Algorithmic Thinking in Problem Solving
Exam 1

Instructions:

Part A – 70 points

Solve all problems, compress your files in a single zip file, and upload it to Blackboard. Upload the compressed file before the class period ends. You are free to use Java or Python3 (but not a mix of the two). Feel free to run "compute grade" as often as you want to as you take the exam to know if your solutions are correct.

There will be another Blackboard assignment for you to re-submit your solutions before the end of the day to receive partial credit in case you could not finish, or your solution(s) need more work. You can expect at least 50% of partial credit – this number may increase depending on how the class does.

Part B – 30 points

Select **one** of the problems, and record a video where you go over the following:

- Explain your code – Trace it using one concrete instance of the problem – Have a visual representation of it, and explain how your code works using the concrete instance as input
- State the time and (auxiliary) space complexities of your algorithm
- State what test cases you would use to test your solution. You don't have to trace your code on these test cases, just verbally state what test cases you would write and justify why you think those test cases are needed.
- Verbally state that ALL of your exam solutions are your own, and that you DID NOT use any external resources to solve these problems.

The shorter the video, the better. Please do not go over 15 minutes.

Due Dates:

Exam 1 – Wed, Nov 11, 1:29PM

Exam 1 (Re-submission) – Wed, Nov 11, 11:59PM

Video – Monday, Nov 16, 11:59PM

[14 / 70] Problem 1 - Majority Element

Given an array of size *n*, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times in the array.

You may assume that the array is non-empty and the majority element always exists in the array.

Example 1 - Input: [3,2,3] Output: 3

Example 2 - Input: [2,2,1,1,1,2,2] Output: 2

[14 / 70] Problem 2 - Single Number

Given a non-empty array of integers, every element appears twice except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Extra Space complexity can be O(n) or O(1)

Example 1 - Input: [2,2,1] Output: 1

Example 2 - Input: [4,1,2,1,2] Output: 4

[14 / 70] Problem 3 - Climbing Stairs

You are climbing a staircase. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Note: Given n will be a positive integer.

Example 1- Input: 2 Output: 2

Explanation - There are two ways to climb to the top. 1. 1 step + 1 step
2. 2 steps

Example 2- Input: 3 Output: 3

Explanation - There are three ways to climb to the top.

1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

[14 / 70] Problem 4 - House Robber

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

Example 1:
Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

Example 2:
Input: [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob = 2 + 9 + 1 = 12.

[14 / 70] Problem 5 - Best Time to Buy and Sell Stock

Say you have an array for which the ith element is the price of a given stock on day i.

If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Note that you cannot sell a stock before you buy one. Example 1:

Input: [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Not 7-1 = 6, as selling price needs to be larger than buying price. Example 2:

Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.