FINAL PROJECT

Design and Analysis of Algorithms

Department of Computer Science

Submitted by: Umer Ghafoor, Moaz Sami
Roll No.: 22I-2328, 22i-2372

Problem - 1

Pseudo Code

```
count_valid_structures(num_blocks)
    d_array[num_blocks + 1] = 0
    d_array[0] = 1
    for i = 1 to num_blocks:
        for j = num_blocks down to i:
            d_array[j] += d_array[j - i]
    return d_array[num_blocks] -1
```

Asymptotic Analysis

Time Complexity

- Initialization : O(1)
- Loops :
  - Outer Loop : O(n)
  - Inner Loop : O(n/2)
  - Total Loop Time : O(n*n/2) = O(n^2)
- Return Statement : O(1)

Final Time Complexity = O(n^2)

Space Complexity

O(n) since an array of size **n+1** is used

Explanation

The function calculates the number of valid structures using dynamic programming, where **dp[i]** represents the number of valid structures with **i** blocks. It iterates through the blocks and updates **dp[j]** using the previous counts.

Finally, it returns dp[num_blocks] - 1 as the total count of valid structures.

Problem - 2

Pseudo Code

calc_optimal_position(strategic_values, num_attacks):

size = **length** of **strategic_values**

Initialize **dp[size+1][(num_attacks + 2] = infinity**

Set **dp[0][0] = 0**

For **i** from **1 to size**:

For **k** from **0 to (i - 1)**:

For **j** from **1 to min(i, num_attacks + 1)**:

If **dp[k][j - 1]** is infinity, skip this iteration

Set **local_cost = 0**

For **m** from **k to (i - 1)**:

For **n** from **(m + 1) to (i - 1)**:

Add **strategic_values[m]** * **strategic_values[n]** to **local_cost**

Update **dp[i][j]** to **minimum of (dp[i][j], dp[k][j - 1] + local_cost)**

Return **dp[size][num_attacks + 1]**

Asymptotic Analysis

Time Complexity

n = size of array

k = number of attacks

- Initialization: **O(n^k)**
- Middle Loops
  - First Loop (i): O(n)
  - Second Loop (k): O(n/2)
  - Third Loop (j): O(min(i, k+ 1)) = O(n)
  - Inner Nested Loops (m, n): O(size^2) (worst case when i=max & m=0)

    Total of Middle Loops = **O(n)** X **O(n/2)** X **O(n^2)** X **O(m)=O(k*(n^4)/2)**

- Return Statement : O(1)

Final Time Complexity **= O(k*n^4)**

Space Complexity

O(n*k) since an array of size **n*k** is used

Explanation

The function calculates the optimal position to place attacks based on strategic values and the number of attacks allowed.Using dynamic programming, It iterates through each possible position and number of attacks, calculating the cost of each attack and updating the dynamic programming array with the minimum cost. Finally, it returns the min cost for the given parameters.

Problem 3

Pseudo Code

1. Start
2. Input the size of the array (n)
3. Allocate memory for two arrays arr1 and arr of size n
4. Input elements for the arr1 array
5. Copy elements from arr1 to arr
6. Rearrange the elements in arr using min-heap approach
7. Output the rearranged array arr
8. Initialize m1, m2, syapa1, and syapa2 variables
9. Loop three times:
   a. Loop through the array arr:
      i. If the current element is equal to m1 and m2 is 0, increment m1
      ii. If the current element is equal to m1, update m1 and reset m2
      iii. If the current element is equal to m1 + m2 + 1, increment m2
      iv. Update syapa1 and syapa2 accordingly
      v. If the current element is less than its previous element, swap them
10. Initialize index as -1 and ck as 0
11. Loop through the original array arr1:
   a. If index is -1 and the current element is m1 - 1, m1 + 1, or syapa2:
      i. Update index to the current index
   b. If index is not -1 and the current element is m1 - 1, m1 + 1, or syapa2:
      i. Set ck as 1
      ii. Output 2 and the range of indexes where the elements are found
      iii. Break the loop
12. If ck is still 0, set m1 as -1 and output -1
13. Output the value of MNPN (m1)

14. End

Asymptotic Analysis

Time complexity:

n(for taking input) + n(for copying array) + n(bottom up heap tree) + 3n(finding the MNPN) + n(for iterating through original array to find indexes)

=> 7n

O(n)

Space Complexity

O(n) since it is using array of size N

Problem 4

Pseudo Code

1. Start
2. Input the number of rows (n) and columns (m)
3. Allocate memory for a 2D array M of size n x m
4. Input elements for the array M
5. Allocate memory for another 2D array A of size n x m
6. Initialize variables cm1, pm1, cm2, pm2, ck, and pk
7. Loop through each column in reverse order (from m-1 to 0):
   a. Update pm1, pm2, and pk
   b. Reset cm1, cm2, and ck
   c. Loop through each row in the current column:
      i. Determine the value for A[j][i] based on conditions provided
      ii. Update cm1, cm2, and ck accordingly
   d. After processing the entire column, move to the next column
8. Output the resulting array A
9. Output the value of cm1
10. End

Asymptotic Analysis

Time Complexity:

n*m(for taking input) + n*m(for putting/adding values in the matrix)

=>2(n*m)

O(n*m)

Space Complexity

O(n*m) since it is using single array of size n X m