0-N Knapsack

You've heard of 0-1 knapsack. Below is the problem statement for the same.

Given weights and values of n items, put these items in a knapsack of capacity ***cap*** to get t
value in the knapsack. In other words, given two integer arrays ***val[0..n-1]*** and ***wt[0..n-1]*** whi
and weights associated with n items respectively. Also given an integer ***cap*** which represent
find out the maximum value subset of ***val[]*** such that sum of the weights of this subset is sm
cap. You cannot break an item, either pick the complete item, or don't pick it (0-1 property).
There is a little twist for 0-N knapsack. You can pick an element more than once. Now you h
value multi subset of ***val[]*** such that sum of the weights of this subset is smaller than or equ

Note: Maximum value subset means the sunset with maximum sum of all the values in subse

Input Format

The first line contains two integers, representing ***n***(size of ***val[]***) and ***cap*** respectively. The sub
n integers representing the ***wt[]*** array. The next line, again, contains ***n*** integers representing

Constraints

1 <= n,cap <= 1000 1 <= wt[i] <= cap 1 <= val[i] <= 100000

Output Format

Print a single integer, the answer to the problem.

Sample Input

```
5 11
3 2 4 5 1
4 3 5 6 1
```

Sample Output

```
16
```

Explanation

We take second item 4 times and fifth item one time to make the total values 16.

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner pavan = new Scanner(System.in);
        int pw = pavan.nextInt();
        int pcap = pavan.nextInt();
        int []pweight = new int[pw];
        int []pvalue = new int[pw];
        for(int i=0; i<pw; i++){
            pweight[i] = pavan.nextInt();
        }

        for(int i=0; i<pw; i++){
            pvalue[i] = pavan.nextInt();
        }

        int [][]dp = new int[pweight.length+1][pcap+1];
        for(int i=0; i<dp.length; i++) {
            Arrays.fill(dp[i], -1);
        }
        System.out.println(Knapsack(pweight, pvalue, pcap,
0, dp));
    }

    public static int Knapsack(int[] wt, int [] val, int
cap, int i, int [][]dp) {
        if(i==wt.length || cap == 0)
            return 0;

        if(dp[i][cap] != -1) {
            return dp[i][cap];
        }

        int inc = 0, inc1 = 0, exc =0;

        if(cap >= wt[i]) {
            inc = val[i]+ Knapsack(wt, val, cap-wt[i], i,
dp);
            inc1 = val[i] + Knapsack(wt, val, cap-wt[i],
i+1, dp);
```

```
        }
        exc = Knapsack(wt, val, cap, i+1, dp);

        return dp[i][cap] = Math.max(inc, Math.max(inc,
exc));
    }
}
```

Count Number of Binary Strings

You are provided an integers N. You need to count all possible distinct binary strings of length N such
are no consecutive 1's.

Input Format

First line contains integer t which is number of test case. For each test case, it contains an integer n w
the length of Binary String.

Constraints

1<=t<=100 1<=n<=90

Output will in be Long

Output Format

Print the number of all possible binary strings.

Sample Input

```
2
2
3
```

Sample Output

```
3
5
```

Explanation

1st test case : 00, 01, 10 2nd test case : 000, 001, 010, 100, 101

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner pavan = new Scanner(System.in);
        int pt = pavan.nextInt();
        while(pt-- > 0){
```

```
            int pn = pavan.nextInt();
            long [][]dp = new long[2][pn+1];
            System.out.println(BSCount(pn, dp, 0));
        }
    }

    public static long BSCount(int n, long[][]dp, int
prev){
        if(n==1){
            if(prev == 0)
                return 2;
            return 1;
        }
        int count = 0;
        if(dp[prev][n]  != 0)
            return dp[prev][n];
        long count1= 0;
        if(prev == 0){
            count1 = BSCount(n-1, dp, 1);
        }
        long count2 = BSCount(n-1, dp, 0) ;
        return dp[prev][n] =count1+count2;
    }
}
```

Longest Increasing subsequence

Find the length of the longest subsequence in a given array A of integers such that all elements of the subsequence are sorted in strictly ascending order.

Input Format

The first line contains a single integer n.
Next line contains n space separated numbers denoting the elements of the array.

Constraints

$0 < n < 10^5$
$0 < A_i < 10^5$

Output Format

Print a single line containing a single integer denoting the length of the longest increasing subsequenc

Sample Input

```
6
50 3 10 7 40 80
```

Sample Output

```
4
```

Explanation

The longest subsequence in test case is : 3,7,40,80

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner pavan =  new Scanner(System.in);
        int pn = pavan.nextInt();
        int []parr = new int[pn];
        for(int i=0; i<pn; i++){
            parr[i] = pavan.nextInt();
        }

        System.out.println(LIS(parr));
```

```
    }

    public static int LIS(int []arr) {
        int []pdp = new int [arr.length];
        Arrays.fill(pdp,1);
        for(int i=0; i<pdp.length; i++) {
            for(int j=i-1; j>=0; j--) {
                if(arr[j] < arr[i]) {
                    int a = pdp[j]+1;
                    pdp[i] = Math.max(pdp[i], a);
                }
            }
        }

        int max = pdp[0];
        for(int i=1; i<pdp.length; i++) {
            if(max < pdp[i]) {
                max = pdp[i];
            }
        }
        return max;
    }
}
```

Coin Change

Given a value N, if we want to make change for N cents, and we have infinite supply of each of S valued coins, In how many ways can we make the change? The order of coins doesn't matter.

Input Format

First line of input contain two space separated integers N and M. Second line of input contains M separated integers - value of coins.

Constraints

```
1<=N<=250 1<=m<=50 1 <= Si <= 50
```

Output Format

Output a single integer denoting the number of ways to make the given change using given coi

Sample Input

