1. **Minimum Operations**

Given a number N. Find the number of operations required to reach **N** starting from **0**. You have 2 operations available:

* Double the number
* Add one to the number

**Example 1:**

**Input:**

N = 8

**Output:** 4

**Explanation**: 0 + 1 = 1, 1 + 1 = 2,

2 \* 2 = 4, 4 \* 2 = 8

â€‹**Example 2:**

**Input**:

N = 7

**Output:** 5

**Explanation**: 0 + 1 = 1, 1 + 1 = 2,

1 + 2 = 3, 3 \* 2 = 6, 6 + 1 = 7

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **minOperation()**which accepts an integer N and return number of minimum operations required to reach N from 0.

**Expected Time Complexity:**O(LogN)  
**Expected Auxiliary Space:**O(1)

**Constraints:**  
1 <= N <= 106

Code:

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

class Solution

{

public:

int dp[1000001]={-1};

int minOperation(int n)

{

dp[1]=1;

for(int i=2;i<=n;i++)

{

if(i%2==0)

dp[i]=dp[i/2]+1;

else

dp[i]=dp[i-1]+1;

}

//code here.

return dp[n];

}

};

// { Driver Code Starts.

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

Solution ob;

cout<<ob.minOperation(n)<<endl;

}

} // } Driver Code Ends

2. **Longest Common Substring**

Given two strings **X** and **Y**. The task is to find the length of the longest common substring.

**Input:**  
First line of the input contains number of test cases **T**. Each test case consist of three lines, first of which contains 2 space separated integers **N** and **M** denoting the size of string **X** and **Y** strings respectively. The next two lines contains the string **X** and **Y**.

**Output:**  
For each test case print the length of longest  common substring of the two strings .

**Constraints:**  
1 <= T <= 200  
1 <= N, M <= 100

**Example:**  
**Input:**  
2  
6 6  
ABCDGH  
ACDGHR  
3 2  
ABC  
AC

**Output:**  
4  
1

**Example:  
Testcase 1:** CDGH is the longest substring present in both of the strings.

**Code:**

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n,m;

cin>>n>>m;

string s1,s2;

cin>>s1>>s2;

int dp[n+1][m+1];

int result=0;

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

{

for(int j=0;j<=m;j++)

{

if(i==0 || j==0)

dp[i][j]=0;

else if(s1[i-1]==s2[j-1])

{

dp[i][j]=1+dp[i-1][j-1];

result=max(result,dp[i][j]);

}

else

dp[i][j]=0;

}

}

cout<<result<<endl;

}

return 0;

}

3. **Longest Increasing Subsequence**

Given a sequence **A** of size **N**, find the **length**of the**longest increasing subsequence** from a given sequence .  
The longest increasing subsequence means to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible. This subsequence is not necessarily contiguous, or unique.

**Note:** Duplicate numbers are not counted as increasing subsequence.

**Input:**  
The first line contains an integer T, depicting total number of test cases. Then following T lines contains an integer N depicting the size of array and next line followed by the value of array.

**Output:**  
For each testcase, in a new line, print the Max length of the subsequence in a separate line.

**Constraints:**  
1 ≤ T ≤ 100  
0 ≤ N ≤ 1000  
0 ≤ A[i] ≤ 300

**Example:**  
**Input**:  
2  
16  
0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15  
6  
5 8 3 7 9 1

**Output:**  
6  
3

**Explanation:  
Testcase 2:** Longest increasing subsequence is of size 3 with elements (there are many subsequence, but listing one of them): **5 7 9**.

Code:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

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

cin>>arr[i];

int dp[n]={1};

for(int i=1;i<n;i++)

{

dp[i]=1;

for(int j=0;j<i;j++)

{

if(arr[j]<arr[i])

dp[i]=max(dp[i],dp[j]+1);

}

}

int maxn=INT\_MIN;

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

maxn=max(maxn,dp[i]);

cout<<maxn<<endl;

}

return 0;

}

4. **Longest Common Subsequence**

Given two sequences, find the length of longest subsequence present in both of them. Both the strings are of uppercase.

**Input:**  
First line of the input contains no of test cases  **T**,the **T** test cases follow.  
Each test case consist of 2 space separated integers **A** and **B** denoting the size of string **str1** and **str2** respectively  
The next two lines contains the 2 string **str1** and **str2** .

