Experiment **#11**		Student ID	
Date		Student Name	[@KLWKS_BOT THANOS]

Experiment Title: Implementation of Programs on Dynamic Programming - III.

Aim/Objective: To understand the concept and implementation of Basic programs on Dynamic Programming.

Description: The students will understand and able to implement programs on Dynamic Programming.

Pre-Requisites:

Knowledge: Dynamic Programming in C/C++/Python Tools: Code Blocks/Eclipse IDE

Pre-Lab:

Given a set of positive integers, determine if there exists a subset whose sum equals a given value.

Example:

Input:

Set: [3, 34, 4, 12, 5, 2]

Target Sum: 9

Output:

Subset exists: Yes

- **Procedure/Program:**

```
#include <stdio.h>
#include <stdbool.h>

bool isSubsetSum(int arr[], int n, int target_sum) {
   bool dp[target_sum + 1];
   for (int i = 0; i <= target_sum; i++) {
      dp[i] = false;
   }

   dp[0] = true;

   for (int i = 0; i < n; i++) {
```

```
        for (int j = target_sum; j >= arr[i]; j--) {
            dp[j] = dp[j] || dp[j - arr[i]];
        }
    }

    return dp[target_sum];
}

int main() {
    int arr[] = {3, 34, 4, 12, 5, 2};
    int target_sum = 9;
    int n = sizeof(arr) / sizeof(arr[0]);

    if (isSubsetSum(arr, n, target_sum)) {
        printf("Subset exists: Yes\n");
    } else {
        printf("Subset exists: No\n");
    }

    return 0;
}
```

- **Data and Results:**

Data: Set: [3, 34, 4, 12, 5, 2], Target Sum: 9

Result: Subset exists: Yes. Subset sum of 9 can be achieved.

- **Analysis and Inferences:**

Analysis: Dynamic programming approach ensures optimal solution with reduced time complexity.

Inferences: Subset sum problem is efficiently solved using dynamic programming method.

In-Lab:

In KL University, all students have to participate regularly in Sports. There is a different Sports activity each day, and each activity has its own duration. The Sports schedule for the next term has been announced, including information about the number of minutes taken by each activity. Sreedhar has been designated Sports coordinator. His task is to assign Sports duties to students, including himself. The University rules say that no student can go three days in a row without any Sports duty. Sreedhar wants to find an assignment of Sport duty for himself that minimizes the number of minutes he spends overall on Sports.

Input format

Line 1: A single integer N, the number of days in the future for which Sports data is available.

Line 2: N non-negative integers, where the integer in position i represents the number of minutes required for Sport work on day i.

Output format

The output consists of a single non-negative integer, the minimum number of minutes that Sreedhar needs to spend on Sports duties this term

Constraint:

There is only one subtask worth 100 marks. In all inputs:

• $1 \leq N \leq 2 \times 105$

• The number of minutes of Sports each day is between 0 and 104, inclusive.

Example:

Input	**Output**
10 3 2 1 1 2 3 1 3 2 1	4

- **Procedure/Program:**

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_N 200000

int main() {
   int N;
   scanf("%d", &N);

   int minutes[N];
   for (int i = 0; i < N; i++) {
      scanf("%d", &minutes[i]);
   }

   int dp[N];
   dp[0] = minutes[0];
   if (N > 1) {
      dp[1] = minutes[0] + minutes[1];
   }

   for (int i = 2; i < N; i++) {
      dp[i] = (dp[i-1] + minutes[i] < dp[i-2] + minutes[i-1] + minutes[i])
            ? dp[i-1] + minutes[i]
            : dp[i-2] + minutes[i-1] + minutes[i];
   }

   printf("%d\n", dp[N-1]);

   return 0;
}
```

- **Data and Results:**

Data:

Input consists of number of days and corresponding minutes data.

Result:

Minimum time Sreedhar spends on sports duties is 4 minutes.

- **Analysis and Inferences:**

Analysis:

The algorithm computes optimal schedule minimizing time with dynamic programming.

Inferences:

Sreedhar can minimize duty time using optimal assignment with constraints.

Post-Lab:

In IPL 2025, the amount that each player is paid varies from match to match. The match fee depends on the quality of opposition, the venue etc. The match fees for each match in the new season have been announced in advance. Each team has to enforce a mandatory rotation policy so that no player ever plays three matches in a row during the season. Nikhil is the captain and chooses the team for each match. He wants to allocate a playing schedule for himself to maximize his earnings through match fees during the season.

Input format

Line 1: A single integer N, the number of games in the IPL season.

Line 2: N non-negative integers, where the integer in position i represents the fee for match i.

Output format

The output consists of a single non-negative integer, the maximum amount of money that Nikhil can earn during this IPL season.

Test data

There is only one subtask worth 100 marks. In all inputs:

• $1 \leq N \leq 2 \times 105$

• The fee for each match is between 0 and 104, inclusive.

Sample1:

Input	**Output**
5 10 3 5 7 3	23

Explanation: 10+3+7+3

Sample 2:

Input	**Output**
8 3 2 3 2 3 5 1 3	17

Explanation: 3+3+3+5+3

- **Procedure/Program:**

```
#include <stdio.h>
#include <stdlib.h>

int max_earnings(int N, int fee[]) {
   if (N == 1) return fee[0];
   if (N == 2) return fee[0] + fee[1];

   int dp[N];
   dp[0] = fee[0];
   dp[1] = fee[0] + fee[1];
   dp[2] = (fee[0] + fee[1] > fee[1] + fee[2]) ? (fee[0] + fee[1]) : (fee[1] + fee[2]);
   dp[2] = (dp[2] > fee[0] + fee[2]) ? dp[2] : (fee[0] + fee[2]);

   for (int i = 3; i < N; i++) {
      dp[i] = dp[i-1];
      if (dp[i-2] + fee[i] > dp[i]) dp[i] = dp[i-2] + fee[i];
      if (dp[i-3] + fee[i] + fee[i-1] > dp[i]) dp[i] = dp[i-3] + fee[i] + fee[i-1];
   }

   return dp[N-1];
}

int main() {
   int N;
   scanf("%d", &N);

   int fee[N];
   for (int i = 0; i < N; i++) {
      scanf("%d", &fee[i]);
   }

   printf("%d\n", max_earnings(N, fee));
   return 0;
}
```

- **Data and Results:**

Data: Input: 5 matches with fees 10, 3, 5, 7, 3.

Result: Maximum earnings from matches: 23, considering the rotation policy.

- **Analysis and Inferences:**

Analysis: Dynamic programming optimizes earnings while preventing three consecutive matches.

Inferences: Efficient algorithm with time complexity O(N) for large inputs.

- **Sample VIVA-VOCE Questions:**

1. Explain the concept of overlapping sub problems in Dynamic Programming

Overlapping Subproblems: When the same smaller problems are solved repeatedly in Dynamic Programming.

2. What is the difference between top-down and bottom-up approaches in Dynamic Programming?

Top-down vs Bottom-up:

- **Top-down**: Solves problems using recursion and stores results (memoization).
- **Bottom-up**: Builds solutions step by step from the base cases.

3. What is Dynamic Programming, and what are its key characteristics?

Dynamic Programming: A method to solve problems by breaking them into smaller problems, saving results to avoid repeated work.

4. Describe the two essential ingredients of a problem that can be solved using Dynamic Programming.

Essential Ingredients:

- **Optimal Substructure**: The best solution comes from combining the best solutions to smaller problems.
- **Overlapping Subproblems**: The problem has subproblems that repeat and need to be solved multiple times.

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.