# L2 : Basic OF Recursion Practice Questions

## 1-Tut : Power

Send Feedback

Write a program to find x to the power n (i.e. x^n). Take x and n from the user. You need to return the answer.

Do this recursively.

**Input format :**

Two integers x and n (separated by space)

**Output Format :**

x^n (i.e. x raise to the power n)

**Constraints :**

1 <= x <= 30

0 <= n <= 30

**Sample Input 1 :**

3 4

**Sample Output 1 :**

81

**Sample Input 2 :**

2 5

**Sample Output 2 :**

32

```
int power(int x, int n) {
  /* Don't write main().
     Don't read input, it is passed as a function argument.
     Return output and don't print it.
     Taking input and printing output is handled automatically.
  */
   if(n == 0){
      return 1;
   }
   int smallans = power(x,n-1);
   return x * smallans;
}
```

## 2-Tut : Print Numbers

Send Feedback

Given is the code to print numbers from 1 to n in increasing order recursively. But it contains a few bugs that you need to rectify such that all the test cases pass.

**Input Format :**

Integer n

**Output Format :**

Numbers from 1 to n (separated by space)

**Constraints :**

1 <= n <= 10000

**Sample Input 1 :**

6

**Sample Output 1 :**

1 2 3 4 5 6

**Sample Input 2 :**

4

**Sample Output 2 :**

1 2 3 4

```
1.  void print(int n){
2.      if(n == 1){
3.          cout << n << " ";
4.          return;
5.      }
6.      //cout << n << " ";
7.      print(n - 1);
8.      cout << n << " ";
9.  }
```

## 3-Tut : Number of Digits

Send Feedback

Given the code to find out and return the number of digits present in a number recursively. But it contains a few bugs that you need to rectify such that all the test cases should pass.

**Input Format :**

Integer n

**Output Format :**

Count of digits

**Constraints :**

1 <= n <= 10^6

**Sample Input 1 :**

156

**Sample Output 1 :**

3

**Sample Input 2 :**

7

**Sample Output 2 :**

1

```
1.  int count(int n){
2.      if(n < 10){
3.          return 1;
4.      }
5.      int smallAns = count(n / 10);
6.      return smallAns + 1;
7.  }
```

## 4-Tut : What is the output

Send Feedback

What will be the output of the following code ?

```
#include <iostream>
using namespace std;

int func(int num){
    return func(num- 1);
}

int main() {
    int num = 5;
    int ans = func(num - 1);
    cout << ans;
}
```

### Options

Compilation Error

Runtime Error

5

None of these

Correct Answer : B

### Solution Description

Since the base case is missing in the code, therefore there will be infinite recursion calls and hence runtime error will occur.

## 5-Tut :What is the output

Send Feedback

What will be the output ?

```
#include <iostream>
using namespace std;

void print(int n){
    if(n < 0){
        return;
```

```
    }
    cout << n << " ";
    print(n - 2);
}

int main() {
    int num = 5;
    print(num);
}
```

## Options

Runtime error
5 4 3 2 1
5 3 1
None of these

Correct Answer : C

## 6-Tut : What is the output

Send Feedback
What will be the output of the following code ?

```
#include <iostream>
using namespace std;

void print(int n){
    if(n < 0){
        return;
    }
    if(n == 0){
        cout << n << " ";
        return;
    }
    print(n --);
    cout << n << " ";
}

int main() {
    int num = 3;
    print(num);
}
```

## Options

Runtime Error
3 2 1
3 3 3
0 1 2 3

Correct Answer : A

## Solution Description

In function print when a recursion call is being made n-- is being passed as input. Here we are using a post decrement operator, so the argument passed to the next function call is n and not n - 1. Thus there will be infinite recursion calls and runtime error will come.

Recursion and Arrays

## 7-Tut : Sum of Array

Send Feedback

Given an array of length N, you need to find and return the sum of all elements of the array.

Do this recursively.

**Input Format :**

Line 1 : An Integer N i.e. size of array

Line 2 : N integers which are elements of the array, separated by spaces

**Output Format :** Sum

**Constraints :**

1 <= N <= 10^3

**Sample Input 1 :**

3

9 8 9

**Sample Output 1 :** 26

**Sample Input 2 :**

3

4 2 1

**Sample Output 2 :**

7

```
int sum(int input[], int n) {
  /* Don't write main(). Don't read input, it is passed as function argument. Return output and don't print it.  Taking input and printing output is handled automatically. */
    if(n == 1){
        return input[0];
    }

    int smallans = sum(input+1,n-1);
    return input[0] + smallans;
}
```

## 8-Tut : Check Number

Send Feedback

Given an array of length N and an integer x, you need to find if x is present in the array or not. Return true or false.

Do this recursively.

### Input Format :

Line 1 : An Integer N i.e. size of array
Line 2 : N integers which are elements of the array, separated by spaces
Line 3 : Integer x

### Output Format :

'true' or 'false'

### Constraints :

1 <= N <= 10^3

### Sample Input 1 :

3
9 8 10
8

### Sample Output 1 :

true

### Sample Input 2 :

3
9 8 10
2

### Sample Output 2 :

false

```
1.  bool checkNumber(int input[], int size, int x) {
2.  
3.      if(size < 1){
4.          return false;
5.      }
6.  
7.      if(input[0] == x){
8.          return true;
9.      }
10. 
11.     return checkNumber(input+1,size-1,x);
12. }
```

## 9-Tut : First Index of Number

Send Feedback

Given an array of length N and an integer x, you need to find and return the first index of integer x present in the array. Return -1 if it is not present in the array.

First index means, the index of first occurrence of x in the input array.

Do this recursively. Indexing in the array starts from 0.

### Input Format :

Line 1 : An Integer N i.e. size of array
Line 2 : N integers which are elements of the array, separated by spaces
Line 3 : Integer x

### Output Format :

first index or -1

### Constraints :

1 <= N <= 10^3

### Sample Input :

4
9 8 10 8
8

### Sample Output :

1

```
1.  int firstIndex(int input[], int size, int x) {
2.
3.      if(size < 1){
4.          return -1;
5.      }
6.
7.      if(input[0] == x){
8.          return 0;
9.      }
10.
11.     int smallans = firstIndex(input+1,size-1,x);
12.     if(smallans != -1){
13.         return 1+smallans;
14.     }
15.     else{
16.         return smallans;
17.     }
18. }
```

## 10-Tut : Last Index of Number

Send Feedback

Given an array of length N and an integer x, you need to find and return the last index of integer x present in the array. Return -1 if it is not present in the array.

Last index means - if x is present multiple times in the array, return the index at which x comes last in the array.

You should start traversing your array from 0, not from (N - 1).

Do this recursively. Indexing in the array starts from 0.

### Input Format :

Line 1 : An Integer N i.e. size of array
Line 2 : N integers which are elements of the array, separated by spaces
Line 3 : Integer x

### Output Format :

last index or -1

### Constraints :

1 <= N <= 10^3

### Sample Input :

4
9 8 10 8
8

### Sample Output :

3

```
int lastIndex(int input[], int size, int x) {
 /* Don't write main().
    Don't read input, it is passed as function argument.
    Return output and don't print it.
    Taking input and printing output is handled automatically.
 */

   if(size < 1)
   {
     return -1;
   }


   int smallans = lastIndex(input+1,size-1,x);

   if(smallans == -1){
      if(input[0] == x){
         return 0;
      }
      else{
         return smallans;
      }
   }
   else
   {
     return 1+smallans;
   }

}
```

## 11-Tut : All Indices of Number

Send Feedback

Given an array of length N and an integer x, you need to find all the indexes where x is present in the input array. Save all the indexes in an array (in increasing order).

Do this recursively. Indexing in the array starts from 0.

### Input Format :

Line 1 : An Integer N i.e. size of array
Line 2 : N integers which are elements of the array, separated by spaces
Line 3 : Integer x

### Output Format :

indexes where x is present in the array (separated by space)

### Constraints :

1 <= N <= 10^3

### Sample Input :

5
9 8 10 8 8
8

### Sample Output :

1 3 4

```
int allIndexes(int input[], int size, int x, int output[]) {
    if(size < 1){
        return 0;
    }
    int oparrsize = allIndexes(input+1,size-1,x,output);

    for(int i = 0; i < oparrsize; i++){
        output[i] = output[i]+1;
    }

    if(input[0] == x)
    {

        for(int i = oparrsize-1; i >= 0; i--){
            output[i+1] = output[i];
        }
        output[0] = 0;
        return 1+oparrsize;
    }
    else
    {
        return oparrsize;
    }
}
```

# L3 : Time Space Complexity Practice Questions

## 1-Tut : Linear Search Worst Case

Send Feedback

The Worst case(s) occur in linear search algorithm when -

### Options

This problem may have one or more correct answers

Item is somewhere in the middle of the array
Item is the last element in the array
Item is present at the first index of the array.
Item is not in the array at all

Correct Answer : B , D

## 2-Tut : Worst Case Time Complexity of Insertion sort

Send Feedback

Worst case time complexity of insertion sort is ?

### Options

This problem may have one or more correct answers

O(N)
O(N^2)
O(NLogN)
O(LogN)

Correct Answer : B

## 3-Tut : Worst Case Time complexity of Selection Sort

Send Feedback

Worst case time complexity of Selection sort is ?

### Options

This problem has only one correct answer

O(N)
O(N^2)
O(NLogN)
O(LogN)

Correct Answer : B

## 4-Tut : Efficiency of an Algorithm

Send Feedback

Two main measures for the efficiency of an algorithm are -

## Options

This problem may have one or more correct answers

Processor and memory
Complexity and capacity
Time and space
Data and space

Correct Answer : C

## 5-Tut : Theoretical Analysis

Send Feedback

In theoretical analysis the time factor when determining the efficiency of algorithm is measured by -

## Options

This problem may have one or more correct answers

Counting microseconds
Counting the number of statements in code
Counting the number of unit operations
Counting the kilobytes of algorithm

Correct Answer; C

## 6-Tut : Time Complexity

Send Feedback

If the number of primary operations of an algorithm that takes an array of size n as input are 3n^2 + 5n.