```
10 4
2 5 3 6
```

Sample Output

```
5
```

Explanation

```
22222
2233
226
235
55

Total 5 ways
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner pavan = new Scanner(System.in);
        int pVal = pavan.nextInt();
```

```
        int pn = pavan.nextInt();
        int []parr = new int[pn];
        for(int i=0; i<pn; i++) {
            parr[i] = pavan.nextInt();
        }

        int [][]dp = new int[pVal+1][parr.length];
        for(int i=0; i<pVal+1; i++) {
            Arrays.fill(dp[i],-1);
        }
        System.out.println(CoinChange(parr, pVal, 0, dp));
    }

        public static int CoinChange(int []coin, int
amount, int i, int[][]dp) {

        if(amount == 0)
            return 1;

        if(i==coin.length)
            return 0;

        if(dp[amount][i] != -1)
            return dp[amount][i];

        int inc = 0;
        int exc = 0;
        if(amount >= coin [i]) {
            inc = CoinChange(coin, amount-coin[i], i,dp);
        }

        exc = CoinChange(coin, amount,i+1,dp);
        return dp[amount][i] = inc+exc;
    }
}
```

Boardpath(Count , Print)

Take as input N, a number. N is the size of a snakes and ladder board (without any snakes and ladders). Take as input M, a number. M is the number of faces of the dice.

a. Write a recursive function which returns the count of different ways the board can be traveled using the dice. Print the value returned.

b. Write a recursive function which prints dice-values for all valid paths across the board (void is the return type for function).

Input Format

Enter a number N (size of the board) and number M(number of the faces of a dice)

Constraints

```
M<=100
N<=100
M^N <=10^9
```

Output Format

Display the number of paths and print all the paths in a space separated manner

Sample Input

```
3
3
```

Sample Output

```
111 12 21 3
4
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here

        Scanner Pavan = new Scanner(System.in);
        int ptarget = Pavan.nextInt();
        int pdice_face = Pavan.nextInt();
```

```
        System.out.println("\n"
+Print_path(0,ptarget,"",pdice_face));

    }

    public static int Print_path(int sum, int des,String
ans, int dice_face) {
        if(sum==des) {
            System.out.print(ans+" ");
            return 1;
        }
        else if(sum > des) {
            return 0;
        }

        int count = 0;

        for(int dice=1; dice<=dice_face; dice++) {
            count += Print_path(sum+dice, des,
ans+dice,dice_face);
        }

        return count;
    }
}
```

0-1 Knapsack

You are packing for a vacation on the sea side and you are going to carry only one bag with capa
1000). You also have N (1<= N <= 1000) items that you might want to take with you to the sea side. U
you can not fit all of them in the knapsack so you will have to choose. For each item you are given i
value. You want to maximize the total value of all the items you are going to bring. What is this max
value?

Input Format

On the first line you are given N and S.
Second line contains N space separated integer where ith integer denotes the size of ith item. Third
N space seperated integers where ith integer denotes the value of ith item.

Constraints

```
1 <= S,N <= 1000
```

Output Format

Output a single integer showing the maximum value that can be obtained by optimally choosing t

Sample Input

```
5 4
1 2 3 2 2
8 4 0 5 3
```

Sample Output

```
13
```

Explanation

```
Total capacity = 4.
Pick size 1 with value 8 and size 2 with value 5.
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner pavan = new Scanner(System.in);
        int pw = pavan.nextInt();
```

```
        int pcap = pavan.nextInt();
        int []pweight = new int[pw];
        int []pvalue = new int[pw];
        for(int i=0; i<pw; i++){
            pweight[i] = pavan.nextInt();
        }

        for(int i=0; i<pw; i++){
            pvalue[i] = pavan.nextInt();
        }

        int [][]dp = new int[pw+1][pcap+1];
        for(int i=0; i<dp.length; i++){
            Arrays.fill(dp[i],-1);
        }
        System.out.println(Knapsack(pweight, pvalue, pcap,
0, dp));
    }

    public static int Knapsack(int[] wt, int [] val, int
cap, int i, int [][]dp) {
        if(i==wt.length || cap == 0)
            return 0;

        if(dp[i][cap] != -1) {
            return dp[i][cap];
        }

        int inc = 0, exc =0;

        if(cap >= wt[i]) {
            inc = val[i] + Knapsack(wt, val, cap-wt[i],
i+1, dp);
        }
        exc = Knapsack(wt, val, cap, i+1, dp);

        return dp[i][cap] = Math.max(inc,  exc);
    }
}
```

Climbing Stairs

Reaching to the top of a staircase, it takes n steps.
The task can be accomplished either by climbing 1 step or 2 steps at a time.
In how many different ways can you climb to the top.
Note: n will be a positive integer.

Input Format

First line contains an integer n.

Constraints

None

Output Format

Print the total number of distinct ways you can climb to top.

Sample Input

2

Sample Output

2

Explanation

None

```
import java.util.*;
public class Main {
    public static void main (String args[]) {
        Scanner pavan = new Scanner(System.in);
```

```
        int pn = pavan.nextInt();
        int[]pdp = new int[Math.max(pn,2)+1];
        pdp[1] = 1;
        pdp[2] = 2;
        for(int pi=3; pi<=pn; pi++){
            pdp[pi] = pdp[pi-1]+pdp[pi-2];
        }
        System.out.println(pdp[pn]);
    }
}
```

Smart Robber

A professional robber planning to rob houses along a street. Each house has a certain amount
the only constraint stopping him from robbing each of them is that adjacent houses have secu
connected and it will automatically contact the police if two adjacent houses were broken into
night.

Given a list of non-negative integers representing the amount of money of each house, determ
amount of money he can rob tonight without alerting the police.

Input Format

First line contains integer t as number of testcases. Second line contains integer n as size of arr
contains a single integer as element of array.

Constraints

None

Output Format

Print the maximum money as output.

Sample Input

```
1
4
1 2 3 1
```

Sample Output

```
4
```

```java
import java.util.*;
public class Main {
    public static void main (String args[]) {
        Scanner pavan = new Scanner(System.in);
        int pt = pavan.nextInt();
        while(pt-- >0){
            int pn = pavan.nextInt();
            int []pamount = new int[pn];
```

```
            for(int i=0; i<pn; i++){
                pamount[i] = pavan.nextInt();
            }

            int []dp = new int[pamount.length];
            Arrays.fill(dp, -1);
            System.out.println(RobberTD(pamount,0,dp));
        }
    }

    public static int RobberTD(int []arr, int i, int []dp)
{
        if (i>=arr.length)
            return 0;

        if(dp[i]!=-1) {
            return dp[i];
        }

        int rob = arr[i] + RobberTD(arr,i+2,dp);
        int Dont_rob = RobberTD(arr,i+1,dp);
        return dp[i] = Math.max(rob,  Dont_rob);
    }
}
```

Minimum Path Sum

Josh is stuck in a m*n grid. He has to travel from top left to bottom right.For every cell t
amount of money josh has to pay. Your task is to find out minimum amount of money jo
bottom right.

Input Format

First line contains two space separated integers m and n i.e number of rows and colum
each containing n integers, denoting the cost of which josh has to pay to pass through

Constraints

None

Output Format

A single integer denoting minimum cost josh has to pay.

Sample Input

```
3 3
1 3 1
1 5 1
4 2 1
```

Sample Output

```
7
```

Explanation

Josh will take 1→3→1→1→1 path to minimize the amount he has to pay

---

```java
import java.util.*;
public class Main {
    public static void main (String args[]) {
```

```java
        Scanner pavan = new Scanner(System.in);
        int pr = pavan.nextInt();
        int pc = pavan.nextInt();
        int [][]pmat = new int[pr][pc];
        for(int pi=0; pi<pr; pi++){
            for(int pj=0; pj<pc; pj++){
                pmat[pi][pj] = pavan.nextInt();
                // System.out.println(pmat[pi][pj]);
            }
        }


        int [][]dp =new int[pr][pc];
        for(int i=0; i<pr; i++){
            Arrays.fill(dp[i], 99999);
        }
        System.out.println(PathSum(pmat, 0, 0, dp));
    }

    public static int PathSum(int [][]mat, int cr, int cc,
int[][]dp) {
        if(cr == mat.length-1 && cc==mat[0].length-1) {
            return mat[cr][cc];
        }

        if(cr >= mat.length || cc>=mat[0].length) {
            return Integer.MAX_VALUE;
        }

        if(dp[cr][cc] != 99999) {
            return dp[cr][cc];
        }

        int h = PathSum(mat, cr, cc+1,dp);
        int v = PathSum(mat, cr+1, cc,dp);
        return dp[cr][cc] = Math.min(h, v) + mat[cr][cc];
    }
}
```

Count of different ways to express N as the sum of 1, 3 and 4

Given N, count the number of ways to express N as sum of 1, 3 and 4.

Input Format

First line contains the size of the array. Next line contains array elements.

Constraints

$1 <= N <= 10^8$

Output Format

Print the integer answer.

Sample Input

```
4
```

Sample Output

```
4
```

Explanation

```
1+1+1+1
1+3
3+1
4
```

```java
import java.util.*;
public class Main {
    public static void main (String args[]) {
```

```
        Scanner psc = new Scanner(System.in);
        int pn = psc.nextInt();
        long []pdp = new long [Math.max(3,pn)+1];
        pdp[0] = pdp[1] = pdp[2] = 1;
        pdp[3] = 2;
        for(int i=4; i<=pn; i++){
            pdp[i] = pdp[i-1]+pdp[i-3]+pdp[i-4];
        }
        System.out.println(pdp[pn]);
        // System.out.println(Ways(pn,pdp));
    }

    // public static int Ways(int pn, int[]pdp) {
    //  if(pn==0)
    //      return 1;
    //  if(pn<0)
    //      return 0;
    //  if(pdp[pn] != 0)
    //      return pdp[pn];

    //  int n1 = Ways(pn-1,pdp);
    //  int n2 = Ways(pn-3,pdp);
    //  int n3 = Ways(pn-4,pdp);

    //  return pdp[pn] = n1+n2+n3;
    // }

}
```

LCS with 3 Strings

Given 3 strings ,the task is to find the longest common sub-sequence in all three given sequ

Input Format

First line contains first string . Second line contains second string. Third line contains the third

Constraints

The length of all strings is |s|< 200

Output Format

Output an integer denoting the length of longest common subsequence of above three stri

Sample Input

```
GHQWNV
SJNSDGH
CPGMAH
```

Sample Output

```
2
```

Explanation

"GH" is the longest common subsequence

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner pavan = new Scanner(System.in);
        String s1 = pavan.next();
        String s2 = pavan.next();
        String s3 = pavan.next();

        int [][][]pdp = new
int[s1.length()][s2.length()][s3.length()];
        for(int i=0;i<pdp.length;i++) {
```

```
            for(int j=0; j<pdp[0].length; j++){
                Arrays.fill(pdp[i][j], -1);
            }
        }
        System.out.println(LCS(s1, s2, s3, 0, 0, 0, pdp));
    }

    public static int LCS(String s1, String s2, String s3,
int i, int j, int k, int [][][]dp) {
        if(s1.length() == i || s2.length() == j ||
s3.length() == k)
            return 0;

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

        int ans = 0;
        if(s1.charAt(i) == s2.charAt(j) && s2.charAt(j) ==
s3.charAt(k)) {
            dp[i][j][k] = 1+LCS(s1, s2, s3, i+1, j+1, k+1,
dp);
        }
        else {
            int fs = LCS(s1, s2, s3, i+1, j, k, dp);
            int ss = LCS(s1, s2, s3, i, j+1, k, dp);
            int ts = LCS(s1, s2, s3, i, j, k+1, dp);
            dp[i][j][k] = Math.max(fs, Math.max(ss,ts));
        }
        return dp[i][j][k];
    }
}
```

