# Exam I Practice Problem Set **Solutions**

*CSE 373 Summer 2020*

The purpose of this problem set is to help you prepare for Exam I by giving you examples of the types of skills you may be asked to use. Please note that it is not intended to be representative of the length of the exam nor the exact types of questions you will be asked. For a more thorough list of the topics that may be included, refer to the topics list and learning objectives list published on the exams page on the course website.

# 1. Design Decisions

For each of the scenarios described below, select one of the following ADTs that would best suit the situation. Then, justify your answer by briefly describing how you would use the ADT. Your justification should mention a specific operation on the ADT, and *why* that operation would be important in this particular situation.

You can choose from the following ADTs: **List, Stack, Queue,** and **Map**. Each ADT should be used exactly once.

## 1.1 NFL Draft

You are writing a portion of a program to manage the NFL draft. In the draft, teams rotate through 7 rounds of picking in which the team with the worst record for the previous year picks first, followed then by the second and so on until the team that won the super bowl picks last. The order of picking is the same in each round. Which ADT would you choose to manage the order of the teams as they rotate through their turn each round?

> **Queue. Because this situation only requires us to access the next team at any given point, and afterward they are put back in the same order for the next round, we can use the remove and add methods of a Queue to achieve this effect. A Queue doesn't let us easily change the order of teams, which is appropriate here since we choose an initial order for the teams and don't change it afterward.**

## 1.2 Music Festival

You are writing a program to manage the arrangement of songs at a music festival. Which ADT would you choose to store the order of songs as you build out the show? The process of designing the show will be iterative and you will need the ability to move, add and remove songs before you get to your final design.

**List. This situation requires us to add, remove, and move songs, and in particular to manipulate songs that may be in the middle of the playlist, which are operations provided by the List ADT. Choosing a List is also well-suited because the playlist needs to be ordered.**

## 1.3 Inbox

You are writing a program to manage unread emails in an inbox. When a new mail arrives, it should appear at the very top of the inbox and as the mails are read, they should be removed from the unread view. Which ADT would you choose to manage the ordering of the mails?

**Stack. Since the most recently-input mail in this situation should be the first one read/removed, we want an ADT that is Last-In-First-Out (FIFO) like a Stack. In addition, this situation calls for ordered data.**

## 1.4 Online Store

You are writing a program to return search results for an online store. When a user types in a query you should return all the products that include the search term in their title. Which ADT would you choose to manage the products and their titles?

**Map. This problem does not enforce any kind of ordering between products, but has two distinct sets of data (products and their titles) that need to be associated with each other. In this situation it will be a common operation to translate from a title to a product, and the Map ADT is designed to provide this functionality specifically.**

# 2. Case and Asymptotic Analysis

Scenario: Joyce is a very nervous boi™ trying to land her first SWE internship at a company called Schmoogle™. On the HackerRank challenge they sent her, she was faced with the following problem:

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target. You may assume that each input would have ***exactly*** one solution, and you may not use the same element twice.

Example:

```
Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].
```

The company states that she should code her solution as if it were to handle large inputs (~1,000,000 entries) on a computer with a huge amount of memory space. She comes up with the following 3 solutions to the problem, but isn't quite sure which one to go with. And so, she turns to you, hoping that you could help her analyze her code (wow she's cheating on a coding challenge??? Already fired lol).

***NOTE***: assume that Joyce is utilizing Java's implementation of HashMap within these solutions. You may assume that both map.get() and map.containsKey() consistently run in O(1) time.

## 2.1 Programming Solution A

```
public int[] twoSum(int[] nums, int target) {
    for (int i = 0; i < nums.length; i++) {
        for (int j = i + 1; j < nums.length; j++) {
            if (nums[j] == target - nums[i]) {
                return new int[] { i, j };
            }
        }
    }
    throw new IllegalArgumentException("No two sum solution");
}
```

a) What is the Big-Theta runtime for this method in the worst case? **$\Theta(n^2)$**
b) What is the Big-Theta runtime for this method in the best case? **$\Theta(1)$**
c) Describe the best and worst case for this method, using example input arrays/target ints to illustrate your response (if applicable). If there is no difference between the two cases, note that instead. (~1-2 sentences)
**The best case would be if the first two elements of the input add up to the target (ex. [1, 2, 3, 4, 5] target = 3). The worst case would if the last two elements of the input add up to the target (ex. [1, 2, 3, 4, 5] target = 9).**

## 2.2 Programming Solution B

```
public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];
        if (map.containsKey(complement)) {
            return new int[] { map.get(complement), i };
        }
        map.put(nums[i], i);
    }
    throw new IllegalArgumentException("No two sum solution");
}
```

a) What is the Big-Theta runtime for this method in the worst case? **Θ(n)**
b) What is the Big-Theta runtime for this method in the best case? **Θ(1)**
c) Describe the best and worst case for this method, using example input arrays/target ints to illustrate your response (if applicable). If there is no difference between the two cases, note that instead. (~1-2 sentences)
**The best case would be if the first two elements of the input add up to the target (ex. [1, 2, 3, 4, 5] target = 3). The worst case would if the last two elements of the input add up to the target (ex. [1, 2, 3, 4, 5] target = 9).**

## 2.3 Programming Solution C

```
public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; i++) {
        map.put(nums[i], i);
    }
    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];
        if (map.containsKey(complement) && map.get(complement) != i) {
            return new int[] { i, map.get(complement) };
        }
    }
    throw new IllegalArgumentException("No two sum solution");
}
```

a) What is the Big-Theta runtime for this method in the worst case? **Θ(n)**
b) What is the Big-Theta runtime for this method in the best case? **Θ(n)**
c) Describe the best and worst case for this method, using example input arrays/target ints to illustrate your response (if applicable). If there is no difference between the two cases, note that instead. (~1-2 sentences)
**There is no difference between the best and worst case. Regardless of the position of elements in the input, the solution still runs in linear time.**

## 2.4 Comparing Solutions

Out of solutions A, B and C, which solution seems to be the best or “most optimal”? Defend your selection in 2-3 sentences.