The worst case time complexity of the algorithm will be ?

## Options

This problem may have one or more correct answers

O(n^3)
O((n^2)*logn)
O(n^2)
O(n)

Correct Answer : C

## 7-Tut : Time Complexity of Code

Send Feedback

What will be the Time Complexity of the following code in terms of ‘n’ ?

Refer the code for C++ -

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

Refer the same code in Java -

```
for(int i = 0; i < n; i++){
   for(; i < n; i++){
      System.out.println(i);
   }
}
```

Refer the same code in Python -

```
  i = 0
   while i<n :
        while i<n :
            print(i)

            i += 1
```

## Options

This problem may have one or more correct answers

O(n)
O(n^2)
O(logn)
O(nlogn)

Correct Answer : A

## 8-Tut : Time Complexity of Code

Send Feedback

What will be the Time Complexity of the following code in terms of ‘n’ ?

### Note : Assume k to be a constant value

Refer the code in C++ -

```
 for(int i = 0; i < n; i++){
    for(int j = 1 ; j < k; j++){
       cout << (i + j ) << endl;
    }
 }
```

Refer the same code in Java -

```
for(int i = 0; i < n; i++){
      for(int j = 1 ; j < k; j++){
        System.out.println(i + j);
      }
}
```

Refer the same code in Python -

```
for i in range(n):
    for j in range(k):
        print(i+j)
```

## Options

This problem may have one or more correct answers

O(n^2)
O(n)
O(logn)
None of these

Correct Answer : B

## 9-Tut : Operations for merging

Send Feedback

For merging two sorted arrays of size m and n into a sorted array of size m+n, we require operations -

## Options

This problem has only one correct answer

O(m * n)
O(m + n)
O(m) if m >= n
O(n) if n > m

Correct Answer : B

## 10-Tut : Worst Case Time complexity of Binary Search

Send Feedback

Worst case time complexity of Binary Search is ?

## Options

This problem has only one correct answer

O(N)
O(N^2)
O(NLogN)
O(LogN)

Correct Answer : D

## 11-Tut : Recurrence for Merge Sort

Send Feedback

What is the recurrence relation for merge sort :

## Options

This problem has only one correct answer

T(n) = 2T(n/2)
T(n) = 2T(n/2) + k
T(n) = 2T(n/2) + O(n)
T(n) = 2T(n/2) + O(log n)

Correct Answer : C

## 12-Tut : Merge sort

Send Feedback

What is the time complexity of merge sort :

## Options

This problem has only one correct answer

O(n)
O(n^2)
O(nlogn)
O(log n)

Correct Answer : C

## 13-Tut : What is time complexity

Send Feedback

What is the time complexity of following code ?

```
int multiplyRec(int m, int n){
    if(n == 1)
        return m;
    return m + multiplyRec(m,  n - 1);
}
```

## Options

This problem has only one correct answer

O(m*n)
O(n)
O(n^2)
O(m)

Correct Answer : B

## 14-Tut : What is time complexity

Send Feedback

What is the time complexity of following code ?

```
int sumOfDigits(int n){
        int sum;
        if(n < 10){
            return n;
        }
```

```
        sum = (n % 10) + sumOfDigits(n / 10);
        return sum;
}
```

## Options

This problem has only one correct answer

O(logn) - log is to the base 10
O(n)
O(n^2)
None of these

Correct Answer : A

## 15-Tut : Fibonacci

Send Feedback

What is the time complexity of following code for calculating nth fibonacci number

```
long fib(int n){
   if(n == 0 || n == 1){
        return n;
   }
   return fib(n - 1) + fib(n - 2);

}
```

## Options

This problem has only one correct answer

O(n)
O(n^2)
O(2^n)
O(n^3)

Correct Answer : C

## 16-Tut : Merge Sort space

Send Feedback

**The space complexity for merge sort is :**

## Options

This problem has only one correct answer

O(n)
O(n^2)
O(nlogn)
O(log n)

Correct Answer : A

## 17-Tut : Fibonacci

Send Feedback

The space complexity for finding nth fibonacci number using recursion is :

## Options

This problem has only one correct answer

O(n)
O(2^n)
O(log n)
O(n^2)
O(nlogn)

Correct Answer : A

Kadane’s Algorithm

1. https://www.codechef.com/problems/KCON
2.

# L4: Language Tools Practice Questions

1. Hussain set - https://www.codechef.com/problems/COOK82C
2. https://www.codechef.com/problems/VOTERS
3. https://www.codechef.com/problems/PERMPAL
4.

## 1-Ass : Love for Characters

Send Feedback

Ayush loves the characters ‘a’, ‘s’, and ‘p’. He got a string of lowercase letters and he wants to find out how many times characters ‘a’, ‘s’, and ‘p’ occurs in the string respectively. Help him find it out.

**Input:**

First line contains an integer denoting length of the string.
Next line contains the string.

**Constraints:**

1<=n<=10^5
‘a’<= each character of string <= ‘z’

**Output:**

Three space separated integers denoting the occurrence of letters ‘a’, ‘s’ and ‘p’ respectively.

**Sample Input:**

6
aabsas

**Sample output:**

3 2 0

```
#include<bits/stdc++.h>
using namespace std;
void aspcount(string str, int l){
    int a = 0,s = 0,p=0,i = 0;

    while(str[i] != '\0'){
        char ch = str[i];
        if(ch == 'a'){
            a++;
        }else if(ch == 's'){
            s++;
        }else if(ch == 'p'){
            p++;
        }
        i++;
    }
    cout << a << " " << s << " " << p << endl;
}

```

```
int main()
{
    int l;
    cin >> l;
    string str;
    cin >> str;
    aspcount(str,l);
            return 0;
}
```

## 2-Ass : Different Names

Send Feedback

In Little Flowers Public School, there are many students with same first names. You are given a task to find the students with same names. You will be given a string comprising of all the names of students and you have to tell the name and count of those students having same. If all the names are unique, print -1 instead.

Note: We don't have to mention names whose frequency is 1.

### Input Format:

The only line of input will have a string ‘str’ with space separated first names of students.

### Output Format:

Print the names of students along with their count if they are repeating. If no name is repeating, print -1

### Constraints:

1 <= |str| <= 10^5

Time Limit: 1 second

### Sample Input 1:

Abhishek harshit Ayush harshit Ayush Iti Deepak Ayush Iti

### Sample Output 1:

harshit 2

Ayush 3

Iti 2

### Sample Input 2:

Abhishek Harshit Ayush Iti

### Sample Output:

-1

```
#include<bits/stdc++.h>
#include <iostream>
using namespace std;
void stringcount(string str){
    map<string,int> mymap;
    string temp;
    stringstream X(str);

    while( getline(X,temp,' ') ){
```

```
        mymap[temp]++;
    }
    bool flag = false;
    map<string,int> :: iterator it;
    it = mymap.begin();

    while(it != mymap.end() ){
        if(it->second >= 2){
            flag = true;
            cout << it->first << " " << it->second << endl;
        }
        it++;
    }

    if(!flag){
        cout << -1;
    }

}
int main() {
    // Write your code here
    string str;
    getline(cin,str);
    stringcount(str);
}
```

## 3-Ass : Extract Unique characters

Send Feedback

Given a string S, you need to remove all the duplicates. That means, the output string should contain each character only once. The respective order of characters should remain same, as in the input string.

### Input format:

The first and only line of input contains a string, that denotes the value of S.

### Output format :

The first and only line of output contains the updated string, as described in the task.

### Constraints :

0 <= Length of S <= 10^8

Time Limit: 1 sec

### Sample Input 1 :

ababacd

### Sample Output 1 :

abcd

### Sample Input 2 :

abcde

### Sample Output 2 :

abcde

```cpp
#include<bits/stdc++.h>
string uniqueChar(string str) {
        // Write your code here
    string str1 = "";
    map<char,int> mymap;
    int i = 0;
    while(str[i] != '\0'){
        mymap[str[i]] = 1;
        i++;
    }
    i = 0;
    while(str[i] != '\0'){

        if(mymap[str[i]]){
            str1 += str[i];
            mymap[str[i]] = 0;
        }
        i++;
    }

    return str1;
}
```

## 4-Ass : Warm Reception

Send Feedback

There is only one beauty parlour in the town CodingNinjasLand. The receptionist at the beauty parlor is flooded with appointment requests because the “Hakori” festival is round the corner and everyone wants to look good on it.

She needs your help. The problem is they don’t have chairs in reception. They are ordering chairs from NinjasKart. They don’t want to order more than required. You have to tell the minimum number of chairs required such that none of the customers has to stand.

### Input Format :

First line contains the number of customers that will come. Second line contains N space-separated integers which represent the arrival timings of the customer. Third line contains N space-separated integers which represent the departure timings of the customer. Arrival and departure timings are given in 24-hour clock.

### Constraints:

1<= N <= 100

Arrival and departure timings lie in the range [0000 to 2359]

Time Limit: 1 second

### Output Format :

You have to print the minimum number of chairs required such that no customer has to wait standing.

