

# CSE 373: 23 Spring Midterm

---

Name:

UW Email:

@uw.edu

## Instructions

- The allotted time is **50** minutes. Please do not turn the page until staff says to do so
- This is an open-book and open-notes exam. You are **NOT** permitted to access electronic devices including calculators.
- Read the directions carefully, especially for problems that require you to show work or provide an explanation.
- We can only give partial credit for work that you've written down.
- Unless otherwise noted, every time we ask for an  $O$ ,  $\Omega$ , or  $\Theta$  bound, it must be simplified and tight.
- If you run out of room on a page, indicate where the answer continues. Try to avoid writing on the very edges of the pages: we scan your exams and edges often get cropped off.

## Advice

- If you feel like you're stuck on a problem, you may want to skip it and come back at the end if you have time.
- Relax and take a few deep breaths. You've got this :-)

Question	Max Points
1. ADT Design	50
2. Code Analysis	50
<b>Total</b>	<b>100</b>

## Resubmission Details

- This exam will be graded out of 100 points. If you are not satisfied with your grade, you will be given the opportunity to resubmit it online and earn up to 50% of the missed points back.
- For example, a student scoring 80/100 points may receive up to 90/100 points on the resubmission.



# 1 ADT Design

Kasey will be teaching a new class for UW next year, CSE 737. This class is expected to be very popular, and has no capacity restriction!

Kasey has tasked you with managing the system she will be using to catalog all of the students as they register for the course.

Your job is to write a program that implements the **Dictionary ADT**, with a student's *unique* ID as a **key**, and their name as a **value**.

You can assume each student ID is a positive integer of fixed length  $k$ , each student's name has max length  $s$ , and there are  $n$  students to be registered for the course.

Students will register in order of student ID, starting with seniors, having the lowest student ID numbers and ending with freshmen, having the highest ID numbers.

This program will need the following functionality:

- *void putStudent(int ID, String name)*
  - Adds a student's ID and their name into the dictionary.
- *String getStudent(int ID)*
  - Returns the name of the student with the given ID from the dictionary.
  - If there is no student with the given ID, it returns *null*.
- *void printNames()*
  - In ascending order (smallest to largest) of ID, prints all names in a list.

For the following three methods, give the tight Big-O runtime on a single call of the method if this ID  $\rightarrow$  Name dictionary was to be implemented with the supplied data structure. You should give the *in-practice runtime* for the specific *case* described in this scenario in terms of  $k$ ,  $s$ , and  $n$ , wherever applicable.

---

*void putStudent(int ID, String name)*

Trie

Binary Search Tree

AVL Tree

---

*String getStudent(int ID)*

Trie

Binary Search Tree

AVL Tree

*void printNames()*

Trie

Binary Search Tree

AVL Tree



Suppose the order of the students added to the course was completely randomized rather than in registration order. How, if at all, would that impact the performance of each of the above data structure choices?

Explain your answer in 2-3 sentences.



In this given randomized scenario, among the given data structures, select one that is most optimal in terms of runtime and spatial complexity. Justify your decision.

Explain your answer in 2-3 sentences.



Suppose the new method `void printIDs(String name)` was added. This method accepts a `String name`, representing a student name, and prints all IDs associated with this name in the class. In order to implement this method in constant  $O(1)$  time, a secondary data structure will be needed in addition to the existing dictionary.

Which data structure is overall best suited for this? Describe how it would work in conjunction with our existing program, as well as how it would be populated and utilized. You may choose any data structure from the course as you see fit.

Explain your answer in 2-3 sentences.

What drawbacks, if any, does implementing `printIDs` with an additional data structure have, in comparison to only using the previous `ID → Name` dictionary?

Explain your answer in 1-2 sentences.

## 2 Code Analysis

Simon is creating a new arcade game and needs your help to implement a leaderboard system for the game.

The following functionality is needed:

- `void addScore(int playerId, int score)`
  - If there is no player with such unique ID in the system, add them in with the given score
  - If the player is already in the system, update their score
- `int top(int K)`
  - Return the sum of the top K scores
  - K is a valid argument (positive, not greater than the current number of players)

Simon came up with two implementations of the leaderboard system, and needs your help deciding which one is better. Analyze the two implementations below, and answer the subsequent questions. Be sure to pay attention to the in-line comments!

You can assume there are  $n$  players added to the system, and that  $k$  represents the parameter passed into the `top(int K)` method.



Solution A:

```
class LeaderboardOne {  
  
    private HashMap<Integer, Integer> scores;  
  
    public LeaderboardOne() {  
        this.scores = new HashMap<Integer, Integer>();  
    }  
  
    public void addScore(int playerId, int score) {  
        if (!this.scores.containsKey(playerId)) {  
            this.scores.put(playerId, 0);  
        }  
        this.scores.put(playerId, this.scores.get(playerId) + score);  
    }  
  
    public int top(int K) {  
  
        // Create an ArrayList for all values stored in dictionary  
        List<Integer> values = new ArrayList<Integer>(this.scores.values());  
  
        // Sort the ArrayList 'values' from highest to lowest  
        // Collections.sort() runs in O(NlogN)  
        Collections.sort(values, Collections.reverseOrder());  
        int total = 0;  
        for (int i = 0; i < K; i++) {  
            total += values.get(i);  
        }  
        return total;  
    }  
  
}
```