Exchanging Coins

Tughlaq introduces a strange monetary system. He introduces copper coins and each coin has a number wri
on it. A coin n can be exchanged in a bank into three copper coins: n/2, n/3 and n/4. A coin can also be sold
gold. One can get as much grams of gold as the number written on the coin. You have one copper coin. Wha
the maximum weight of gold one can get from it?

Input Format

The input file contains a single integer ***n***, the number on the coin.

Constraints

```
0 <= n <= 1 000 000 000
```

Output Format

Print the maximum weight of gold you can get.

Sample Input

```
12
```

Sample Output

```
13
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner pavan = new Scanner(System.in);
        int pn = pavan.nextInt();
        HashMap<Integer,Long> dp =new HashMap<>();
        System.out.println(maxVal(pn, dp));
    }
    public static long maxVal(int n, Map<Integer,Long> dp)
{
        if(n<=0)
            return 0;
        if(n==1)
            return 1;
```

```
        if(dp.containsKey(n)){
            return dp.get(n);
        }

        long n1 = maxVal(n/2, dp);
        long n2 = maxVal(n/3, dp);
        long n3 = maxVal(n/4, dp);
        dp.put(n, Math.max(n1+n2+n3, n));
        return dp.get(n);
    }
}
```

Valentine Magic

It's Valentine's Day in Russia today. as we all know number of girls in Russia is more than number of boys 😜 boys have an extra advantage while choosing girl for the valentine evening. Each boy has certain number c chocolates and each girl has certain number of candies. Now you being the anchor of evening wants to pa the boys with girls such that the sum of absolute difference between boy's chocolate and girl's candy in a p minimized. Ofcourse some of the girls will remain unpaired but that's okay as we are in Russia 😜

Input Format

The first line consists of two integers N and M. then follow N integers in second line. Then follow M integers in line. M >= N

Constraints

1 <= N <= 5000
1 <= M <= 5000
M >= N
1 <= chocolates <= 1000000
1 <= candies <= 1000000

Output Format

The only line which consists of required sum of absolute differences.

Sample Input

```
2 5
4 5
1 2 3 4 5
```

Sample Output

```
0
```

Explanation

we can pair boy numbered 1 with girl numbered 4 and boy numbered 2 with girl numbered 5

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner pavan =  new Scanner(System.in);
        int pboys = pavan.nextInt();
        int pgirls = pavan.nextInt();
        int []boys = new int[pboys];
        int []girls = new int[pgirls];
```

```
        for(int i=0; i<pboys; i++){
            boys[i] = pavan.nextInt();
        }

        for(int i=0; i<pgirls; i++){
            girls[i] = pavan.nextInt();
        }

        Arrays.sort(boys);
        Arrays.sort(girls);
        int [][]dp = new
int[boys.length+1][girls.length+1];
        for(int i=0; i<dp.length; i++) {
            Arrays.fill(dp[i], -1);
        }
        System.out.println(MinDiff(boys, girls, 0, 0,
dp));
    }

    public static int MinDiff(int []boys, int [] girls,
int i, int j, int [][]dp) {
        if(i == boys.length)
            return 0;
        if(j == girls.length)
            return 1000001;

        if(dp[i][j]  != -1)
            return dp[i][j];
        int pair = Math.abs(boys[i] - girls[j]) +
MinDiff(boys, girls, i+1, j+1, dp);
        int No_Pair = MinDiff(boys, girls, i, j+1, dp);
        return dp[i][j] = Math.min(pair,  No_Pair);
    }

}
```

Length Of LCS

A subsequence is a sequence that can be derived from another sequence by deleting some elem changing the order of the remaining elements. For example, the sequence {A,B,D} is a subsequenc {A,B,C,D,E,F}, obtained after removal of elements C, E and F.

Given two strings A and B of size n and m respectively, you have to find the length of Longest Com Subsequence(LCS) of strings A and B, where LCS is the longest sequence present in both A and B.

Input Format

Two strings A and B.

Constraints

```
1 <= n,m <= 10^3
```

Output Format

Output the LCS of A and B.

Sample Input

```
abc
acd
```

Sample Output

```
2
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner pavan = new Scanner(System.in);
        String ps1 = pavan.next();
        String ps2 = pavan.next();
```

```
        int [][]pdp = new int[ps1.length()][ps2.length()];
        for(int i=0;i<pdp.length;i++) {
            Arrays.fill(pdp[i], -1);
        }
        System.out.println(LCS(ps1, ps2, 0, 0, pdp ));
    }

    public static int LCS(String s1, String s2, int pi,
int pj, int [][]dp) {
        if(s1.length() == pi || s2.length() == pj)
            return 0;
        if(dp[pi][pj] != -1)
            return dp[pi][pj];
        int ans = 0;
        if(s1.charAt(pi) == s2.charAt(pj)) {
            dp[pi][pj] = 1+LCS(s1, s2, pi+1, pj+1, dp);
        }
        else {
            int fs = LCS(s1, s2, pi+1, pj, dp);
            int ss = LCS(s1, s2, pi, pj+1, dp);
            dp[pi][pj] = Math.max(fs, ss);
        }
        return dp[pi][pj];
    }
}
```

Minimum Money Needed

Cody went to the market to buy some oranges for his N friends. There he finds oranges wrappe
the price of i^th packet as val[i].
Now he wants to buy exactly W kg oranges, so he wants you to tell him what minimum price he
exactly W kg oranges. Weight of i^th packet is i kg. If price of i^th packet is -1 then this packet i
sale. The market has infinite supply of orange packets.

Input Format

First line of input contains two space separated integers N and W, the number of friend he has
Oranges in kilograms which he should buy.

The second line contains W space separated integers in which the i^th integer specifies the pri
packet. A value of -1 denotes that the corresponding packet is unavailable

Constraints

```
1 <= N,W,val[ ] <= 10^3
```

Output Format

Output a single integer denoting the minimum price Code should pay to get exactly W kg oran
it is not possible to fill the bag.

Sample Input

```
5 5
1 2 3 4 5
```

Sample Output

```
5
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
     Scanner inn = new Scanner(System.in);
            int n = inn.nextInt();
            int w = inn.nextInt();
            int arr[] = new int[w];
            for (int i = 0; i < w; i++) {
```

```
                arr[i] = inn.nextInt();
            }

//////////////////////////////////////////////////////////
//////////
            int dipadp[][]=new int[w+1][w+1];
            for (int i = 0; i <=w ; i++) {
                Arrays.fill(dipadp[i],-1);
            }
            int res=pavan(w,w,arr,dipadp);
            res=res==Integer.MAX_VALUE?-1:res;
            System.out.println(res);
        }

        public static int pavan(int n,int w,int arr[],int
[][]dp){

//////////////////////////////////////////////////////////
/////////////
            if(w==0){
                return 0;
            }
            if(n==0){
                return Integer.MAX_VALUE;

            }

            if(dp[n][w]!=-1){
                return dp[n][w];
            }
            int lo=Integer.MAX_VALUE;
            if(arr[n-1]!=-1 && w-n>=0){
                lo=arr[n-1]+pavan(n,w-n,arr,dp);
            }
            int doo=0+pavan(n-1,w,arr,dp);

            return dp[n][w]=Math.min(lo,doo);

        }
```

```
}
```

Ugly Numbers

You are provided a sequence of number. All numbers of that sequence is in increasing order (inclu
whose only prime factors are 2, 3 or 5 (except 1). You need to find the nth number of that sequenc

Input Format

First line contains integer t which is number of test case. For each test case, it contains an intege

Constraints

1<=t<=100 1<=n<=10000

Output Format

Print nth number of that sequence.

Sample Input

```
2
7
10
```

Sample Output

```
8
12
```

Explanation

Sequence : 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, ...

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner pavan = new Scanner(System.in);
            int pt = pavan.nextInt();
            while(pt-- > 0) {
            int n = pavan.nextInt();
            long next_multiple_2 = 2;
            long next_multiple_3 = 3;
            long next_multiple_5 = 5;
```

```
            int i2 =0, i3 = 0, i5 = 0;
            long [] ugly = new long [n];
            ugly[0] = 1;
            long next_ugly =1;

            for(int i = 1; i<n; i++){
                next_ugly =
Math.min(next_multiple_2,Math.min(next_multiple_3,next_mul
tiple_5));
                ugly[i] = next_ugly;

                if(ugly[i] == next_multiple_2){
                    i2++;
                    next_multiple_2 = ugly[i2]*2;
                }
                if(ugly[i] == next_multiple_3){
                    i3++;
                    next_multiple_3 = ugly[i3]*3;
                }
                if(ugly[i] == next_multiple_5){
                    i5++;
                    next_multiple_5 = ugly[i5]*5;
                }
            }
            System.out.println(next_ugly);
        }
    }
}
```

Palindrome Partitioning

You are given a string. You need to partition that string such that each substrings after partitioning would be palindromic string. You have to do this work with minimum number of partitioning.

Input Format

First line contains integer t which is number of test case. For each test case, it contains a string S.

Constraints

```
1<=t<=50 1<=S<=1000
```

Output Format

Print the minimum number of partitioning.

Sample Input

```
1
ababbbabbababa
```

Sample Output

```
3
```

Explanation

```
a|babbbab|b|ababa
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner pavan = new Scanner(System.in);
        int pt = pavan.nextInt();
        while(pt-- > 0){
            String s = pavan.next();

            int [][]dp = new int[s.length()][s.length()];

            for(int i=0; i<dp.length; i++){
                Arrays.fill(dp[i] , 100001);
            }
```

```
            System.out.println(sol(s , 0 , s.length()-1,
dp));
        }
    }

     public static int sol(String s , int i , int j, int
[][]dp)
    {
        int left,right;
        if(i>=j)
            return 0;

        if(is_palindrome(s , i , j) == true)
            return 0;

        if(dp[i][j] != 100001)
            return dp[i][j];


        for(int k=i; k<j ; k++)
        {
            if(is_palindrome(s , i , k ) == true)
            {
                int count = Math.min(dp[i][j] , sol(s , i
, k, dp) + sol(s, k+1 , j, dp) +1);

                dp[i][j] = count;
            }
        }
        return dp[i][j];


    }


    public static boolean is_palindrome(String s , int st
, int end){
        while(st < end)
        {
            if(s.charAt(st++) != s.charAt(end--))
                return false;
        }
```

```
        return true;
    }
}
```

Buying Fruits

Prateek went to purchase fruits mainly apples, mangoes and oranges. There are N different fruit s
Each fruit seller sells all three fruit items, but at different prices. Prateek, obsessed by his nature to
optimally, decided not to purchase same fruit from adjacent shops. Also, **Prateek will purchase e**
of fruit item (only 1kg) from each shop.
Prateek wishes to spend minimum money buying fruits using this strategy. Help Prateek determine
money he will spend. If he becomes happy, he may offer you discount on your favorite course in C
All the best!