### Sample Test Cases:

Sample Input 1 :

5
900 1000 1100 1030 1600
1900 1300 1130 1130 1800
Sample Output 1:

4
Explanation:

4 because at 1100 hours, we will have maximum number of customers at the shop, throughout the day. And that maximum number is 4.

```
#include<bits/stdc++.h>
using namespace std;
int main() {
        // Write your code here
    int n;
    cin >> n;
    int arrival[n];
    int departure[n];

    for(int i=0;i<n;i++){
        cin >> arrival[i];
    }
    for(int i=0;i<n;i++){
        cin >> departure[i];
    }
    sort(arrival,arrival+n);
    sort(departure,departure+n);
    map<int,int> m;
    map<int,int>::iterator it;
    for(int i=0;i<n;i++){
        int count = i+1;
        int temp = arrival[i];
        for(int j=0;j<n;j++){
            if(departure[j] <= temp){
                count--;
            }
        }
        m[temp] = count;
    }
    int max = 0;
    for(it=m.begin();it!=m.end();it++){
        if(it->second > max){
            max = it->second;
        }
    }
    cout << max << endl;
}
```

## 5-Ass : Tell the positions

Send Feedback

In a class there are ‘n’ number of students. They have three different subjects: Data Structures, Algorithm Design & Analysis and Operating Systems. Marks for each subject of all the students are provided to you. You have to tell the position of each student in the class. Print the names of each student according to their position in class. Tie is broken on the basis of their roll numbers. Between two students having same marks, the one with less roll number will have higher rank. The input is provided in order of roll number.

### Input Format:

First line will have a single integer ‘n’, denoting the number of students in the class.
Next ‘n’ lines each will have one string denoting the name of student and three space separated integers m1, m2, m3 denoting the marks in three subjects.

### Output Format:

Print ‘n’ lines having two values: First, the position of student in the class and second his name.

### Constraints:

1 <= n <= 10^5
0 <= m1, m2, m3 <= 100

### Sample Input:

3
Mohit 94 85 97
Shubham 93 91 94
Rishabh 95 81 99

### Sample Output:

1 Shubham
2 Mohit

3 Rishabh

```
#include<bits/stdc++.h>
using namespace std;

struct student{
    int roll;
    string name;
    int marks;
};

bool compare(student c1,student c2){
    if(c1.marks > c2.marks){
        return 1;
    }else if(c1.marks < c2.marks){
        return 0;
    }else if(c1.marks == c2.marks){
        if(c1.roll < c2.roll){
            return 1;
        }else{
            return 0;
```

```
        }
    }
}

int main()
{
    long n;
    cin >> n;
    student arr[n];
    int roll = 1;
    for(int i=0;i<n;i++){
        string name;
        int m1,m2,m3;
        cin >> name;
        cin >> m1 >> m2 >> m3;
        int marks = m1+m2+m3;
        arr[i] = {roll,name,marks};
        roll += 1;
    }
    sort(arr,arr+n,compare);// important h, ye dekh lo how the third parameter compare works.
    int count = 0;
    for(int i=0;i<n;i++){
        count++;
        cout << count << " " << arr[i].name << endl;
    }

            return 0;
}
```

# L5 : Language Tools and Time and Space Complexity Practice Questions

## 1-Ass : Rotate array

Send Feedback

You have been given a random integer array/list(ARR) of size N. Write a function that rotates the given array/list by D elements(towards the left).

**Note:**

Change in the input array/list itself. You don't need to return or print the elements.

**Input format :**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.
First line of each test case or query contains an integer 'N' representing the size of the array/list.
Second line contains 'N' single space separated integers representing the elements in the array/list.
Third line contains the value of 'D' by which the array/list needs to be rotated.

**Output Format :**

For each test case, print the rotated array/list in a row separated by a single space.
Output for every test case will be printed in a separate line.

**Constraints :**

1 <= t <= 10^4
0 <= N <= 10^6
0 <= D <= N
Time Limit: 1 sec

**Sample Input 1:**

```
1
7
1 2 3 4 5 6 7
2
```

**Sample Output 1:**

```
3 4 5 6 7 1 2
```

**Sample Input 2:**

```
2
7
1 2 3 4 5 6 7
0
4
1 2 3 4
2
```

**Sample Output 2:**

```
1 2 3 4 5 6 7

3 4 1 2
```

```
1. #include<bits/stdc++.h>
2. void rotate(int *input, int d, int n)
3. {
```

```
    //Write your code here
    reverse(input,input+n);
    reverse(input,input+(n-d));
    reverse(input+(n-d),input+n);
}
```

## 2-Ass : Pair sum to 0

Send Feedback

Given a random integer array A of size N. Find and print the count of pair of elements in the array which sum up to 0.

Note: Array A can contain duplicate elements as well.

### Input format:

The first line of input contains an integer, that denotes the value of the size of the array. Let us denote it with the symbol N.

The following line contains N space separated integers, that denote the value of the elements of the array.

### Output format :

The first and only line of output contains the count of pair of elements in the array which sum up to 0.

### Constraints :

0 <= N <= 10^4

Time Limit: 1 sec

### Sample Input 1:

5

2 1 -2 2 3

### Sample Output 1:

2

```
#include<bits/stdc++.h>
int pairSum(int *arr, int n) {
        // Write your code here
    unordered_map<int,int> mymap;
    for(int i = 0; i < n; i++){
        mymap[arr[i]]++;
    }

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

        if(arr[i] == 0){
        continue;
        }

        int comp = -arr[i];
        if(mymap[comp] != 0){
            count += (mymap[arr[i]] * mymap[comp]);
            mymap[arr[i]] = 0;
```

```
            mymap[comp] = 0;
        }

    }
    int countzero = mymap[0];
    int result = count + ( (countzero)*(countzero-1) )/2;
    return result;
}
```

Note : if less than 20% test case are failing due to TLE then try to optimize the check condition's, mostly it works

## 3-Ass : Longest Consecutive Sequence

Send Feedback

You are given an array of unique integers that contain numbers in random order. You have to find the longest possible sequence of consecutive numbers using the numbers from given array.

You need to return the output array which contains starting and ending element. If the length of the longest possible sequence is one, then the output array must contain only single element.

### Note:

1. Best solution takes O(n) time.
2. If two sequences are of equal length, then return the sequence starting with the number whose occurrence is earlier in the array.

### Input format:

The first line of input contains an integer, that denotes the value of the size of the array. Let us denote it with the symbol n.

The following line contains n space separated integers, that denote the value of the elements of the array.

### Output format:

The first and only line of output contains starting and ending element of the longest consecutive sequence. If the length of longest consecutive sequence, then just print the starting element.

### Constraints :

0 <= n <= 10^6

Time Limit: 1 sec

### Sample Input 1 :

13

2 12 9 16 10 5 3 20 25 11 1 8 6

### Sample Output 1 :

8 12

### Sample Input 2 :

7

3 7 2 1 9 8 41

### Sample Output 2 :

7 9

Explanation: Sequence should be of consecutive numbers. Here we have 2 sequences with same length i.e. [1, 2, 3] and [7, 8, 9], but we should select [7, 8, 9] because the starting point of [7, 8, 9] comes first in input array and therefore, the output will be 7 9, as we have to print starting and ending element of the longest consecutive sequence.

**Sample Input 3 :**

```
7
15 24 23 12 19 11 16
```

**Sample Output 3 :**

```
15 16
```

```
#include<bits/stdc++.h>
vector<int> longestConsecutiveIncreasingSequence(int *arr, int n) {
    // Your Code goes here
    int* dp = new int[n]();
    unordered_map<int,int> m;
    int index = 0;

    for(int i=0;i<n;i++){
        m[arr[i]] = index++; //storing rank for first occurance
    }
    m[-1] = n+1;
    for(int i=0;i<n;i++){
        dp[i] = 1;
    }

    sort(arr,arr+n);
    for(int i=1;i<n;i++){
        if(arr[i-1]+1 == arr[i]){
            dp[i] = dp[i-1] + 1; //if prev element is consecutive
        }
    }

    int maxi = INT_MIN; //max length of subsequence possible
    for(int i=0;i<n;i++){
        maxi = max(maxi,dp[i]);
    }

    vector<int> ans;
    for(int i=0;i<n;i++){
        if(dp[i] == maxi){ //all possible subsequences of length "maximum"
            ans.push_back(arr[i]-dp[i]+1);
        }
    }

    int idx = n+1;
    int finalAns; //among all possible answers onw which has first occurance is original array
             for(int i=0;i<ans.size();i++){
        if(m[ans[i]]<idx){
            idx = m[ans[i]];
            finalAns = ans[i];
        }
    }
```

```
43.
44.     vector<int> v;
45.     v.push_back(finalAns);
46.     v.push_back(finalAns+maxi-1);
47.
48.     return v;
49.
50. }
```

## 4-Ass : Duplicate in array

Send Feedback

You have been given an integer array/list(ARR) of size N which contains numbers from 0 to (N - 2). Each number is present at least once. That is, if N = 5, the array/list constitutes values ranging from 0 to 3, and among these, there is a single integer value that is present twice. You need to find and return that duplicate number present in the array.

### Note :

Duplicate number is always present in the given array/list.

### Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.
First line of each test case or query contains an integer 'N' representing the size of the array/list.
Second line contains 'N' single space separated integers representing the elements in the array/list.

### Output Format :

For each test case, print the duplicate element in the array/list.
Output for every test case will be printed in a separate line.

### Constraints :

1 <= t <= 10^2
0 <= N <= 10^6
Time Limit: 1 sec

### Sample Input 1:

1
9
0 7 2 5 4 7 1 3 6

### Sample Output 1:

7

### Sample Input 2:

2
5
0 2 1 3 1
7
0 3 1 5 4 3 2

### Sample Output 2:

1

3

```
#include<bits/stdc++.h>
int findDuplicate(int *arr, int n)
{
    //Write your code here
    // we can solve it through both way using map as well as set
    map<int,int> m;
    for(int i = 0; i < n; i++){
        m[arr[i]]++;

        if(m[arr[i]] > 1){
            return arr[i];
        }
    }

}
```

Note : Using map ( BST ) TLE is coming while unordered_map ( Hashmap ) all test cases passed.

```
#include<bits/stdc++.h>
int findDuplicate(int *arr, int n)
{
    //Write your code here
    // we can solve it through both way using map as well as set
    unordered_map<int,int> m;
    for(int i = 0; i < n; i++){
        m[arr[i]]++;

        if(m[arr[i]] > 1){
            return arr[i];
        }
    }

}
```

Using set :

```
#include<bits/stdc++.h>
int findDuplicate(int *arr, int n)
{
    //Write your code here
    // we can solve it through both way using map as well as set
    set<int> s;
    for(int i = 0; i < n; i++){
        if(s.find(arr[i]) != s.end()){
            return arr[i];
        }
        s.insert(arr[i]);
    }

}
```

# 5-Ass : Triplet sum

Send Feedback

You have been given a random integer array/list(ARR) and a number X. Find and return the triplet(s) in the array/list which sum to X.

**Note :**

Given array/list can contain duplicate elements.

**Input format :**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.
First line of each test case or query contains an integer 'N' representing the size of the first array/list.
Second line contains 'N' single space separated integers representing the elements in the array/list.
Third line contains an integer 'X'.

**Output format :**

For each test case, print the total number of triplets present in the array/list.

Output for every test case will be printed in a separate line.

**Constraints :**

1 <= t <= 10^2
0 <= N <= 10^3
0 <= X <= 10^9

Time Limit: 1 sec

**Sample Input 1:**

```
1
7
1 2 3 4 5 6 7
12
```

**Sample Output 1:**

```
5
```

**Sample Input 2:**

```
2
7
1 2 3 4 5 6 7
19
9
2 -5 8 -6 0 5 10 11 -3
10
```

**Sample Output 2:**

```
0
5
```

**Explanation for Input 2:**

Since there doesn't exist any triplet with sum equal to 19 for the first query, we print 0.

For the second query, we have 5 triplets in total that sum up to 10. They are, (2, 8, 0), (2, 11, -3), (-5, 5, 10), (8, 5, -3) and (-6, 5, 11)

```
1.  int tripletSum(int *arr, int n, int num)
2.  {
3.              //Write your code here
4.      sort(arr,arr+n);
5.      int numTriplets = 0;
6.
7.      for(int i =0; i < n-2; i++)
8.      {
9.          int l = i+1;
10.         int r = n-1;
11.         int count = 0;
12.         while(l < r){
13.             if(arr[i] + arr[l] + arr[r] == num){
14.
15.                 if (arr[l] == arr[r])
16.                 {
17.                     int totalElementsFromStartToEnd = (r -l) + 1;
18.                     count += (totalElementsFromStartToEnd * (totalElementsFromStartToEnd - 1) / 2);
19.                     break;
20.                 }
21.                 int tempStartIndex = l + 1;
22.                 int tempEndIndex = r - 1;
23.                 while (tempStartIndex <= tempEndIndex && arr[tempStartIndex] == arr[l])
24.                 {
25.                     tempStartIndex += 1;
26.
27.                 }
28.                 while (tempEndIndex >= tempStartIndex && arr[tempEndIndex] == arr[r])
29.                 {
30.                     tempEndIndex -= 1;
31.                 }
32.                 int totalElementsFromStart = (tempStartIndex - l);
33.                 int totalElementsFromEnd = (r - tempEndIndex);
34.                 count += (totalElementsFromStart * totalElementsFromEnd);
35.                 l = tempStartIndex; r = tempEndIndex;
36.
37.             }else if(arr[i] + arr[l] + arr[r] < num){
38.                 l++;
39.             }
40.             else{
41.                 r--;
42.             }
43.         }
44.         numTriplets+=count;
45.     }
46.
47.   return numTriplets;
48. }
```

## 6-Ass : Find the Unique Element

Send Feedback

You have been given an integer array/list(ARR) of size N. Where N is equal to [2M + 1].

Now, in the given array/list, 'M' numbers are present twice and one number is present only once.

You need to find and return that number which is unique in the array/list.

**Note:**

Unique element is always present in the array/list according to the given condition.

**Input format :**

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.
First line of each test case or query contains an integer 'N' representing the size of the array/list.
Second line contains 'N' single space separated integers representing the elements in the array/list.

**Output Format :**

For each test case, print the unique element present in the array.
Output for every test case will be printed in a separate line.

**Constraints :**

1 <= t <= 10^2
0 <= N <= 10^6
Time Limit: 1 sec

**Sample Input 1:**

```
1
7
2 3 1 6 3 6 2
```

**Sample Output 1:**

```
1
```

**Sample Input 2:**

```
2
5
2 4 7 2 7
9
1 3 1 3 6 6 7 10 7
```

**Sample Output 2:**

```
4

10
```

```
1.  int findUnique(int *arr, int n) {
2.      // Write your code here
3.      int res = arr[0];
4.      for(int i = 1;  i < n; i++){
5.          res = res^arr[i];
6.      }
7.      return res;
8.  }
```

# L6: Searching and Sorting Practice Questions

## 1-Tut : Aggressive Cows Problem

Send Feedback

Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000).

His C (2 <= C <= N) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, FJ wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance ?

### Input Format:

The first line of input contains a number of test cases, which will be denoted by the symbol t. In the following lines, the t test cases are written.

The first line of each of the test cases contain two space separated integers: N and C. The following line contains N space separated integers, where the ith integer denotes the position of ith stall.

It is given that the sum of N over all the test cases doesn't exceed 1000, 000.

### Output Format:

As described in the problem statement, the first and only line of output will print the largest minimum distance possible.

The output for each test case will be printed on a separate line.

### Sample Input 1:

```
2
11 3
15 5 2 4 17 16 12 8 19 18 11
15 13
20 8 16 3 13 9 11 10 15 17 18 14 1 2 5
```

### Sample Output 1:

```
8

1
```

```
#include<bits/stdc++.h>
using namespace std;

bool check(int cows,long long positions[],int n,long long distance){

        int count = 1;
        long long last_position = positions[0];

        for(int i=1;i<n;i++){
                if(positions[i] - last_position >= distance){
                        last_position = positions[i];
                        count++;
                }
```

```
14.
15.                     if(count == cows){
16.                             return true;
17.                     }
18.             }
19.             return false;
20. }
21.
22. int main(){
23.             int t;
24.             cin >> t;
25.             while(t--){
26.                     int n,c;
27.                     cin >> n >> c;
28.
29.                     long long positions[n];
30.                     for(int i=0;i<n;i++){
31.                             cin >> positions[i];
32.                     }
33.                     sort(positions,positions+n);
34.                     long long start = 0;
35.                     long long end = positions[n-1] - positions[0];
36.
37.                     long long ans = -1;
38.
39.                     while(start<=end){
40.                             long long mid = start + (end-start)/2;
41.
42.                             if(check(c,positions,n,mid)){
43.                                     ans = mid;
44.                                     start = mid+1;
45.                             }else{
46.                                     end = mid-1;
47.                             }
48.
49.                     }
50.                     cout << ans <<endl;
51.             }
52.
53.             return 0;
54. }
```

## 2-Tut : Inversion Count

Send Feedback

For a given integer array/list 'ARR' of size 'N', find the total number of 'Inversions' that may exist.

An inversion is defined for a pair of integers in the array/list when the following two conditions are met.

A pair ('ARR[i]', 'ARR[j]') is said to be an inversion when:
1. 'ARR[i] > 'ARR[j]'
2. 'i' < 'j'

Where 'i' and 'j' denote the indices ranging from [0, 'N').

### Input format :

The first line of input contains an integer 'N', denoting the size of the array.
The second line of input contains 'N' integers separated by a single space, denoting the elements of the array 'ARR'.

### Output format :

Print a single line containing a single integer that denotes the total count of inversions in the input array.

### Note:

You are not required to print anything, it has already been taken care of. Just implement the given function.

### Constraints :

1 <= N <= 10^5
1 <= ARR[i] <= 10^9

Where 'ARR[i]' denotes the array element at 'ith' index.
Time Limit: 1 sec

### Sample Input 1 :

3
3 2 1

### Sample Output 1 :

3

### Explanation of Sample Output 1:

We have a total of 3 pairs which satisfy the condition of inversion. (3, 2), (2, 1) and (3, 1).

### Sample Input 2 :

5
2 5 1 3 4

### Sample Output 2 :

4

### Explanation of Sample Output 1:

We have a total of 4 pairs which satisfy the condition of inversion. (2, 1), (5, 1), (5, 3) and (5, 4).

```
1.  #include<iostream>
2.  using namespace std;
3.  long long merge(long long *A,int left,int mid,int right){
4.
5.              int i=left,j=mid,k=0;
```

```
6.
7.              int temp[right-left+1];
8.              long long count = 0;
9.              while(i<mid && j<=right){
10.                     if(A[i] <= A[j]){
11.                             temp[k++] = A[i++];
12.                     }else{
13.                             temp[k++] = A[j++];
14.                             count += mid - i;
15.                     }
16.             }
17.             while(i<mid){
18.                     temp[k++] = A[i++];
19.             }
20.             while(j<=right){
21.                     temp[k++] = A[j++];
22.             }
23.
24.             for(int i=left,k=0;i<=right;i++,k++){
25.                     A[i] = temp[k];
26.             }
27.             return count;
28. }
29. long long merge_sort(long long *A,int left,int right){
30.
31.             long long count = 0;
32.             if(right > left){
33.                     int mid = (left + right)/2;
34.
35.                     long long countLeft = merge_sort(A,left,mid);
36.                     long long countRight = merge_sort(A,mid+1,right);
37.                     long long myCount = merge(A,left,mid+1,right);
38.
39.                     return myCount + countLeft + countRight;
40.             }
41.             return count;
42.
43. }
44.
45. long long getInversions(long long *arr, int n){
46.     // Write your code here.
47.     long long ans = merge_sort(arr,0,n-1);
48.             return ans;
49. }
```

Live Problem -

1. https://www.codechef.com/problems/CHEFRES
2. https://www.codechef.com/ZCOPRAC/problems/ZCO15002

3-Ass : **Murder**

Send Feedback

Once detective Saikat was solving a murder case. While going to the crime scene he took the stairs and saw that a number was written on every stair. He found it suspicious and decides to remember all the numbers that he has seen till now. While remembering the numbers he found that he can find some pattern in those numbers. So he decides that for each number on the stairs he will note down the sum of all the numbers previously seen on the stairs which are smaller than the present number. Calculate the sum of all the numbers written in his notes diary.

