implementation A

```java
public static boolean constrainedDiffs(int[] nums, int maxDiff) {
    for (int i = 0; i < nums.length; i++) {
        for (int j = 0; j < nums.length; j++) {
            int diff = nums[i] - nums[j];
            if (diff >= maxDiff) {
                return false;
            }
        }
    }
    return true;
}
```

(a) What is the simplified Big-Theta runtime in terms of n for this method in **the worst case**?

(b) What is the simplified Big-Theta runtime in terms of n for this method in **the best case**?

(c) Describe the inputs that would lead to the best and worst case for this method (consider giving example inputs if it would make your description more clear). If there is no difference between the two cases, note that instead (no explanation needed). (1-3 sentences)

4.2. Implementation B

```java
public static boolean constrainedDiffs(int[] nums, int maxDiff) {
    int min = nums[0];
    int max = nums[0];

    for (int i = 0; i < nums.length; i++) {
        if (nums[i] < min) {
            min = nums[i];
        }
    }

    for (int i = 0; i < nums.length; i++) {
        if (max < nums[i]) {
            max = nums[i];
        }
    }

    return (max - min) < maxDiff;
}
```

(a) What is the simplified Big-Theta runtime in terms of n for this method in **the worst case**?

(b) What is the simplified Big-Theta runtime in terms of n for this method in **the best case**?

(c) Describe the inputs that would lead to the best and worst case for this method (consider giving example inputs if it would make your description more clear). If there is no difference between the two cases, note that instead (no explanation needed). (1-3 sentences)

5. Recursive Code Analysis

Learning Objectives: Model recursive code using a recurrence relation (LEC06), Describe the 3 most common recursive patterns and identify whether code belongs to one of them (LEC06), Use the Master Theorem to characterize a recurrence relation (LEC06), Characterize a recurrence with the Tree Method (LEC07).

5.1. Recursive Code Modeling

Consider the following code. In this question, you will write a recurrence to model the **runtime** (not the result) of this Java method in terms of n, the length of the given `LinkedDeque`. You should assume the `LinkedDeque` uses the same implementation as described in Project 1.

```java
public static void reverseDeque(LinkedDeque<Integer> deque) {
    int size = deque.size();
    for (int i = 0; i < size; i += 1) {
        System.out.println(deque.get(i));
    }

    if (size > 1) {
        int item = deque.removeFirst();
        reverseDeque(deque);
        deque.addLast(item);
    }
}
```

Write a recurrence for the worst-case runtime of this code. If you need to use constants but their exact value isn't known or doesn't matter to the overall runtime, use c_1, c_2, c_3, etc.

Follow this recurrence template:

$$T(n) = \begin{cases} \text{BASE\_CASE}, & \text{BASE\_CONDITION} \\ \text{RECURSIVE\_CASE}, & \text{RECURSIVE\_CONDITION} \end{cases}$$

(a) Give BASE_CASE and BASE_CONDITION:

(b) Give RECURSIVE_CASE and RECURSIVE_CONDITION:

5.2. Characterizing Recurrences

Give a Big-Theta bound for this recurrence. You may use any technique to do so; you do not have to show your work.

$$T(n) = \begin{cases} 17 & \text{if } n < 25 \\ 9T(\frac{n}{2}) + n^3 & \text{otherwise} \end{cases}$$

6. Hash Maps

Learning Objectives: Differentiate between the "worst" and "in practice" runtimes of a Separate Chaining Hash Map, and describe what assumptions the latter involves (LEC08), Compare the relative pros/cons of various Map implementations (LEC08), Describe the properties of a good hash function and the role of a hash function in making a Hash Map efficient (LEC09), Trace operations in a Separate Chaining Hash Map on paper (such as insertion, getting an element, resizing) (LEC08).

6.1. Hashing

Consider the following Java class. In this problem, we will use the short-hand syntax `Numbers([1, 2, 3])` to represent a `Numbers` object initialized with the numbers `[1, 2, 3]`.

```java
public class Numbers {
    private List<Integer> nums;

    public Numbers(int[] nums) {
            this.nums = new ArrayList<>();
        for (int num : nums) {
           this.nums.add(num);
        }
    }

    public int hashCode() {
        int hash = 0;
        for (int num : nums) {
            hash += num;
        }
        return hash;
    }

    // Other methods include an appropriate equals
}
```

6.2) Consider if we use a hash table with $M = 3$ buckets (assume no resizing for this problem). Which of the following `Numbers` will collide with `Numbers([4, 1, 6])`. *Select all that apply.*

☐ `Numbers([1])` ☐ `Numbers([2])` ☐ `Numbers([7, 3, 7])`
☐ `Numbers([5, 6])` ☐ `Numbers([1, 2])` ☐ `Numbers([])`

6.3) Does there exist a `Numbers` that is guaranteed to collide with `Numbers([4, 1, 6])` on any choice of M buckets?

