Lab 10

Problem 1

You are tasked with helping a maintenance worker move between floors in a building. Each floor of the building has a cost associated with moving to it. The worker can either go up one floor at a time or skip one floor and go up two floors at once.

You need to determine the minimum cost for the worker to reach the top of the building, given that they can either start at the first floor or the second floor.

Input:

- A list `cost` of integers, where `cost[i]` is the cost for moving to floor `i`. The worker can either move one floor up (paying the cost of the current floor) or skip one floor and go two floors up (paying the cost of the skipped floor and the next one).

Output:

- The minimum total cost for the worker to reach the top of the building.

Task:

1. Explain how you would use dynamic programming to calculate the minimum cost for the worker to reach the top of the building.
2. With the given example `cost = [1,100,1,1,1,100,1,1,100,1]`, explain step-by-step how the dynamic programming approach works to calculate the minimum cost of reaching the top.

Problem 2

You are working on a network monitoring tool that tracks the number of active devices in a network at any given time. Each device in the network has a unique identifier, and the identifiers are represented as integers. These identifiers can be seen as a sequence of numbers, where each number's binary representation shows which devices are active at specific time intervals.

Your task is to write a function that, for each integer from `0` to `n`, returns the number of devices that are active, which corresponds to the number of `1`s in the binary representation of each number.

The function should generate a list where each index `i` (from `0` to `n`) tells you how many devices are active (i.e., how many `1`s are in the binary representation of `i`).

Input:

- An integer `n` which represents the highest device identifier you want to track.

Output:

- A list `ans` of length `n + 1`, where `ans[i]` is the number of active devices (the number of `1`s) in the binary representation of the number `i`.

Example:

Input: `n = 2`

Output: `[0, 1, 1]`

Explanation:

- The binary representation of 0 is `0`, which has 0 `1`s (no active devices).
- The binary representation of 1 is `1`, which has 1 `1` (one active device).
- The binary representation of 2 is `10`, which has 1 `1` (one active device).

Task:

1. How can you solve this problem efficiently using dynamic programming that runs in **O(n)** time?
2. For the given example `n = 5`, explain how you would calculate the binary representations of each number from `0` to `5` and count the number of `1`s in each binary number to build the final output array.

Problem 3

Imagine you are programming a robot to move across a factory floor. The floor is divided into checkpoints, and each checkpoint indicates how far the robot can jump forward. The robot starts at the first checkpoint and aims to reach the last one.

Each checkpoint has a number that represents the **maximum steps** the robot can jump from there. The robot can jump from 1 up to the maximum number of steps allowed at each checkpoint. You need to determine if the robot can reach the last checkpoint starting from the first one.

Input:

- A list `nums` of integers where `nums[i]` is the maximum number of steps the robot can take from checkpoint `i`.

Output:

- A boolean value: `True` if the robot can reach the last checkpoint, otherwise `False`.

Task:

1. Explain how you would solve this problem using dynamic programming.
2. With the given examples `nums = [2, 3, 1, 1, 4]` and `nums = [3, 2, 1, 0, 4]`, explain step-by-step how the approach works to determine if the robot can reach the last checkpoint.

Problem 4

You are managing a rectangular garden represented as a 2D grid. Each cell in the grid is either:

- `1` (indicating that the plot is **healthy and green**), or
- `0` (indicating a **damaged or unusable** plot).

Your goal is to find out how many **square plots** (subgrids) of any size can be formed such that **all cells in the square are green (1)**.

A square plot must be fully green — all cells in it must be 1s — and can be of any size `1×1`, `2×2`, ..., up to the size that fits within the garden.

Example:

Input:

```
garden = [
  [0, 1, 1, 1],
  [1, 1, 1, 1],
  [0, 1, 1, 1]
]
```

Output: `15`

Explanation:

- There are 10 squares of size 1×1 (each individual 1).
- 4 squares of size 2×2.
- 1 square of size 3×3.
- Total = **10 + 4 + 1 = 15**

Task:

1. How would you solve this problem using dynamic programming?
2. Use the example `garden = [[1, 0, 1], [1, 1, 0], [1, 1, 0]]` to walk through how the DP table is built step-by-step and how the final result is calculated.

Problem 5

You are working on a video compression algorithm. A video is broken into `n` frames, and each frame has a **complexity score** represented by an integer in an array `frames`. To compress the video, you can **group contiguous frames** into segments, with each segment being **at most k frames long**.

For every segment you create, the compression algorithm simplifies all frames in that segment by assigning them the **maximum complexity** within that segment (since the hardest frame to compress defines the difficulty of the segment). The **total cost** of compression is the **sum of all new frame values after partitioning**.

Your task is to find the **maximum total cost** you can get by optimally partitioning the video into segments of length at most `k`.

Example

Input:

```
frames = [1, 15, 7, 9, 2, 5, 10]
k = 3
```

Output: `84`

Explanation:

- One optimal way to partition is: `[15,15,15] [9] [10,10,10]`
- Sum = 15+15+15 + 9 + 10+10+10 = **84**

Task:

1. Explain how you would solve this problem using dynamic programming in O(nk) time
2. Use the input `frames = [1, 4, 1, 5, 7, 3, 6, 1, 9, 9, 3], k = 4` to walk through how your approach calculates the result step by step.