Answers may not fit in integers . You have to take a long type.

### Input Format:

First line of input contains an integer T, representing the number of test cases.
For each test case, the first line of input contains integer N, representing the number of stairs.
For each test case, the second line of input contains N space separated integers, representing numbers written on the stairs.

### Constraints

T<=10^5
1<=N<=10^5
All numbers will be between 0 and 10^9
Sum of N over all test cases doesn't exceed 5*10^5

### Output Format

For each test case output one line giving the final sum for each test case.

### Sample Input 1:

1
5
1 5 3 6 4

### Sample Output 1:

15

### Explanation:

For the first number, the contribution to the sum is 0.
For the second number, the contribution to the sum is 1.
For the third number, the contribution to the sum is 1.
For the fourth number, the contribution to the sum is 9 (1+5+3).
For the fifth number, the contribution to the sum is 4 (1+3).

Hence the sum is 15 (0+1+1+9+4).

```
#include<iostream>
using namespace std;
typedef long long ll;

ll merge(ll *arr, ll left, ll mid, ll right)
{
        ll i=left;
        ll k=0;
        ll j=mid;
```

```
	ll sum=0;
	ll *temp=new ll [right-left+1];
	while(i<mid && j<=right)
	{
		if(arr[i]<arr[j])
		{
			sum+=(arr[i]*(right-j+1));
			temp[k++]=arr[i++];
		}
		else
		{
			temp[k++]=arr[j++];
		}
	}

	while(i<mid)
	{
		temp[k++]=arr[i++];
	}

	while(j<=right)
	{
		temp[k++]=arr[j++];
	}

	for(ll i=left, k=0; i<=right; i++, k++)
	{
		arr[i]=temp[k];
	}

	delete[]temp;
	return sum;
}

ll merge_sort(ll *arr, ll left, ll right)
{
	ll count=0;
	if(right>left)
	{
		ll mid=(right+left)/2;
		ll left_count=merge_sort(arr, left, mid);
		ll right_count=merge_sort(arr, mid+1, right);
		ll merge_count=merge(arr, left, mid+1, right);

		return left_count+right_count+merge_count;
	}
	return count;
}

```

```
59. ll getAns(ll *arr, ll n)
60. {
61.         ll ans=merge_sort(arr, 0, n-1);
62.         return ans;
63. }
64.
65. int main()
66. {
67.         ll t;
68.         cin>>t;
69.         while(t--)
70.         {
71.                 ll n;
72.                 cin>>n;
73.                 ll *arr=new ll [n];
74.                 for(ll i=0; i<n; i++)
75.                 {
76.                         cin>>arr[i];
77.                 }
78.                 cout<<getAns(arr, n)<<endl;
79.         }
80. }
```

## 4-Ass : Distribute Candies

Send Feedback

Shaky has N (1<=N<=50000) candy boxes each of them contains a non-zero number of candies (between 1 and 1000000000). Shaky want to distribute these candies among his K (1<=K<=1000000000) IIIT-Delhi students. He wants to distribute them in a way such that:

1. All students get an equal number of candies.

2. All the candies which a student gets must be from a single box only.

As he wants to make all of them happy so he wants to give as many candies as possible. Help Shaky in finding out what is the maximum number of candies which a student can get.

### Input

First-line contains T the number of test cases.
The first line of each test case contains N and K.
Next line contains N space-separated integers, ith of which is the number of candies in the ith box.

### Output

For each test case print the required answer in a separate line.

### Constraints

1<= T <= 10^5
1 <= N <= 10^5
1 <= K <= 10^9
0 <= A[i] <= 10^9
Sum of N over all test cases doesn't exceed 10^6

### Sample Input:

2
3 2
3 1 4
4 1
3 2 3 9

**Sample Output:**

3

9

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

bool check(ll arr[],ll n,ll k,ll range){
    ll count = 0;
    for(ll i = 0;i<n;i++){
        count += arr[i]/range;  //possibility of box can accommodate more than one student
        if(count >=k){
            return true;
        }
    }
    return false;
}
int main() {
    int t;
    cin >> t;
    while(t--){
        ll n,k;
        cin >> n >> k;
        ll arr[n];
        for(ll i=0;i<n;i++){
            cin >> arr[i];
        }
        sort(arr,arr+n);
        ll start = 0;
        ll end = arr[n-1];
        ll ans = -1;
        while(start <= end){
            ll mid = (start + end )/2;
            if(check(arr,n,k,mid)){
                ans = mid;
                start = mid+1;
            }else{
                end = mid -1;
            }
        }
        cout << ans << endl;
    } }
```

## 5-Ass : Momos Market

Send Feedback

Shreya loves to eat momos. Her mother gives her money to buy vegetables but she manages to save some money out of it daily. After buying vegetables, she goes to "Momo's Market", where there are ‘n’ number of momos' shops of momos. Each of the shops of momos has a rate per momo. She visits the market and starts buying momos (one from each shop) starting from the first shop. She will visit the market for ‘q’ days. You have to tell that how many momos she can buy each day if she starts buying from the first shop daily. She cannot use the remaining money of one day on some other day. But she will save them for other expenses in the future, so, you also need to tell the sum of money left with her at the end of each day.

### Input Format:

First-line will have an integer ‘n’ denoting the number of shops in the market.
The next line will have ‘n’ numbers denoting the price of one momo of each shop.
The next line will have an integer ‘q’ denoting the number of days she will visit the market.
Next ‘q’ lines will have one integer ‘X’ denoting the money she saved after buying vegetables.

### Constraints:

1 <= n <= 10^5
1 <= q <= 10^4
1 <= X <= 10^9
1 <= rate per momo <=10^9 (for each shop)

### Output:

There will be ‘q’ lines of output each having two space separated integers denoting number of momos she can buy and amount of money she saved each day.

### Sample Input:

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

### Sample Output:

```
3 2
```

### Explanation:

Shreya visits the "Momo's Market" for only one day. She has 11 INR to spend. She can buy 3 momos, each from the first 3 shops. She would 9 INR (2 + 1 + 6) for the same and hence, she will save 2 INR.

```
1.  #include<iostream>
2.
3.  using namespace std;
4.  typedef long long int ll;
5.  bool checker(ll *arr, ll n, ll mid, ll target)
6.  {
7.          if(arr[mid]<=target)        {
8.                  return true;
9.          }
```

```
10.             return false;
11.   }
12.   void momos(ll *arr, ll n, ll target)
13.   {
14.             ll start=0;
15.             ll end=n-1;
16.             ll mid;
17.             while(start<end)
18.             {
19.                       mid=(start+end)/2;
20.                       if(checker(arr, n, mid, target))
21.                       {
22.                                 start=mid;
23.                       }
24.                       else
25.                       {
26.                                 end=mid;
27.                       }
28.             if(end-start==1)
29.                       {
30.                                 if(checker(arr, n, end, target))
31.                                 {
32.                                           cout<<end+1<<" "<<target-arr[end]<<endl;
33.                                           return;
34.                                 }
35.                                 else
36.                                 {
37.                  if(!checker(arr, n, start, target))
38.                  {
39.                     cout<<0<<" "<<target<<endl;
40.                     return;
41.                  }
42.                                           cout<<start+1<<" "<<target-arr[start]<<endl;
43.                                           return;
44.                                 }
45.                       }
46.             }
47.      cout<<mid+1<<" "<<target-arr[mid]<<endl;
48.      return;
49.   }
50.   int main()
51.   {
52.             ll n;
53.             cin>>n;
54.             ll *prices=new ll [n];
55.             for(ll i=0; i<n; i++)
56.             {
57.                       cin>>prices[i];
58.             }
```

```
59.         ll q;//number of days.
60.         cin>>q;
61.         ll *money=new ll [q];
62.         for(ll i=0; i<q; i++)
63.         {
64.                 cin>>money[i];
65.         }
66.         ll *prefix_sum_prices=new ll [n];
67.         ll sum=0;
68.         for(ll i=0; i<n; i++)
69.         {
70.         sum+=prices[i];
71.                 prefix_sum_prices[i]=sum;
72.         }
73.         for(ll i=0; i<q; i++)
74.         {
75.         momos(prefix_sum_prices, n, money[i]);
76.         }
77.  }
```

## 6-Ass : Taj Mahal Entry

Send Feedback

Taj Mahal is one of the seven wonders of the world. Aahad loves to travel places and wants to visit the Taj Mahal. He visited Agra to view the Taj Mahal. There is a ticketing system at Taj Mahal. There are total ‘n’ windows which provide the tickets to get entry into Taj Mahal. There are ‘Ai’ people already present at each window to get the tickets. Each window gives a ticket to one person in one minute. Initially, Aahad stands in front of the first window. After each minute, if he didn’t get the ticket, he moves on to the next window to get the ticket. If he is at window 1, he will move to 2. If at 2nd, he will move to 3rd. If he is at last window, he will move to 1st again and so on. Find the window number at which he will get the ticket.

### Input Format:

First line contains a single integer ‘n’ denoting the no. of windows.
Next line contains ‘n’ space separated integers denoting the no. of people already standing in front of the ith window. (1 <= i <= n)

### Output Format:

Print a single integer denoting the window number that Aahad will get the ticket from.

### Constraints:

1 <= n <= 10^5
1 <= Ai <= 10^9

### Sample Input:

4
2 3 2 0

### Sample Output:

3

### Explanation:

Aahad at Window 1: [2, 3, 2, 0]

Aahad at Window 2: [1, 2, 1, 0]
Aahad at Window 3: [0, 1, 0, 0]

So, when Aahad is at window 3, he got zero people before him. Hence, he will get the ticket at window 3.

```
#include<iostream>
using namespace std;
typedef long long int ll;
int main()
{
	int n;
	cin>>n;
	int *arr=new int [n];
	for(int i=0; i<n; i++)
	{
		cin>>arr[i];
	}
	int*t=new int [n];
	for(int i=0; i<n; i++)
	{
		if((arr[i]-i)%n==0)
		{
			t[i]=(arr[i]-i)/n;
		}
    else if(arr[i]-i<0)
    {
       t[i]=0;
    }
		else
		{
			t[i]=((arr[i]-i)/n)+1;
		}
	}
	int *ans=new int [n];
	int minimum=9999999999;
	int minimum_index=-1;
	for(int i=0; i<n; i++)
	{
		ans[i]=i+t[i]*n;
		if(ans[i]<minimum)
		{
			minimum=ans[i];
			minimum_index=i;
		}
	}
	cout<<minimum_index+1;

}
```

Method-2 :

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin>>n;
    int w[n];
    for(int i=0;i<n;i++){
        cin>>w[i];
    }
    int del=0;
    for(int i=0;;i++){
        i=i%n;
        w[i]-=del;
        del++;
        if(w[i]<=0){
            cout<<i+1<<endl;
            break;
        }
    }
        return 0;
}
```

## 7-Ass : Sorting the Skills

Send Feedback

There is a company named James Peterson & Co. The company has ‘n’ employees. The employees have skills, which is denoted with the help of an integer. The manager of James Peterson & Co. wants to sort the employees on the basis of their skills in ascending order. He is only allowed to swap two employees which are adjacent to each other. He is given the skills of employees in an array of size ‘n’. He can swap the skills as long as the absolute difference between their skills is 1. You need to help the manager out and tell whether it is possible to sort the skills of employees or not.

### Input Format:

First Line will have an integer ‘t’ denoting the no. of test cases.
First line of each test case contains an integer ‘n’ denoting the no. of employees in the company.
Second line of each test case contains ‘n’ integers.

### Output Format:

For each test case, print “Yes” if it is possible to sort the skills otherwise “No”.

### Constraints:

1 <= t <= 10^4
1 <= n <= 10^5
Sum of n over all test cases doesn't exceed 10^6

### Sample Input:

2
4

1 0 3 2
3
2 1 0

**Sample Output:**

Yes
No

**Explanation:**

In first T.C., [1, 0, 3, 2] -> [0, 1, 3, 2] -> [0, 1, 2, 3]

In second T.C., [2, 1, 0] -> [1, 2, 0] OR [2, 1, 0] -> [2, 0, 1] So, it is impossible to sort.

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
        int t;
    cin >> t;
        while(t--)
        {
                int n;
      cin >> n;
                vector<int> a(n);
                for(int i = 0 ; i < n ; ++i) {
                        cin >> a[i];
                }

                string ans = "Yes";

                for(int i = 0 ; i < n-1 ; ++i) {
                        if(a[i] > a[i+1]) {
                                if(a[i] - a[i+1] == 1) {
                                        swap(a[i], a[i+1]);
                                }
                                else {
                                        ans = "No";
                                }
                        }
                }

                cout << ans << '\n';
        }
        return 0;
}
```

## 8-Ass : Collecting the balls

Send Feedback

There are ‘n’ numbers of balls in a container. Mr. Sharma and Singh want to take balls out from the container. At each step, Mr. Sharma took ‘k’ balls out of the box and Mr. Singh took one-tenth of the remaining balls. Suppose there are 29 balls at the moment and k=4. Then, Mr. Sharma will take 4 balls and Mr. Singh will take 2 balls (29-4 = 25; 25/10 = 2). If there are less than ‘k’ balls remaining at some moment, then Mr. Sharma will take all the balls which will get the container empty. The process will last until the container becomes empty. Your task is to choose a minimal ‘k’ for Mr. Sharma such that Mr. Sharma will take at least half of the balls from the container.

### Input Format:

The first line of input will contain T (number of test cases).
The next n lines of input contain a single integer ‘n’.

### Output Format:

For every test case print a single integer denoting the minimal value of ‘k’ in a newline.

### Constraints:

1 <= T <= 10^4
1 <= n <= 10^18
Time Limit: 1 second

### Sample Input:

1
68

### Sample Output:

3

### Explanation:

68-3 = 65; 65/10 = 6; 65-6 = 59
59-3 = 56; 56/10 = 5; 56-5 = 51
51-3 = 48; 48/10 = 4; 48-4 = 44
44-3 = 41; 41/10 = 4; 41-4 = 37
…..
…..
…..
6-3 = 3; 3/10 = 0; 3-0 = 3

3-3 = 0; 0/10 = 0; 0-0 = 0

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

bool check(const int& k, const int& n) {
        int sharma = 0, singh = 0;

        int x = n, step = 1;
        while(x) {
```

```
                // sharma's turn
                sharma += min(k, x);
                x -= min(k, x);

                // singh's turn
                singh -= (x / 10);
                x -= (x / 10);
        }
        return (2*sharma >= n);
}

int32_t main()
{
        ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

        int t; cin >> t;
        while(t--)
        {
                int n; cin >> n;

                int ans = n;

                int left = 1, right = n;
                while(left <= right) {
                        int mid = left + (right - left) / 2;
                        if(check(mid, n)) {
                                ans = mid;
                                right = mid - 1;
                        }
                        else {
                                left = mid + 1;
                        }
                }
                cout << ans << '\n';
        }
        return 0;
}
```

Note : when replacing with int it is giving TLE for some test cases

# L7 : Advanced Recursion Practice Questions

## 1-Tut : Replace Character Recursively

Send Feedback

Given an input string S and two characters c1 and c2, you need to replace every occurrence of character c1 with character c2 in the given string.

Do this recursively.

### Input Format :

The first line of input will contain an integer T, which will denote the value of number of test cases. In the following lines, T test cases will be written.
The first line of each test case will contain a string S.
The following line of each test case will contain two space separated characters, c1 and c2, respectively.

### Output Format :

For each test case, the first and only line of output contains the updated string S.

### Constraints :

1 <= T <= 100
1 <= Length of String S <= 100

### Sample Input :

```
1
abacd
a x
```

### Sample Output :

```
xbxcd
```

```
#include<bits/stdc++.h>
using namespace std;
void replacechar(char s[], char c1, char c2)
{
    if(s[0] == '\0'){
        return;
    }
    if(s[0] == c1){
        s[0] = c2;
    }
    replacechar(s+1,c1,c2);

}
int main(){

    // write your code here
    int T; cin >> T;
    while(T--)
    {
        char s[100]; cin >> s;
```

```
        char c1,c2;
        cin >> c1 >> c2;
        replacechar(s,c1,c2);
        cout << s << endl;

    }
    return 0;
}
```

## 2-Tut : Remove Duplicates Recursively

Send Feedback

Given a string S, remove consecutive duplicates from it recursively.

### Input Format :

First line of input will contain T number of test cases
Next T line will contain the string S

### Output Format :

For every test case print the answer in a separate line

### Constraints :

1 <= T <= 10
1 <= |S| <= 10^4
where |S| represents the length of string

### Sample Input 1 :

1
aabccba

### Sample Output 1 :

abcba

### Sample Input 2 :

1
xxxyyyzwwzzz

### Sample Output 2 :

xyzwz

```
#include<bits/stdc++.h>
using namespace std;
void removeconsduplicate(char s[]){
    if (s[0] == '\0'){
        return;
    }
    if(s[0] == s[1]){
        int i=0;
        while(s[i] != '\0')
        {
            s[i]=s[i+1];
            i++;
        }
```

```
        removeconsduplicate(s);
    }
     removeconsduplicate(s+1);

}
int main(){

    // write your code here
    int T; cin >> T;
    while(T--)
    {
        char s[10000];
        cin >> s;
        removeconsduplicate(s);
        cout << s << endl;

    }
    return 0;
}
```

## 3-Tut : Merge Sort Code

Send Feedback

Sort an array A using Merge Sort.

Change in the input array itself. So no need to return or print anything.

### Input format :

First line of input will contain T number of test cases
First line of every input will contain a single integer N size of the input array.
second line of each input will contain N space-separated integers.

### Output format :

For every test case print, array elements in increasing order (separated by space) in a separate line.

### Constraints :

1 <= T <= 10
1 <= n <= 10^5

### Sample Input 1 :

1
6
2 6 8 5 4 3

### Sample Output 1 :

2 3 4 5 6 8

### Sample Input 2 :

1
5
2 1 5 2 3

### Sample Output 2 :

1 2 2 3 5

```
#include<bits/stdc++.h>
using namespace std;
void merge(int input[], int si, int ei){

    int mid = (si + ei)/2;
    int size = ei -si +1;
    int arr[size];
    int i = si, j = mid +1, k = 0;

    while( i <= mid && j <= ei){

        if(input[i] <= input[j]){
           arr[k++] = input[i++];
        }
        else if(input[j] < input[i]){
            arr[k++] = input[j++];
        }
    }

    while(i <= mid){
        arr[k++] = input[i++];
    }
    while(j <= ei){
        arr[k++] = input[j++];
    }
    i = si;
    j = 0;
    while(j < k){
        input[i++] = arr[j++];
    }
}


void mergesort(int arr[],int si,int ei){
    if(si >= ei){
        return;
    }
    int mid = (si+ei)/2;
    mergesort(arr,si,mid);
    mergesort(arr,mid+1,ei);

    merge(arr,si,ei);

}
int main(){

    // write your code here
    int T; cin >> T;
```

```
50.
51.     while(T--){
52.         int N; cin >> N;
53.         int arr[N];
54.
55.         for(int i = 0; i < N; i++){
56.             cin >> arr[i];
57.         }
58.         mergesort(arr,0,N-1);
59.         for(int i = 0; i < N; i++){
60.             cout << arr[i] << " ";
61.         }
62.         cout << endl;
63.     }
64.     return 0;
65. }
```

## 4-Tut : Quick Sort Code

Send Feedback

Sort an array A using Quick Sort.

Change in the input array itself. So no need to return or print anything.

### Input format :

First line will contain T number of test cases and each test case will consist of two lines.
Line 1 : Integer n i.e. Array size
Line 2 : Array elements (separated by space)

### Output format :

for every test case print array elements in increasing order (separated by space) in a new line.

### Constraints :

1 <= T <= 10
1 <= n <= 10^5
0 <= arr[i] <= 10^9

### Sample Input 1 :

1
6
2 6 8 5 4 3

### Sample Output 1 :

2 3 4 5 6 8

### Sample Input 2 :

1
5
1 5 2 7 3

### Sample Output 2 :

1 2 3 5 7

```
#include<bits/stdc++.h>
using namespace std;
int partition(int arr[],int si,int ei){

    int pivot = arr[si];
    int countsmaller = 0;
    int i = si+1;

    while(i <= ei){
        if(arr[i] <= pivot){
            countsmaller++;
        }
        i++;
    }

    int pi = si+countsmaller;
    arr[si] = arr[pi];
    arr[pi] = pivot;
    i = si;
    int j  = ei;

    while(i < pi && j > pi){
        if(arr[i] <= pivot){
            i++;
        }
        else if(arr[j] > pivot){
            j--;
        }else{
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
            j--;
        }

    }
    return pi;
}

void quicksort(int arr[], int si, int ei){
    if(si >= ei){
        return;
    }

    int pi = partition(arr,si,ei);
    quicksort(arr,si,pi-1);
    quicksort(arr,pi+1,ei);
}

```

```
int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int N;cin >> N;
        int arr[N];
        for(int i = 0; i < N; i++){
            cin >> arr[i];
        }
        quicksort(arr,0,N-1);
        for(int i = 0; i < N; i++){
            cout << arr[i] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

## 5-Tut : Return Keypad Code

Send Feedback

Given an integer n, using phone keypad find out all the possible strings that can be made using digits of input n.

The numbers and their corresponding codes are given below:

0: ""
1: ""
2: "abc"
3: "def"
4: "ghi"
5: "jkl"
6: "mno"
7: "pqrs"
8: "tuv"
9: "wxyz"

Return empty string for numbers 0 and 1.

Note:

1. The order of strings are not important.
2. The input number will have digits between: [2, 9].

### Input Format :

First line of input will contain T number of test cases.
Each input consists of a single line containing an integer n.

### Output Format :

For each test case, print all possible strings in different lines.

### Constraints :

1 <= T <= 100

1 <= n <= 10^6

**Sample Input:**

1
23

**Sample Output:**

ad
ae
af
bd
be
bf
cd
ce
cf

```
#include<bits/stdc++.h>
using namespace std;
string options[] = {" "," ", "abc", "def", "ghi", "jkl", "mno","pqrs","tuv","wxyz"};
int keypad(int num, string output[]){
    /* Insert all the possible combinations of the integer number into the output string array. You do not need to
    print anything, just return the number of strings inserted into the array.
    */
    if(num == 0 || num == 1){
        return 1;
    }

    int ld = num % 10;
   //  num = num/10;
    int smallanssize = keypad((num / 10 ), output);

    int k = 0;
    for(int i = 0; i < options[ld].size(); i++){

        for(int j = 0; j < smallanssize; j++){
            output[k] = output[j];
            k++;
        }
    }

    k = 0;
    for(int i = 0; i < options[ld].size(); i++){

        for(int j = 0; j < smallanssize; j++){
            output[k] = output[k] + options[ld][i];
            k++;
        }
    }
```

```
33.
34.     return ((options[ld].size()) * smallanssize);
35. }
36.
37. int main(){
38.
39.     // write your code here
40.     int T; cin >> T;
41.     while(T--){
42.     int n; cin >> n;
43.
44.     string output[10000];
45.     int count = keypad(n, output);
46.     for(int i = 0; i < count && i < 10000; i++){
47.         cout << output[i] << endl;
48.     }
49.     }
50.     return 0;
51. }
```

## 6-Tut : Print Keypad Combinations Code

Send Feedback

Given an integer n, using phone keypad find out and print all the possible strings that can be made using digits of input n.

**Note : The order of strings are not important. Just print different strings in new lines.**

**Input Format :**

Integer n

**Output Format :**

All possible strings in different lines

**Constraints :**

1 <= n <= 10^6

**Sample Input:**

23

**Sample Output:**

ad
ae
af
bd
be
bf
cd
ce
cf

```
#include <iostream>
#include <string>
using namespace std;
string options[] = {" "," ", "abc", "def", "ghi", "jkl", "mno","pqrs","tuv","wxyz"};
void printKeypadhelper(int num , string output){

    if(num == 0){
        cout << output << endl;
        return;
    }
    int ld = num%10;
    int smallnum = num / 10;
    string option = options[ld];
    for(int i = 0; i < option.size(); i++){

        printKeypadhelper(smallnum, option[i] + output);

    }
}


void printKeypad(int num){
    /*
    Given an integer number print all the possible combinations of the keypad. You do not need to return anything just print them.
    */
    string output = "";
    printKeypadhelper(num,output);

}
```

# L8: BackTracking Practice Questions

## 1-Tut : N-Queen Problem

Send Feedback

You are given N, and for a given N x N chessboard, find a way to place N queens such that no queen can attack any other queen on the chess board. A queen can be killed when it lies in the same row, or same column, or the same diagonal of any of the other queens. You have to print all such configurations.

Note: Don't print anything if there isn't any valid configuration.

### Input Format:

The first line of input contains an integer, that denotes the value of N.

### Output Format :

In the output, you have to print every board configuration in a new line. Every configuration will have N*N board elements printed row wise and are separated by space. The cells where queens are safe and placed, are denoted by value 1 and remaining all cells are denoted by value 0. Please refer to sample test cases for more clarity.

### Constraints :

1 <= N <= 10

Time Limit: 1 second

### Sample Input 1:

4

### Sample Output 1 :

0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0

0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0

### Explanation:

The first and second configurations are shown in the image below. Here, 1 denotes the placement of a queen and 0 denotes an empty cell. Please note that in both the configurations, no pair of queens can kill each other.

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |

First Configuration

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |

Second Configuration

```
#include<bits/stdc++.h>
using namespace std;

void print(int **board,int n){
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cout<<board[i][j]<<" ";
        }
    }
    cout<<endl;
}

bool ispossible(int **board,int n,int i,int j){
    //check col
    for(int row=0;row<n;row++){
        if(board[row][j]==1){
            return false;
        }
    }
    //check row
    for(int col=0;col<n;col++){
        if(board[i][col]==1){
            return false;
        }
    }
    //check diagonal left bend top
    for(int row=i,col=j;row>=0&&col>=0;row--,col--){
        if(board[row][col]==1){
            return false;
        }
    }
    //check diagonal right bend top
    for(int row=i,col=j;row<n&&row>=0&&col>=0&&col<n;row--,col++){
        if(board[row][col]==1){
            return false;
        }
    }
    return true;
}

void find_all_possible(int **board,int n,int i,int j){
    //crossed limits
    if(i==n){
        print(board,n);
    }
    //check for all cols of a particular row
        for(int col=j;col<n;col++){ //col
            if(ispossible(board,n,i,col)){
                board[i][col]=1;
```

```
                find_all_possible(board,n,i+1,0);
                board[i][col]=0;
            }
        }
}

void placeNQueens(int n){
    int **board=new int*[n];
    for(int i=0;i<n;i++){
        board[i]=new int[n];
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            board[i][j]=0;
        }
    }
            find_all_possible(board,n,0,0);
    for(int i=0;i<n;i++){
        delete board[i];
    }
    delete board;
}

int main(){

   int n;
   cin >> n ;
   placeNQueens(n);

