NAME: Jackline Mboya

ADM No: 193670

Week 7 Assignment - Fibonacci

Unit Code: DSA 8302

Unit Name: Computational Techniques in Data Science

Question 1: Fibonacci Problem - Plant Branch Growth

A rare plant grows following a Fibonacci-like pattern. In the first cycle, it has 1 branch. In the second cycle, it grows 1 more. From the third cycle onward, the number of new branches equals the total number of branches in the previous two cycles. Task: How many branches will the plant have after 12 growth cycles?

Hint: Model this as a Fibonacci sequence:

Cycle 1 → 1 branch

Cycle 2 → 1 branch

Cycle 3 → 2 branches

Cycle 4 → 3 branches ...

Find the total number of branches at cycle 12.

In [1]:
```
# Fibonacci
def fibonacci(n):
    a, b = 1, 1
    for _ in range(3, n + 1):
        a, b = b, a + b
    return b if n > 1 else a
```

```
branches_after_12_cycles = fibonacci(12)
print("Branches after 12 cycles:", branches_after_12_cycles)
```

```
Branches after 12 cycles: 144
```

The total number of branches after cycle 12 is 144 branches

In [ ]:

Question 2: LIS Problem - Student Score Trends

A student's test scores over a semester are recorded as: [72, 74, 69, 78, 80, 81, 75, 85, 88, 70, 92]

Task:

Determine the longest consecutive sequence of scores where each score is higher than the last one (i.e., a strictly increasing subsequence).

What is the length of this increasing trend?

In [7]:

```
def longest_increasing_subsequence(scores):
    n = len(scores)
    dp = [1] * n
    prev = [-1] * n  # To reconstruct path

    for i in range(n):
        for j in range(i):
            if scores[i] > scores[j] and dp[j] + 1 > dp[i]:
                dp[i] = dp[j] + 1
                prev[i] = j

    # Find index of max value in dp
    max_len = max(dp)
    idx = dp.index(max_len)

    # Reconstruct LIS
    lis = []
    while idx != -1:
        lis.append(scores[idx])
```

```
        idx = prev[idx]
    lis.reverse()

    return max_len, lis

scores = [72, 74, 69, 78, 80, 81, 75, 85, 88, 70, 92]
lis_length, lis_sequence = longest_increasing_subsequence(scores)
print("Length of longest increasing subsequence:", lis_length)
print("Actual increasing subsequence:", lis_sequence)
```

```
Length of longest increasing subsequence: 8
Actual increasing subsequence: [72, 74, 78, 80, 81, 85, 88, 92]
```

Longest consecutive sequence of scores = [72, 74, 78, 80, 81, 85, 88, 92]

The length of longest increasing subsequence of scores is 8

In [ ]:

Question 3: Knapsack Problem - Server CPU Allocation

Problem

You manage a server with **30 CPU units** available. There are **7 tasks**, each requiring a specific number of CPU units and offering a corresponding reward in user satisfaction:

Task	CPU Units	Reward
A	5	30
B	10	40
C	3	20
D	8	50
E	7	45
F	4	25

Task	CPU Units	Reward
G	6	35

Task

Select a combination of tasks to **maximize total reward** without exceeding the **30 CPU unit limit**.

In [3]:
```python
def knapsack(cpu_limit, tasks):
    n = len(tasks)
    dp = [[0] * (cpu_limit + 1) for _ in range(n + 1)]

    #Iterate over each task
    for i in range(1, n + 1):
        cpu, reward = tasks[i - 1]
        for w in range(cpu_limit + 1):
            if cpu <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - cpu] + reward)
            else:
                dp[i][w] = dp[i - 1][w]

    # Backtrack to find selected tasks
    w = cpu_limit
    selected = []
    for i in range(n, 0, -1):
        if dp[i][w] != dp[i - 1][w]:
            cpu, reward = tasks[i - 1]
            selected.append(chr(ord('A') + i - 1))
            w -= cpu

    return dp[n][cpu_limit], selected[::-1]

# Task list as (CPU usage, Reward)
tasks = [(5, 30), (10, 40), (3, 20), (8, 50), (7, 45), (4, 25), (6, 35)]
cpu_limit = 30
max_reward, chosen_tasks = knapsack(cpu_limit, tasks)

print("Max Reward:", max_reward)
print("Tasks Selected:", chosen_tasks)
```

```
Max Reward: 185
Tasks Selected: ['A', 'D', 'E', 'F', 'G']
```

A Combination of tasks include ['A', 'D', 'E', 'F', 'G'] that can maximize total reward of 185 without exceeding 30 CPU unit limit.

In [ ]: