6️⃣

HW6

▼ **Question 1**

[**10 points**] Suppose you have a rod of length N, and you want to cut up the rod and sell the pieces in a way that maximizes the total amount of money you get. A piece of length i is worth p_i dollars. Devise a **Dynamic Programming** algorithm to determine the maximum amount of money you can get by cutting the rod strategically and selling the cut pieces.

Solution 1

Create a $dp[1..N]$ of length N to store maximum profit at each index i.

Subproblem -

💡 What is the maximum profit that can be made when the rod is of length i

Naturally, maximum profit = 0 when the length of the rod is 0. So, $dp[0] = 0$

```
Let dp[0..N] be the DP array to store the maximum profit at each index i
dp[0] = 0
for j = 1 to N do
  max = Integer.MIN_VALUE;
  for int i = 1 to j do
    max = Math.max(max, p[i] - dp[j - i])
  end for
end for
return dp[N]
```

Time Complexity = $O(n^2)$ because of the 2 for nested for loops

▼ **Question 2**

[**10 points**] Tommy and Bruiny are playing a turn-based game together. This game involves N marbles placed in a row. The marbles are numbered 1 to N from the left to the right. Marble i has a positive value m_i. On each player's turn, they can remove either the leftmost marble or the rightmost marble from the row and receive points equal to the sum of the remaining marbles' values in the row. The winner is the one with the higher score when there are no marbles left to remove.

Tommy always goes first in this game. Both players wish to maximize their score by the end of the game.

Assuming that both players play optimally, devise a **Dynamic Programming** algorithm to return the difference in Tommy and Bruiny's score once the game has been played for any given input.

Your algorithm **must** run in $O(N^2)$ time.

Solution 2

First, we will calculate the prefix sum of the given array so as to calculate the sum between any 2 indices, i and j in constant time i.e. $O(1)$

$OPT_{(i,j)} = max(score-if-left-picked, score-if-right-picked)$. This is for the player whose turn it is to play the game.

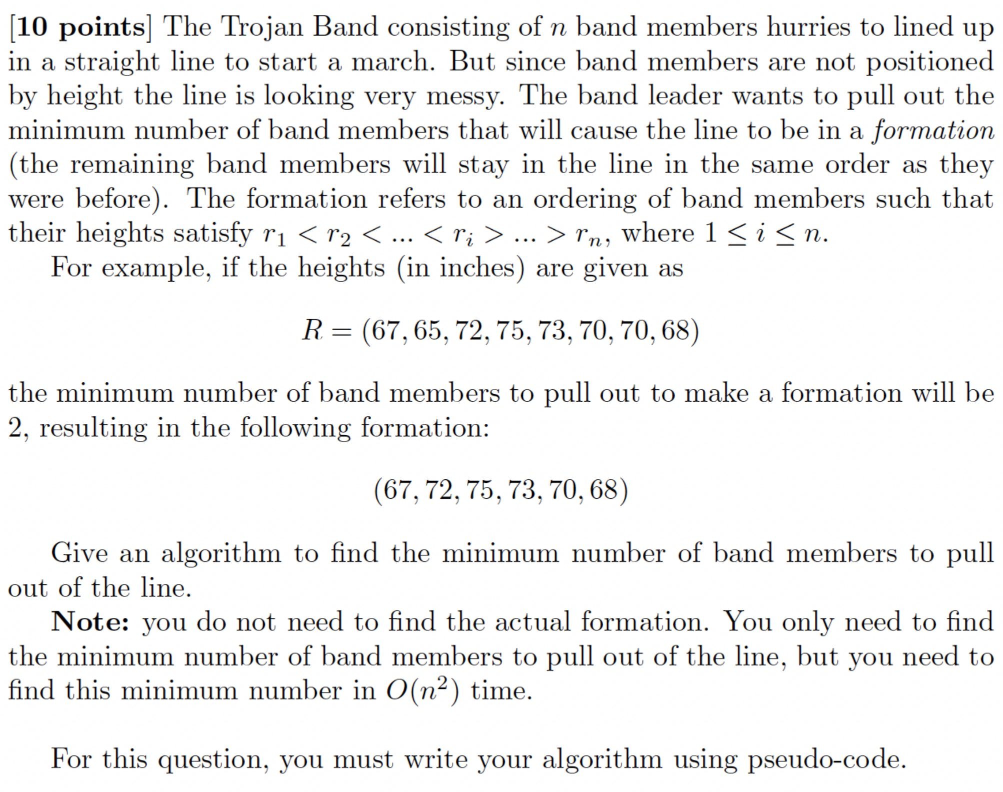
```
Let N be the number of marbles
Let PrefixSum[N + 1] be the prefix_sum of the array that has been calculated
OPT(i, j) = n * n array to store the answer for each subproblem

for i = n - 2 to 0 do
  for j = i + 1 to N - 1 do
    score_if_left_picked = prefixSum(j + 1) - prefixSum(i + 1) - OPT(i + 1, j)
    score_if_right_picked = prefixSum(j) - prefixSum(i) - OPT(i, j - 1)
    OPT(i, j) = max(score_if_left_picked, score_if_right_picked);
  end for
end for
return OPT(0, n - 1)
```

