**LEBANESE AMERICAN UNIVERSITY**

**School of Arts and Science**

**Department of Computer Science and Mathematics**

**CSC 310: Algorithms and Data Structures**



# Lab X

**Important Note:** Input should be read from a file named after the problem name. For example, problem 1 has the input file “p1.in”.

**Problem 1**

A table composed of **N x M** cells, each having a certain quantity of apples, is given. You start from the upper-left corner. At each step you can go down or right one cell. Find the maximum number of apples you can collect.

**Input**

The first line is an integer *T* representing the number of test cases.

Each test case is made up of a two integer *N* and *M*, then follows *N*x*M* elements follow.

**Output**

For each test case, output the maximum number of apples you can collect.

**Sample Input Sample Output**

2 192

2 7 170

7 12 29 27 31 34 30

18 5 21 18 32 9 22

2 7

28 24 12 27 7 5 23

6 7 20 14 6 30 29

**Problem 2**

Given a rod of length *n* inches and an array of prices that contains prices of all pieces of size smaller than *n*. Determine the maximum value obtainable by cutting up the rod and selling the pieces.d

**Input**

The first line of input is an integer *T* representing the number of test cases.

Each test case is made up of a single integer *n* representing the length of the rod respectively. Then *n* integers follows the cost of each piece where the size of the piece is the index of the array starting at 1.

**Output**

For each test case, output the maximum value obtainable by cutting up the rod.

**Sample Input Sample Output**

1 22

8

1 5 8 9 10 17 17 20

**Problem 3**

The **Longest Increasing Subsequence** (or **LIS**) of a sequence of integers is the longest sorted subsequence found in the sequence.

**Input**

The first line of input is an integer *T* representing the number of test cases.

Each test case is made up of a single integer *N* and *N* integers follow.

**Output**

For each test case, output The length of the longest increasing subsequence**.**

**Sample Input Sample Output**

1 5

9 2 6 3 4 1 2 9 5 8

**Problem 4**

Given a natural number *n*, what is the least number of moves you need to reduce *n* to 1? Valid moves are:

* subtract 1.
* divide by 2, applicable if n is divisible by 2.
* divide by 3, applicable if n is divisible by 3.

Use Dynamic Programming to solve this problem.

**Input**

The first line of input is an integer *T* representing the number of test cases.

Each test case is made up of as single integer *n*.

**Output**

For each test case, output the minimum number of moves to reduce *n* to 1.

**Sample Input Sample Output**

1 3

10

**Problem 5**

Given a natural number *n*, what is the least number of moves you need to reduce *n* to 1? Valid moves are:

* subtract 1.
* divide by 2, applicable if n is divisible by 2.
* divide by 3, applicable if n is divisible by 3.

Use recursive backtracking to solve this problem.

**Input**

The first line of input is an integer *T* representing the number of test cases.

Each test case is made up of as single integer *n*.

**Output**

For each test case, output the minimum number of moves to reduce *n* to 1.

**Sample Input Sample Output**

1 3

10