## Solution B:

```
class LeaderboardTwo {

    Map<Integer, Integer> scores;
    TreeMap<Integer, Integer> sortedScores;

    public LeaderboardTwo() {
        this.scores = new HashMap<Integer, Integer>();
        // Creates a TreeMap with keys sorted from highest to lowest
        this.sortedScores = new TreeMap<>(Collections.reverseOrder());
    }

    public void addScore(int playerId, int score) {
        if (!this.scores.containsKey(playerId)) {
            this.scores.put(playerId, score);
            if (!sortedScores.containsKey(score)) {
                sortedScores.put(score, 0);
            }
            sortedScores.put(score, sortedScores.get(score) + 1);
        } else {
            int preScore = this.scores.get(playerId);
            int playerCount = this.sortedScores.get(preScore);
            if (playerCount == 1) {
                this.sortedScores.remove(preScore);
            } else {
                this.sortedScores.put(preScore, playerCount - 1);
            }
            int newScore = preScore + score;
            this.scores.put(playerId, newScore);
            if (!sortedScores.containsKey(newScore)) {
                sortedScores.put(newScore, 0);
            }
            sortedScores.put(newScore, sortedScores.get(newScore) + 1);
        }
    }
}
```

```
public int top(int K) {

    int count = 0;
    int sum = 0;

    // loops over the scores in the TreeMap from highest to lowest
    // Map.Entry: Item containing a Key and Value from the Map
    // .entrySet() executes in O(1) time
    for (Map.Entry<Integer, Integer> entry: this.sortedScores.entrySet()) {
        int times = entry.getValue();
        int key = entry.getKey();
        for (int i = 0; i < times; i++) {
            sum += key;
            count++;
            if (count == K) {
                break;
            }
        }
        if (count == K) {
            break;
        }
    }
    return sum;
}
```

For the following three methods, give the tight Big-O runtime if this leaderboard was to be implemented with the chosen solution. You should give the *in-practice runtime* for each method described in this scenario in terms of  $k$  and  $n$ , wherever applicable.

*Solution 1:*

`void addScore(int playerId, int score)`

`int top(int K)`

---

*Solution 2:*

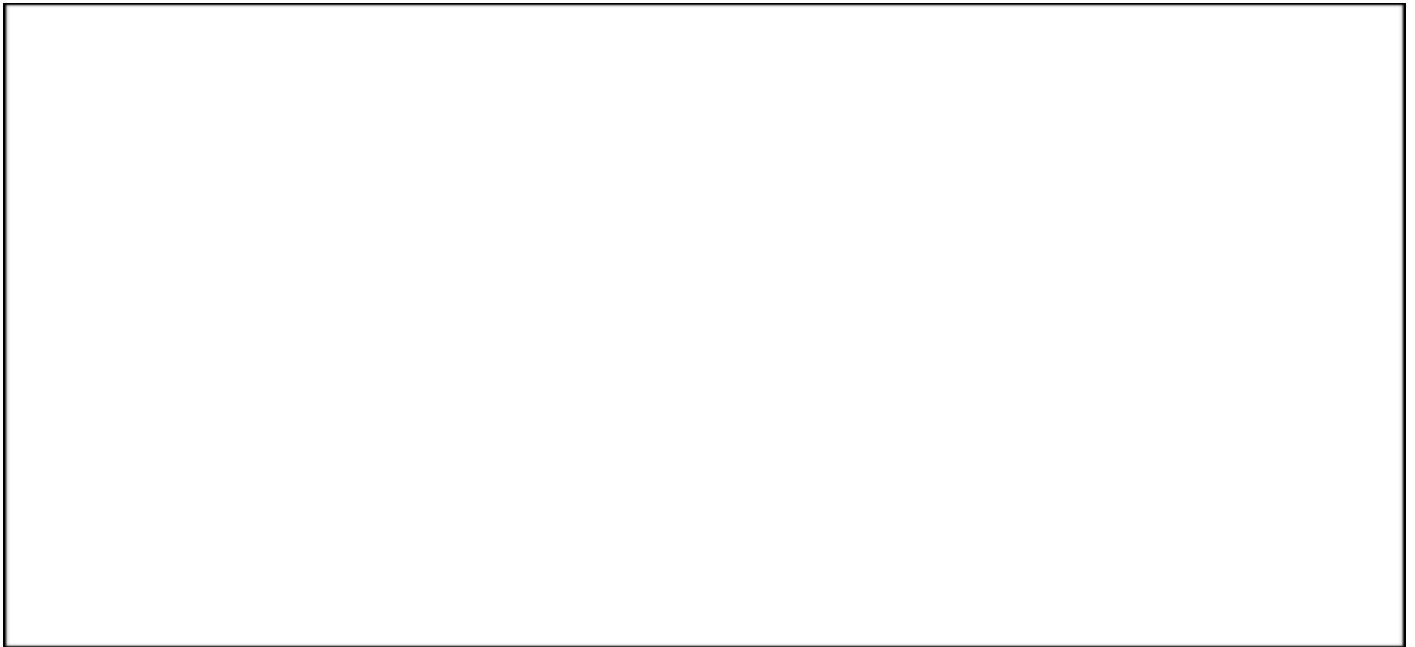
`void addScore(int playerId, int score)`

`int top(int K)`

Each of these solutions has tradeoffs. Discuss under what conditions or scenarios either solution might be more effective or efficient than the other and why.

Explain each answer in 2-3 sentences.

Solution 1:



Solution 2:

