Homework 6

Graded

Student

Abhishek Soundalgekar

Total Points

40 / 40 pts

Question 1

Problem 1

10 / 10 pts

- ✔ **– 0 pts** Correct
- **– 2 pts** Incorrect Subproblem
- **– 3 pts** Incorrect Recurrence
- **– 2 pts** Incorrect Pseudo Code
- **– 1 pt** Base Case missing in pseudocode
- **– 1 pt** Incorrect final answer location
- **– 1 pt** Incorrect Time Complexity
- **– 10 pts** Incorrect
- **– 0 pts** 10/10

Question 2

Problem 2

10 / 10 pts

✔ **– 0 pts** Correct

– 2 pts Incorrect Subproblems

– 3 pts Incorrect Recurrence

– 2 pts Incorrect pseudocode structure / Incorrect iteration of nested loops.

– 1 pt Incorrect/Missing Base Case in pseudocode

– 1 pt Incorrect/Missing final answer location in psuedocode

– 1 pt Incorrect/Missing Time Complexity

– 1 pt Miss to use prefix sum for subarray sum calculation.

Question 3

Problem 3

10 / 10 pts

✔ **– 0 pts** Correct

– 1 pt Incorrect code for where final answer could be found

– 2 pts Incorrect subproblem definition

– 3 pts Incorrect recurrence relation

– 2 pts Incorrect pseudo code

– 1 pt Incorrect base case and their values

– 1 pt Incorrect complexity

Question 4

Problem 4

10 / 10 pts

✔ **– 0 pts** Correct

– 3 pts Wrong Recurrence Relation

– 2 pts Pseudocode missing

– 1 pt Wrong Complexity

Question assigned to the following page: 1

CSCI 570 Spring 2025

Homework 6

Name: Abhishek Soundalgekar

USC ID: 2089011000

Question 1

Answer 1

A piece of length i is worth p_i dollars $(1 \leq i \leq N)$

1.1 Subproblems to be solved

For each integer rod length i from (0 to N), where N is the total length of the given rod, we need to calculate the maximum revenue obtainable from a rod of length i by optimally cutting it into pieces. This can be obtained by building up the solution from the smallest rod (length 0) to the full rod of length N.

1.2 Recurrence Relation for the Subproblems:

Let price array be price $[p0, p1, p2, p3 \ldots, pN]$ containing the prices of each cut size i in the array of lengths $[1, 1 \leq i \leq N$

$OPT(i, a)$ = value of the optimal sol^n using a subset of rod lengths $[1, 2 \ldots i]$ with max allowed length of a.

Question assigned to the following page: 1

If i belongs to the solution

$$OPT(i, a) = price[i] + OPT(i, a - length[i])$$

Else,

$$OPT(i, a) = OPT(i-1, a)$$

Where a is the max allowed length

1.3 Pseudocode

Let the array of pieces of rod lengths from [1 ..... N] be price[]
price[i] is the price for some length i where $1 \leq i \leq N$.

Let OPT [0 ..... N] be an array that is memorized to give the value of optimal solution i.e. max revenue.

Base case
Initialize OPT[0] = 0 (for rod of length 0)

Question assigned to the following page: 1

```
for j = 1 to N
    Initialize revenue = -∞
    for i = 1 to j
        revenue = max(revenue, price[i] + OPT[j - i])
    end for
    OPT[j] = revenue
end for
```

OPT[N] is the value of optimal solution, the max revenue obtained

1.c] Time complexity:

The pseudocode above shows the outer loop runs in N time for j and the inner loop runs j times for each value of j from 1 to N. This means the time complexity will be $O(N^2)$

Question assigned to the following page: 2

Question 2
Answer 2

N stones placed in a row, numbered 0 to N-1 from left to right

2.1 Subproblems to be solved :

Let OPT [i][j] be the maximum difference of score which Alice can achieve over Bob for stones in the range of i to j. In every iteration Alice can pick the leftmost or rightmost stone to maximize the score by gaining the sum of the remaining stones.