**Output:**  
For each test case print the length of longest  common subsequence of the two strings .

**Constraints:**  
1<=T<=200  
1<=size(str1),size(str2)<=100

**Example:**  
**Input:**  
2  
6 6  
ABCDGH  
AEDFHR  
3 2  
ABC  
AC

**Output:**  
3  
2

**Explanation**  
LCS for input Sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3.

LCS of "ABC" and "AC" is "AC" of length 2

Code:

using namespace std;

//int dp[500][500];

int lcs(string s1,string s2,int i,int j,vector<vector<int>> &dp)

{

if(i==s1.length() || j==s2.length())

return 0;

if(dp[i][j]!=-1)

return dp[i][j];

if(s1[i]==s2[j])

return dp[i][j]=1+lcs(s1,s2,i+1,j+1,dp);

int op1=lcs(s1,s2,i+1,j,dp);

int op2=lcs(s1,s2,i,j+1,dp);

dp[i][j]=max(op1,op2);

return dp[i][j];

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n,m;

cin>>n>>m;

string s1,s2;

vector<vector<int>> dp(n,vector<int>(m,-1));

cin>>s1>>s2;

cout<<lcs(s1,s2,0,0,dp)<<endl;

}

return 0;

}

5. **0 - 1 Knapsack Problem**

You are given weights and values of N items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. Note that we have only one quantity of each item.  
In other words, given two integer arrays val[0..N-1] and wt[0..N-1] which represent values and weights associated with N items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item, or don’t pick it (0-1 property).

Input:  
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case consists of four lines.  
The first line consists of N the number of items.  
The second line consists of W, the maximum capacity of the knapsack.  
In the next line are N space separated positive integers denoting the values of the N items,  
and in the fourth line are N space separated positive integers denoting the weights of the corresponding items.

Output:  
For each testcase, in a new line, print the maximum possible value you can get with the given conditions that you can obtain for each test case in a new line.

Constraints:  
1 ≤ T ≤ 100  
1 ≤ N ≤ 1000  
1 ≤ W ≤ 1000  
1 ≤ wt[i] ≤ 1000  
1 ≤ v[i] ≤ 1000

Example:  
Input:  
2  
3  
4  
1 2 3  
4 5 1  
3  
3  
1 2 3  
4 5 6  
Output:  
3  
0  
Explanation:  
Test Case 1: With W = 4, you can either choose the 0th item or the 2nd item. Thus, the maximum value you can generate is the max of val[0] and val[2], which is equal to 3.  
Test Case 2: With W = 3, there is no item you can choose from the given list as all the items have weight greater than W. Thus, the maximum value you can generate is 0.

Code:

using namespace std;

int maxval(int n,int w,int val[],int wt[])

{

int dp[n+1][w+1];

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

{

for(int j=0;j<=w;j++)

{

if(i==0 || j==0)

dp[i][j]=0;

else if(j>=wt[i-1])

{

dp[i][j]=max(val[i-1]+dp[i-1][j-wt[i-1]],dp[i-1][j]);

}

else

dp[i][j]=dp[i-1][j];

}

}

return dp[n][w];

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n,w;

cin>>n>>w;

int val[n];

int wt[n];

//memset(dp,-1,sizeof(dp));

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

cin>>val[i];

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

cin>>wt[i];

cout<<maxval(n,w,val,wt)<<endl;

}

return 0;

}

6. **Minimum number of jumps**

Given an array of integers where each element represents the max number of steps that can be made forward from that element. The task is to find the minimum number of jumps to reach the end of the array (starting from the first element). If an element is **0**, then cannot move through that element.

**Input:**  
The first line contains an integer T, depicting total number of test cases.  Then following T lines contains a number n denoting the size of the array. Next line contains the sequence of integers *a*1, *a*2, ..., *an.*

**Output:**  
For each testcase, in a new line, print the minimum number of jumps. If answer is not possible print "**-1"(**without quotes**)**.

**Constraints:**  
1 ≤ T ≤ 100  
1 ≤ N ≤ 107  
0 <= ai <= 107

**Example:**  
**Input:**  
2  
11  
1 3 5 8 9 2 6 7 6 8 9  
6  
1 4 3 2 6 7  
**Output:**  
3  
2

**Explanation:  
Testcase 1:** First jump from 1st element, and we jump to 2nd element with value 3. Now, from here we jump to 5h element with value 9. and from here we will jump to last.

Code:

using namespace std;

int minstep(int arr[],int n)

{

int dp[n];

// memset(dp,INT\_MAX,sizeof(dp));

dp[0]=0;

for(int i=1;i<n;i++)

{

dp[i]=INT\_MAX;

for(int j=0;j<i;j++)

{

if(i<=j+arr[j] && dp[j]!=INT\_MAX)

dp[i]=min(dp[i],dp[j]+1);

}

}

if(dp[n-1]!=INT\_MAX)

return dp[n-1];

return -1;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

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

cin>>arr[i];

cout<<minstep(arr,n)<<endl;

}

return 0;

}