Input Format

First line indicates number of test cases **T**. Each test case in its first line contains **N** denoting the nu
sellers in Fruit Market. Then each of next **N** lines contains three space separated integers denotin
mangoes and oranges per kg with that particular fruit seller.

Constraints

1 ≤ T ≤ 10 1 ≤ N ≤ 10^5 Cost of each
fruit(apples/mangoes/oranges) per kg does not exceed 10^4

Output Format

For each test case, output the minimum cost of shopping taking the mentioned conditions into a
separate line.

Sample Input

```
2
3
1 50 50
50 50 50
1 50 50
4
1 2 3
3 2 1
1 1 1
1 1 1
```

Sample Output

```
52
4
```

Explanation

Test Case 1:
There are two ways, each one gives 52 as minimum cost. One
way is buy apples from first shop, mangoes from second shop
and apples from third shop or he can buy apples from first
shop, oranges from second shop and apples from third shop.

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner pavan = new Scanner(System.in);
        int ptest = pavan.nextInt();
        while(ptest-->0){
            int pn= pavan.nextInt();
            int parr[][]=new int[pn][3];
            for (int pi = 0; pi <pn ; pi++) {
                for (int pj = 0; pj <3 ; pj++) {
                    parr[pi][pj]=pavan.nextInt();
                }
            }
            System.out.println(fruits(parr));

        }
    }
    public static int fruits(int [][]points){
        int n=points.length;
        int[][] dp= new int [n][4];

        dp[0][0]=Math.min(points[0][1],points[0][2]);
        dp[0][1]=Math.min(points[0][0],points[0][2]);
        dp[0][2]=Math.min(points[0][1],points[0][0]);

dp[0][3]=Math.min(points[0][0],Math.min(points[0][1],point
s[0][2]));


        for (int day = 1; day < n; day++) {
            for (int last = 0; last < 4; last++) {
                dp[day][last]=Integer.MAX_VALUE;
                for (int task = 0; task <3 ; task++) {
                    if(task!=last){
                        int
point=points[day][task]+dp[day-1][task];

dp[day][last]=Math.min(point,dp[day][last]);
                    }
                }

            }
```

```
        }

        return dp[n-1][3];

    }

}
```

K-Ordered LCS

Any programmer worth his salt would be familiar with the famous Longest Common Subsec
Mancunian was asked to solve the same by an incompetent programmer. As expected, he
complicate matters, a twist was introduced in the problem.

In addition to the two sequences, an additional parameter **k** was introduced. A k-ordered
the LCS of two sequences if you are allowed to change **atmost** k elements in the first seque
wish to. Can you help Mancunian solve this version of the classical problem?

Input Format

The first line contains three integers N, M and k, denoting the lengths of the first and secon
value of the provided parameter respectively. The second line contains N integers denoting
first sequence. The third line contains M integers denoting the elements of the second sequ

Constraints

$1 <= N, M <= 2000$
$1 <= k <= 5$
$1 <=$ element in any sequence $<= 10^9$

Output Format

Print the answer in a new line.

Sample Input

```
5 5 1
1 2 3 4 5
5 3 1 4 2
```

Sample Output

```
3
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner pavan = new Scanner(System.in);
        int pn = pavan.nextInt();
        int pm = pavan.nextInt();
        int pk = pavan.nextInt();
```

```
        int []narr = new int[pn];
        int []marr = new int[pm];
        for(int i=0; i<pn; i++){
            narr[i] = pavan.nextInt();
        }

        for(int j=0; j<pm; j++){
            marr[j] = pavan.nextInt();
        }

        int[][][]dp = new int[pn][pm][pk];
        for(int i=0; i<pn; i++){
            for(int j=0; j<pm; j++){
                Arrays.fill(dp[i][j], -1);
            }
        }

        System.out.println(PKLCS(narr, marr, pn-1, pm-1,
dp, pk-1));

    }


    public static int PKLCS(int[] psub1,int[] psub2,int
i,int j,int[][][]pdp,int k){
        if(i<0||j<0){
            return 0;
        }
        if(k<0){
            return -1000000000;
        }


        int ans=pdp[i][j][k];

    // System.out.println(ans);
    if(ans!=-1){
            return pdp[i][j][k];
        }
```

```
        ans=Math.max(PKLCS(psub1,psub2,i-
1,j,pdp,k),PKLCS(psub1,psub2,i,j-1,pdp,k));

        if(psub1[i]==psub2[j]){
             ans=Math.max(ans,1+PKLCS(psub1,psub2,i-1,j-
1,pdp,k));
        }

        ans=Math.max(ans,1+PKLCS(psub1,psub2,i-1,j-
1,pdp,k-1));


        return pdp[i][j][k]=ans;

    }
}
```

Number of Steps to reach 1

Given a positive number N your task is to bring this number to 1 by performing only a set of op
operations can be either dividing the number by 2 only if the number is even or you can add o
the number is odd.
More Precisely:
1) N=N/2 (if N is even)
2)N=N+1/ N=N-1 (if N is odd)
Your task is to minimize these number of operations.

Input Format

A single positive integer N

Constraints

```
n<=100000
```

Output Format

Print on a single line the minimum number of steps needed to reach 1 by performing the given

Sample Input

```
8
```

Sample Output

```
3
```

Explanation

```
8->4->2->1
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner pavan = new Scanner(System.in);
        int pn = pavan.nextInt();
        int []dp = new int[pn+2];
        System.out.println(MinSteps(pn, dp));
```