◯ Yes ◯ No ◯ Not enough information to tell

6.4. Resizing

Recall our separate chaining hash map with linked lists for each bucket. Consider what would happen if we removed the ability for this hash map to resize. In other words, we will initialize the hash table to be size 10 and then never resize it once it becomes too full. Could this implementation of the chaining hash map be used as a valid implementation of the Map ADT (e.g., it would be functionally correct)? Make sure your answer clearly states whether or not this would be able to function without errors. Additionally, depending on your answer, also discuss the following:

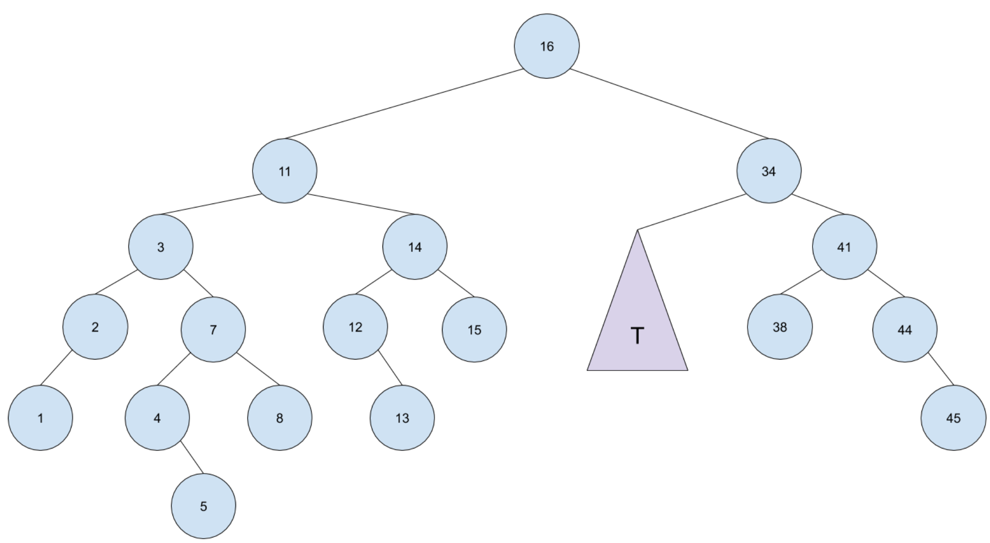
- If your answer is yes (it will work without errors), discuss whether or not the "in practice" runtime of the structure differs from a hash table that resizes. Explain why that changed does or doesn't happen.
- If your answer is no (this implementation won't work due to some error), give an example sequence of operations that would cause this hash table to break and briefly justify what specifically causes this implementation to malfunction.

7. Trees

Learning Objectives: Apply the BST invariant and describe its implications (LEC10), Describe how AVL Rotations and invariants lead to an efficient runtime (LEC10), Identify invariants for data structures (LEC03).

7.1. AVL Tree Invariants

Consider this AVL tree of integers, where an entire subtree is left out and is labeled T.



7.2) What range of values can be found for the keys in the subtree T such that this whole tree is an AVL Tree? Assume duplicates are not allowed.

- If there is only one possible value, state it.
- If there is a range of values $min - max$ where min is the minimum possible value (inclusive) and max is the maximum possible value (also inclusive), state that.

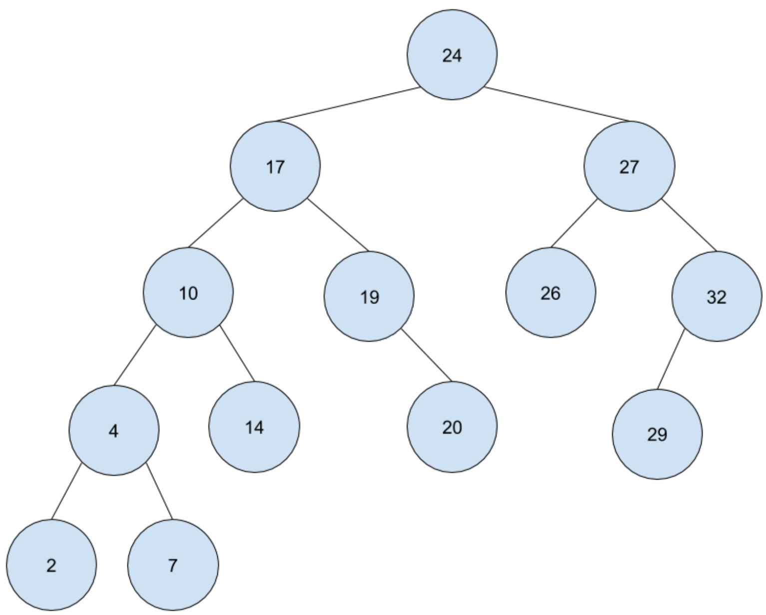
- If there are no constraints on the values, say *any values*.
- If there are no possible values to put here, say *N/A*

7.3) What are the possible heights of the subtree T such that this entire tree is an AVL Tree.

- If there is only one possible height, state it.
- If there are multiple possible heights, list all of them (min and max should be inclusive).
- If this is not possible, write *N/A*.

7.4. AVL Insertion

Consider this AVL tree. After inserting the value 3, what is the resulting tree after any AVL rotations? Upload a drawing of your final tree.



7.5. AVL Rotations, but backwards?

Suppose we had the following AVL Tree. We are interested in figuring out what value would be necessary to insert into the tree, such that after any AVL rotation(s), the node with the value 32 become the root of the whole tree.

7.6) What *single* value could we insert into this AVL tree using an AVL insert operation that would cause the root of the tree to become the node with the key 32? There may be multiple correct answers, but you should pick one. Recall, we don't allow duplicates in these trees.

7.7) Draw the final tree after the insertion happens and any rotations occur. Upload your answer as an image of the final tree.