Therefore, in each subproblem we need to calculate this remaining sum for Alice and Bob and the max difference OPT [i][j] for all values of i and j for stones from 0 to N-1

Question assigned to the following page: 2

2.2 Recurrence Relation:

When the left stone i is removed then we will have $sum(i+1, j)$ for the remaining stones and if right most stone j is removed then we will be left with $sum(i, j-1)$. Therefore, recurrence relation is:

$$OPT[i][j] = \max\left(sum(i+1, j) - OPT[i+1][j],\ sum(i, j-1) - OPT[i][j-1]\right).$$

2.3 Pseudocode

R → Array of stone values, N → Total number of stones.
Precompute prefix sum to get $S(i, j)$ faster.

```
let S[0...N] be an array where S[0] = 0
for i=1 to N:
    S[i] = S[i-1] + R[i-1]    // 0-indexed R

(function to compute sum from index i to j (0 indexed)
function sum(i, j):
    return S[j+1] - S[i]
```

Question assigned to the following page: 2

let OPT [0...N-1][0...N-1] (2D array)

(Base case: when there is only one stone left):
for i from 0 to N-1:
    OPT [i][i] = 0

(Fill OPT table for all subarrays of length 2 to N)
for length from 2 to N:
    for i from 0 to N-length:
        j = i + length - 1
        left_choice = sum(i+1, j) - OPT [i+1][j]
        right_choice = sum(i, j-1) - OPT [i][j-1]
        OPT [i][j] = max (left_choice, right_choice)
    end for
end for
return OPT [0][N-1]
(The optimal solution value ie the max score difference is found at OPT [0][N-1]

2.4 Time complexity
while calculating the array values the length is ranging from 2 to N in the outer loop & the inner loop when length is less, i value goes up till N-2 which means that combining the time the algorithm runs in $O(N^2)$ time. The DP table is of size $O(N^2)$ and each entry is computed in constant time. $\therefore O(N^2)$

Question assigned to the following page: 3

Question 3
Answer 3

3.1 Subproblems to be solved.

We are required to remove minimum number of people from the line whose heights are lesser to get sequence as:
$a_0 < a_1 < \ldots < a_i > a_{i+1} > \ldots a_k$ for k members where $k \leq n$.

Therefore, we would need to find the sum of longest increasing and decreasing subsequences for each i from 0 to n which will give us the minimum number of people that will need to be removed for some position i by subtracting from n.

3.2 Recurrence Relation :

max(LIS[i], 1+LIS[j])

LIS till i $LIS[i] = 1 + \max(LIS[j] + LIS[j])$
where $j < i$ and $height[j] < height[i]$

LDS till j is given as
max(LDS[i], 1+LDS[j])
$LDS[i] = \max(LDS[j] + LDS[j])$
where $j > i$ and $height[j] < height[i]$

Question assigned to the following page: 3

Base cases:

$LIS[i] = 1$ for all i.

$LDS[i] = 1$ for all i.

3.3 Pseudocode

(n will also be 0 here)

```
Base cases: Initialize LIS[n] = LDS[n] = 1
for i = 1 to n-1
    for j = 0 to i-1
        if height[i] > height[j]
            LIS[i] = max(LIS[i], 1 + LIS[j])
    end for
end for

for i = n-2 to 0
    for j = n-1 to i+1
        if height[i] > height[j]
            LDS[i] = max(LDS[i], 1 + LDS[j])
    end for
end for
max_sequence = 0
for i = 0 to N-1
    max_sequence = max(max_sequence, LIS[i] + LDS[i] - 1)
end for
the sol^n is found as n - max_sequence
```

Question assigned to the following page: 3

3.4 Time complexity :

For the calculation of longest increasing and decreasing subsequence we run the outer loop and inner loop n times for each position from 0 to n-1 which means that the algorithm runs in $O(n^2)$ time.

Space Complexity:

LIS & LDS are arrays each require O(n) total space complexity O(n).

Question assigned to the following page: 4

Question 4
Answer 4

n different types of coins.

4.1 Subproblems to be solved:

We have to maximise the value obtained by each of the coin types from 1 to n for given weight limit W.

OPT [w] will store the value of each coin

4.2 Recurrence Relation:
The recurrence relation for OPT [w] is determined by considering whether to include coin i of weight w[i] & having value v[i]

if $w_i \leq w$ then
OPT [i, w] = max (OPT [i-1, w], v[i] + OPT (i-1, w - weight [i]))

else
OPT [i, w] = OPT [i-1, w]

Question assigned to the following page: 4

4.3] Pseudocode

Base Case:

We initialize the array OPT[0,w] = 0 for each w = 0 to W.

If there is no weight to fill then OPT[i,0] = 0, for all values of i.

```
for i = 1 to n
    for w = 0 to W
        if weight[i] ≤ w
            OPT[i,w] = max(OPT[i-1,w], V[i] +
```

else OPT[i,w] = OPT[i-1,w] OPT[i-1, w - weight[i]])

```
    end for
end for
```

The optimal value will be found at OPT[n,W].

4.4. Time Complexity

The outer loop runs in n iterations & the inner loop runs in W iterations. Therefore, the overall time complexity is O(nW).

Space Complexity

OPT array is of size nxw so space complexity is O(nw).