```
    }
    public static int MinSteps(int n, int[]dp) {
        // TODO Auto-generated method stub
        if(n==1 || n==0)
            return 0;
        int ev = 0;
        int od = 0;
        if(dp[n] != 0)
            return dp[n];
        if(n%2 == 0) {
            ev = MinSteps(n/2, dp)+1;
        }
        else {
            od = Math.min(MinSteps(n-1, dp), MinSteps(n+1,
dp))+1;
        }
        return dp[n] = ev+od;
    }
}
```

Minimum trials needed (Plate Dropping)

Aakash has **K** identical plates and **N** floors. He needs to find the lowest floor at which
aka critical floor. However, he doesn't like climbing stairs again and again. So he decid
him design a strategy that would tell him the minimum no of trials he need to perform
floor.

Assume that plate will only break at critical floor and floors higher than that.

_Hint :
Recursion tells u what u intend to calculate.
Test case 3 is only for students who think that the test cases are easy.

Input Format

The first line contains **T**, the number of test cases.
Next T line follows 2 space separated integers, first being the number of plates **K**, next
N.

Constraints

$0 < T < 10$

$0 < K < 1000$

$0 < N < 1000$

Output Format

T lines of required answer

Sample Input

```
1
2 3
```

Sample Output

```
2
```

Explanation

Aakash can start dropping plates from floor 1, 2 or 3.

Consider floor 1 :

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner pavan = new Scanner(System.in);
        int pt = pavan.nextInt();
        while(pt-- > 0){
            int plt = pavan.nextInt();
            int flr = pavan.nextInt();
            System.out.println(MinTN(plt, flr));
        }
    }

    public static int MinTN(int plt,int fl){
        int pdp[][]=new int[plt+1][fl+1];
        for (int i = 1; i <=fl ; i++) {
            pdp[1][i]=i;
        }
        for (int i = 1; i <=plt ; i++) {
            pdp[i][1]=1;
            pdp[i][0]=0;
        }
        for (int i = 2; i <=plt ; i++) {
            for (int j = 2; j <=fl ; j++) {
                int jabab=Integer.MAX_VALUE;
                for (int k = 1; k <=j ; k++) {
                    jabab=Math.min(jabab,1+Math.max(pdp[i-
1][k-1],pdp[i][j-k]));
                }
                pdp[i][j]=jabab;
            }
        }
        return pdp[plt][fl];
    }
}
```

Print LCS

A subsequence is a sequence that can be derived from another sequence by deleting some elements wit changing the order of the remaining elements. For example, the sequence {A,B,D} is a subsequence of {A,B,C,D,E,F}, obtained after removal of elements C, E and F.

Given two strings A and B of size n and m respectively, you have to print the Longest Common Subsequen of strings A and B, where LCS is the longest sequence present in both A and B.

Note: It is gauranteed that there is only one unique longest common subsequence

Input Format

Two strings A and B.

Constraints

```
1 <= n,m <= 10^3
```

Output Format

Output the LCS of A and B.

Sample Input

```
abc
acd
```

Sample Output

```
ac
```

```java
import java.util.*;
public class Main {
    public static void main(String args[]) {
        // Your Code Here
        Scanner pavan = new Scanner(System.in);
        String ps1 = pavan.next();
        String ps2 = pavan.next();

        int pn = ps1.length();
        int pm = ps2.length();
```

```
        int pdp[][] = new int[pn + 1][pm + 1];
        for (int pi = 1; pi <= pn; pi++) {
            for (int pj = 1; pj <= pm; pj++) {
                if (ps1.charAt(pi - 1) == ps2.charAt(pj -
1)) {
                    pdp[pi][pj] = 1 + pdp[pi - 1][pj - 1];
                } else {
                    pdp[pi][pj] = 0 + Math.max(pdp[pi -
1][pj], pdp[pi][pj - 1]);
                }

            }
        }
        StringBuilder ans = new StringBuilder();
        int i = pn, j = pm;
        while (i > 0 && j > 0) {
            if (ps1.charAt(i - 1) == ps2.charAt(j - 1)) {
                ans.append(ps1.charAt(i - 1));
                i--;
                j--;
            } else if (pdp[i - 1][j] > pdp[i][j - 1]) {
                i--;
            } else {
                j--;
            }
        }
        System.out.println(ans.reverse().toString());
    }
}
```

Tiling Problem -1

Given a board of size **2xn** , you have place **2x1** tiles . You can place the tile horizontally or vertic
number of ways.

Input Format

Size n

Constraints

1<= n <= 100

Output Format

Number of ways

Sample Input

7

Sample Output

21

```java
import java.util.*;
public class Main {
    public static void main (String args[]) {
        Scanner pavan = new Scanner(System.in);
        int pn = pavan.nextInt();
        long []dp = new long[pn+1];
        System.out.println(NOW(pn,dp));
    }

    public static long NOW(int n, long []dp){
        if(n<=2){
            return n;
        }
        if(dp[n] != 0)
            return dp[n];

        return dp[n] = NOW(n-1, dp) + NOW(n-2, dp);
```

```
    }
}
```

Optimal Game Strategy -2

Piyush and Nimit are playing a coin game. They are given n coins with values x_1, x_2 x
They take alternate terms. In each turn, a player picks either the first coin or the last c
removes it from the row. The value of coin is received by that player. Determine the ma
can win with if he moves first. Both the players play optimally.

Input Format

First line contains the number of coins 'n'.
Second line contains n space separated integers where ith index denotes the value o

Constraints

$1 < N <= 10000$, N is always even
$0 <= A_i <= 1000000$

Output Format

Print a single line with the maximum possible value that Piyush can win with.