**“Meh” Answer: The best solution is Solution B, because it has the fastest worst case and best case (Θ(n) and Θ(1), respectively). It takes the best aspects of Solution A and Solution C and combines them in one.**

*The above answer is not good because the response does not mention the context of the problem within the defense (especially since the problem specifically mentions storage space). It would only get partial credit.*

**Good Answer: Since the problem mentions that the code will handle large inputs on a machine with ample space, we should choose the solution with the fastest best case/worst case runtime, even if it uses up more memory. Therefore, Solution B is the best choice.**

# 3. Hashing

In the following problems, refer to the following `Point` class.

```
public class Point {
	public final int x, y;
	public Point(int x, int y) {
		this.x = x;
		this.y = y;
	}
	public int hashCode() {
		return this.x + this.y;
	}
}
```

## 3.1 Collisions

a) Which of the following points will collide with `Point(1, 2)` in a hash table with 2 buckets?

`Point(1, 1)`
`Point(3, 1)`
`Point(1, 4)` **`COLLISION`**
`Point(2, 1)` **`COLLISION`**
`Point(1, 3)`
`Point(1, 5)`

**A hash table with 2 buckets has bucket indices 0 and 1. Point(1, 2) has a hash code of 2 + 1 = 3 so it has a bucket index of 1. Point(2, 1) shares the same hash code so it also has the same bucket index. Point(1, 4) has the hash code 5 which also shares the same bucket index.**

b) Does there exist a point that is guaranteed to collide with Point(1, 2) for any number of buckets?

(**X**) Yes
( ) No
( ) More Information is Needed

If yes, mark all of the points that are guaranteed to collide.

```
Point(1, 1)
Point(3, 1)
Point(1, 4)
Point(2, 1) GUARANTEED COLLISION
Point(1, 3)
Point(1, 5)
```

**The only time Points will be guaranteed to collide with other points on any choice of buckets is when their hash codes are exactly the same. From the example above, we know that Point(2, 1) shares the same hash code as Point(1, 2).**

## 3.2 Properties of Hash Maps

One of the important properties for a hash function is for it to be **deterministic**: giving the function the same input should produce the same output.

a) Describe an implementation of the hashCode function for the Point class that is NOT deterministic. You do not need to write code (describe at a high level), and you may assume you have access to any external libraries you want to use.

**A non-deterministic hashCode implementation could simply return a random integer.**

b) Suppose we have a Hash Map of Point objects with a non-deterministic hashCode method. Give an example series of method calls on that map that could give incorrect behavior. You do not need to justify your answer in this part.

**put(5)**
**get(5)**

c) Explain *why* a non-deterministic hash function could break the Hash Map. What role does a hash function play in the implementation of a Hash Map, and why would inconsistent results prevent that behavior?

**The role of a hash function in a Hash Map is to characterize a key in such a way that the data structure can determine what bucket it is stored in without having to search any elements of the map to do so (afterward, the data structure iterates**

**through that bucket to find the element itself). If a hash function is non-deterministic, it cannot be used to determine the bucket for a key because a different bucket could be determined when an element was stored vs. when an element was looked up.**

# 4. Recursive Code Analysis

## 4.1 Recursive Code Modeling

Consider the following recurrence relation, where c is some arbitrary constant.

$$T(n) = \begin{cases} c & \text{if } n=0 \\ 5T(n/3) + n^2 & \text{otherwise} \end{cases}$$

Complete the following Java code so that the above recurrence models its **runtime** in terms of n.

```
public void fun(int n) {
      if ( n == 0 ) {
            System.out.println("fun");
      } else {
            for (int i = 0; i < n; i++) {
                  for (int j = 0; j < n; j++) {
                        System.out.println("very ");
                  }
            }

            for (int i = 0; i < 5; i++) {
                  fun(n / 3);
            }
      }
}
```

# 5. AVL Trees

## 5.1 AVL Insertions

Give an ordering of the keys: 1, 2, 3, 4, 5, 6, 7 such that inserting those keys into an empty AVL tree will cause no rotations.

**One possibility is: 4, 2, 6, 1, 3, 5, 7. There are other possible answers.**

## 5.2 Properties of AVL Trees

For each of the following statements:

- State whether the statement is true or false
- If the statement is false, briefly justify your answer (~1-2 sentences).

a) If you're inserting a **new** key into an AVL tree, the best-case runtime is O(1).

**False. We have to create a new node, which always happens below a leaf node. All leaf nodes are distance Ω(log n) from the root, hence the running time is at least Ω(log n).**

b) The **maximum** number of rotations on a single insert in an AVL tree is Θ(1).

**True. (The maximum is 2 rotations, but that is still a constant).**

c) It is possible to insert a value into an AVL tree such that no rotations can fix the AVL invariant.

**False. As long as the AVL invariant held before the insertion and the insertion is made following the rules of the BST invariant, at most two rotations will be needed to restore the AVL invariant because a single insertion can only imbalance the height of a node by at most one.**