Time Complexity Analysis

$O(n^2)$ for nested for loops and $O(n^2)$ for 2D array extra space

▼ **Question 3**



[**10 points**] The Trojan Band consisting of n band members hurries to lined up in a straight line to start a march. But since band members are not positioned by height the line is looking very messy. The band leader wants to pull out the minimum number of band members that will cause the line to be in a *formation* (the remaining band members will stay in the line in the same order as they were before). The formation refers to an ordering of band members such that their heights satisfy $r_1 < r_2 < ... < r_i > ... > r_n$, where $1 \leq i \leq n$.

For example, if the heights (in inches) are given as

$$R = (67, 65, 72, 75, 73, 70, 70, 68)$$

the minimum number of band members to pull out to make a formation will be 2, resulting in the following formation:

$$(67, 72, 75, 73, 70, 68)$$

Give an algorithm to find the minimum number of band members to pull out of the line.

Note: you do not need to find the actual formation. You only need to find the minimum number of band members to pull out of the line, but you need to find this minimum number in $O(n^2)$ time.

For this question, you must write your algorithm using pseudo-code.

Solution 3

This is a classic problem of longest increasing subsequence (LIS) which allows us to find the length of the longest increasing subsequence for elements of a given array. In this case, we will apply LIS twice → once from left to right and next from right to left.

```
Let arr be the heights in the given array
Create dp1[1..N] and dp2[1..N] to store the LIS at every index i for the array as it is and then the array reversed
```

```
for i = 1 to N do
  for j = 0 to i do
    if ((arr[i] > arr[j]) and (dp1[i] <= dp1[j])) {
      dp1[i] = 1 + dp1[j]
    end if
  end for
end for

Now reverse the array and apply the same LIS algorithm again and store LIS in dp2[1..N]
for i = 1 to N do
  for j = 0 to i do
    if ((arr[i] > arr[j]) and (dp2[i] <= dp2[j])) {
      dp2[i] = 1 + dp2[j]
    end if
  end for
end for

Now create actual increasing subsequences using dp1[1..N] and dp2[1..N]
We get the 2 longest increasing subsequences as
67 72 75
and other as
68 70 73 75 (in reverse order)
This will require O(n^2) time

Now, we traverse both these longest subsequences and check which numbers are missing from the original array.
This operation will require O(N) time.

return count(number of elements missing from original array)
```

Time Complexity Analysis

$O(n^2)$ for nested for loop. i.e. $O(n^2) + O(n^2) + O(n^2) + O(n) = O(n^2)$ final time complexity

💡 Question to grader/instructor: This approach also gives the correct answer. Will marks be allotted in the finals/midterms if we don't provide the OPT(i, j) based solution?

▼ Question 4

[**15 points**] You've started a hobby of retail investing into stocks using a mobile app, RogerGood. You magically gained the power to see N days into the future and you can see the prices of one particular stock. Given an array of prices of this particular stock, where $prices[i]$ is the price of a given stock on the ith day, find the maximum profit you can achieve through various buy/sell actions. RogerGood also has a fixed fee per transaction. You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction.

Note: You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

Solution 4

$buy(i)$ is the max profit one can make if we do not have any stock present till this date

$sell(i)$ is the max profit one can make if we have one unit of this stock present in the current date

```
Let n = length of the prices array
buy[1..N + 1] new array initialized to 0
sell[1..N + 1] new array initialized to 0

for i = n to 0 do
```

```
  buy[i] = max(sell[i + 1] - prices[i], buy[i + 1])
  sell[i] = max(buy[i + 1] + prices[i] - fee, sell[i + 1])
end for
return buy[0]
```

Time Complexity Analysis

Since we traverse $prices[1..N]$ only once, time complexity is $O(N)$