Sample Input

```
4
1 2 3 4
```

Sample Output

```
6
```

Explanation

Piyush will pick the coin 4. Then Nimit can pick either 1 or 3. In both the cases Piyush picks coin 2 and wins with a total of 6.

```java
import java.util.*;
public class Main {
```

```
    public static void main(String args[]) {
        // Your Code Here
        Scanner pavan = new Scanner(System.in);
        int pn = pavan.nextInt();
        int []parr=  new int[pn];
        for(int i=0; i<parr.length; i++) {
            parr[i] = pavan.nextInt();
        }

        int [][]pdp = new int[parr.length][parr.length];
        for(int i=0; i<pdp.length; i++) {
            Arrays.fill(pdp[i], -1);
        }

        //      int []arr = {2,3,8,4};
        System.out.println(OGS(parr,0,parr.length-1,
pdp));
    }

    public static int OGS(int []arr, int i, int j, int
[][]dp) {
        if(i>j) {
            return 0;
        }

        if(dp[i][j] != -1)
            return dp[i][j];
        int first = arr[i]+Math.min(OGS(arr,i+2,j, dp),
OGS(arr,i+1,j-1, dp));
        int second = arr[j]+Math.min(OGS(arr,i+1,j-1, dp),
OGS(arr,i,j-2, dp));

        return dp[i][j] = Math.max(first, second);
    }
}
```

Rod Cutting Problem

Given a rod of length n and a list of prices of rods of length of i, where 1<=i<=n, find the optir into smaller rods to maximize profit.The rod of length i has a value price[i-1].

Input Format

An integer N representing length of prices array. Prices array. An integer n representing rod l

Constraints

1<=n<=1000
1<=prices[i]<=1000

Output Format

An integer representing maximum profit

Sample Input

```
8
1 5 8 9 10 17 17 20
4
```

Sample Output

```
10
```

Explanation

Best: Cut the rod into two pieces of length 2 each to gain revenue of 5 + 5 = 10

Cut Profit 4 9 1, 3 (1 + 8) = 9 2, 2 (5 + 5) = 10 3, 1 (8 + 1) = 9 1, 1, 2 (1 + 1 + 5) = 7 1, 2, 1 (1 + 5 + 1) = 7 2, 1, 1 (5 + 1 + 1) = 7 1, 1, 1, 1 (1 + 1 + 1 + 1) = 4

```java
import java.util.*;
public class Main {
```

```java
    public static void main (String args[]) {
        Scanner pavan = new Scanner(System.in);
        int pn = pavan.nextInt();
        int []parr = new int[pn];
        for(int i=0; i<parr.length; i++){
            parr[i] = pavan.nextInt();
        }
        int rodlen = pavan.nextInt();
        int []dp = new int[rodlen+1];
        System.out.println(maxProfit(rodlen, parr, dp));
    }

    public static int maxProfit(int rodlen, int[]parr, int
[]dp){
        if(rodlen <= 0)
            return 0;
        int inc = 0;
        int exc = 0;
        if(dp[rodlen] != 0)
            return dp[rodlen];
        for(int j=1; j<=rodlen; j++){

            inc = Math.max(inc,parr[j-1]+maxProfit(rodlen-
j, parr, dp));
            // System.out.println(dp[rodlen][j]+"
"+rodlen+" "+j);
        }
        return dp[rodlen] =inc;
    }
}
```

House Robber 2

You are a professional robber planning to rob houses along a street. Each house has a certai
stashed. All houses at this place are arranged in a circle. That means the first house is the nei
one. Meanwhile, adjacent houses have a security system connected, and it will automatically
if two adjacent houses were broken into on the same night.

Given an integer array nums representing the amount of money of each house, Print the maxi
money you can rob tonight without alerting the police.

Input Format

First line takes an integer N(size of array)
Second line containing n space separated integer describing array

Constraints

```
1 <= N <= 100
0 <= nums[i] <= 1000
```

Output Format

Print the maximum amount of money you can rob tonight without alerting the police.

Sample Input

```
3
2 3 2
```

Sample Output

```
3
```

Explanation

```
You cannot rob house 1 (money = 2) and then rob house 3
(money = 2), because they are adjacent houses.
```

```java
import java.util.*;
public class Main {
    public static void main (String args[]) {
        Scanner pavan = new Scanner(System.in);
        int pn = pavan.nextInt();
```

```
        int []parr = new int[pn];
        for(int i=0; i<pn; i++) {
            parr[i] = pavan.nextInt();
        }
        System.out.println(HR2(parr));
    }

    public static int HR2(int []nums) {
           if(nums.length == 1){
               return nums[0];
            }
            if(nums.length == 2)
               return Math.max(nums[0],nums[1]);

            int []dp = new int[nums.length];
            dp[0] = nums[0];
            dp[1] = nums[1];
            for(int i=2; i<nums.length; i++){
               dp[i] = Math.max(nums[i]+dp[i-2], dp[i-
1]);
               dp[i-1] = Math.max(dp[i-1],dp[i-2]);
               // System.out.println(dp[i]+" "+dp[i-1]);
            }

            int temp  = dp[nums.length-2];
            // System.out.println("    "+temp);
            // System.out.println();
            dp[0] = nums[1];
            dp[1] = nums[2];
            for(int i=3; i<nums.length;i++){
               dp[i-1] = Math.max(nums[i]+dp[i-3],dp[i-
2]);
               dp[i-2] = Math.max(dp[i-3],dp[i-2]);
               // System.out.println(dp[i-1]+" "+dp[i-
2]);
            }

            int temp2 = Math.max(dp[nums.length -
3],dp[nums.length-2]);
            // System.out.println(temp2);
            return Math.max(temp,temp2);
    }
```

```
}
```