}
```

## 2-Tut : Rat In A Maze Problem

Send Feedback

You are given an integer N and a binary 2D array of size N*N. The given binary matrix represents a maze and a rat starts from the top left cell and has to reach to the bottom right cell of the maze. The rat can move in all the four directions: up, down, left and right.

In the given binary matrix, 0 signifies the cell is a dead end and 1 signifies the cell can be used to move from source to destination cell.

You have to print all the paths, following which rat can move from the top left cell to the bottom right cell of the given binary matrix.

### Input Format

The first line of input contains an integer that denotes the value of N.

Each of the following N lines denote rows of the binary matrix and contains either 0 or 1. Each row of the binary matrix contains N elements.

### Output Format :

Every possible solution has to be printed on a separate line. Each of the possible solution will have N*N elements, printed row wise and separated by space. The cells that are part of the path should be 1 and remaining all cells should be 0. Please refer to sample test cases for more clarity.

### Constraints

1 <= N <= 18

0 <= Number of cells with value 1 <= 15

Time Limit: 1 second

### Sample Input 1 :

3

1 0 1

1 0 1

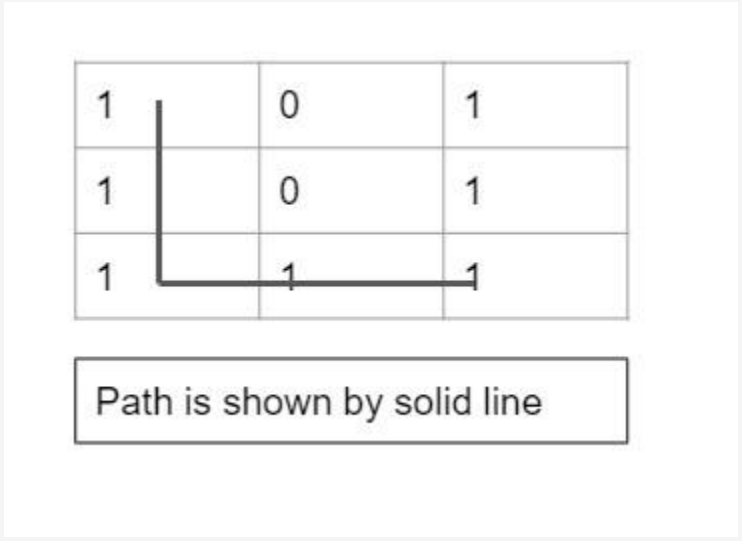
1 1 1

### Sample Output 1 :

1 0 0 1 0 0 1 1 1

### Explanation :

Only 1 path is possible. The path is shown in the image below.




### Sample Input 2 :

3

1 0 1

1 1 1

1 1 1

### Sample Output 2 :

1 0 0 1 1 1 1 1 1

1 0 0 1 0 0 1 1 1

1 0 0 1 1 0 0 1 1

1 0 0 1 1 1 0 0 1

### Explanation:

As described in the Sample Output 2, four paths are possible.

```
/*
        Note:
        To get all the test cases accepted, make recursive calls in following order: Top, Down, Left, Right.
        This means that if the current cell is (x, y), then order of calls should be: top cell (x-1, y), down cell (x+1, y), left cell (x, y-1) and right cell (x, y+1).
*/
#include<bits/stdc++.h>
using namespace std;


void printSolution(int** solution,int n){
        for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                        cout << solution[i][j] << " ";
                }
        }
        cout<<endl;
}
void mazeHelp(int maze[][20],int n,int** solution,int x,int y){


        if(x == n-1 && y == n-1){
                solution[x][y] =1;
                printSolution(solution,n);
        solution[x][y] =0;
                return;
        }
        if(x>=n || x<0 || y>=n || y<0 || maze[x][y] ==0 || solution[x][y] ==1){
                return;
        }
        solution[x][y] = 1;
        mazeHelp(maze,n,solution,x-1,y);
        mazeHelp(maze,n,solution,x+1,y);
        mazeHelp(maze,n,solution,x,y-1);
        mazeHelp(maze,n,solution,x,y+1);
        solution[x][y] = 0;
}
void ratInAMaze(int maze[][20], int n){

  int** solution = new int*[n];
  for(int i=0;i<n;i++){
        solution[i] = new int[n];
  }
  mazeHelp(maze,n,solution,0,0);


}
#include<bits/stdc++.h>
```

```
using namespace std;
int main() {

        // Write your code here
    int N;cin >> N;
    int maze[20][20];

    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < N; j++){
            cin >> maze[i][j];
        }
    }
    ratInAMaze(maze,N);
        return 0;
}

```

## Live Problem -1 : https://www.codechef.com/problems/AX06

## 3-Tut : Crossword Problem

Send Feedback

Coding Ninjas has provided you a crossword of 10*10 grid. The grid contains '+' or '-' as its cell values. Now, you are also provided with a word list that needs to placed accurately in the grid. Cells marked with '-' are to be filled with word list.

For example, The following is an example for the input crossword grid and the word list.

```
+-++++++++
+-++-+++++
+-------++
+-++-+++++
+-++-+++++
+-++-+++++
++++-+++++
++++-+++++
++++++++++
----------
```

CALIFORNIA;NIGERIA;CANADA;TELAVIV

Output for the given input should be:

+C++++++++

+A++T+++++

+NIGERIA++

+A++L+++++

+D++A+++++

+A++V+++++

++++I+++++

++++V+++++

++++++++++

CALIFORNIA

Note: We have provided such test cases that there is only one solution for the given input.

### Input format:

The first 10 lines of input contain crossword. Each of 10 lines has a character array of size 10. Input characters are either '+' or '-'.

The next line of input contains the word list, in which each word is separated by ';'.

### Output format:

Print the crossword grid, after placing the words of word list in '-' cells.

### Constraints

The number of words in the word list lie in the range of: [1,6]

The length of words in the word list lie in the range: [1, 10]

Time Limit: 1 second

### Sample Input 1:

+-++++++++

+-++-+++++

+-------++

+-++-+++++

+-++-+++++

+-++-+++++

++++-+++++

++++-+++++

++++++++++

----------

CALIFORNIA;NIGERIA;CANADA;TELAVIV

### Sample Output 1:

+C++++++++

+A++T+++++

+NIGERIA++

+A++L+++++

+D++A+++++

+A++V+++++

++++I+++++

++++V+++++

++++++++++

CALIFORNIA

```
#include<bits/stdc++.h>
using namespace std;
#include<iostream>
#include<string>
#include<vector>
using namespace std;
#define n 10
void printer(char cross[n][n])
{
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			cout << cross[i][j];
		}
		cout << endl;
	}
	cout << endl;
}
bool isvalid_vertical(char cross[n][n], int r, int c, string word)
{
	int j = r;
	for (int i = 0;i < word.length();i++)
	{
		if (j > 9)
			return false;
		if (cross[j][c] == '+' || (cross[j][c] != '-' && cross[j][c] != word[i]))
			return false;
		if (cross[j][c] == '-' || cross[j][c] == word[i])
			j++;
	}
	return true;
}
bool isvalid_horizontal(char cross[n][n], int r, int c, string word)
{
	int j = c;
	for (int i = 0;i < word.length();i++) {
		if (j > 9)
			return false;
		if (cross[r][j] == '+' || (cross[r][j] != '-' && cross[r][j] != word[i]))
			return false;
		if (cross[r][j] == '-' || cross[r][j] == word[i])
			j++;
	}
	return true;
}
void set_vertical(char cross[n][n], int r, int c, vector<bool>& v, string word)
{
	int row = r;
```

```
50.             for (int i = 0; i < word.length(); i++)
51.             {
52.                     if (cross[row + i][c] == '-')
53.                     {
54.                             cross[row + i][c] = word[i];
55.                             v.push_back(true);
56.                     }
57.                     else
58.                     {
59.                             v.push_back(false);
60.                     }
61.             }
62. }
63. void reset_vertical(char cross[n][n], int r, int c, vector<bool>v)
64. {
65.             int row = r;
66.             for (int i = 0; i < v.size(); i++)
67.             {
68.                     if (v[i])
69.                     {
70.                             cross[row + i][c] = '-';
71.                     }
72.             }
73. }
74. void set_horizontal(char cross[n][n], int r, int c, vector<bool>& v, string word)
75. {
76.             int col = c;
77.             for (int i = 0; i < word.length(); i++)
78.             {
79.                     if (cross[r][col + i] == '-')
80.                     {
81.                             cross[r][col + i] = word[i];
82.                             v.push_back(true);
83.                     }
84.                     else
85.                     {
86.                             v.push_back(false);
87.                     }
88.
89.             }
90. }
91. void reset_horizontal(char cross[n][n], int r, int c, vector<bool>v)
92. {
93.             int col = c;
94.             for (int i = 0; i < v.size(); i++)
95.             {
96.                     if (v[i])
97.                     {
98.                             cross[r][col + i] = '-';
```

```
                }
            }
        }

        bool crossword(char cross[n][n], vector<string>words, int index)
        {
            //base case is if index become == v.size()----> then print the crossword;
            if (index >= words.size())
            {
                printer(cross);
                return true;
            }
            for (int r = 0; r < n; r++)
            {
                for (int c = 0; c < n; c++)
                {
                    if (cross[r][c] == '-' || cross[r][c] == words[index][0])
                    {
                        if (isvalid_vertical(cross, r, c, words[index]))
                        {
                            //maintain a helper array
                            vector<bool>helper;
                            set_vertical(cross, r, c, helper, words[index]);
                            //pass this helper array in the function set_vertical where this helper array is a boolean array.
                            //set vertical(helper, ...., ....)
                            if (crossword(cross, words, index + 1))
                            {
                                return true;
                            }
                            //setvertical call karne k baad crossword call karna hai for next index words.
                            // crossword(cross, words, index+1)
                            // ye jo crossword function hai wo boolean hai. if it returns true, just return.
                            // or if it return false, call the reset vertical function---> reset_vertical(helper, ....)
                            reset_vertical(cross, r, c, helper);
                            // agar is valid vertical false return karta hai to hum isvalid_horizontal mei check kar lenge.
                            // isme bhi vhi hoga helper, set_horizontal, crossword, reset_horizontal.
                        }
                        if (isvalid_horizontal(cross, r, c, words[index]))
                        {
                            vector<bool>helper;
                            set_horizontal(cross, r, c, helper, words[index]);
                            if (crossword(cross, words, index + 1))
                            {
```

```
                                                return true;
                                            }
                                            reset_horizontal(cross, r, c, helper);
                                        }
                                    }
                                }
                            }
                        }
                        return false;
                    }
                    int main()
                    {
                        char arr[n][n];
                        for (int i = 0; i < n; i++)
                        {
                            string s;
                            cin >> s;
                            for (int j = 0; j < n; j++)
                            {
                                arr[i][j] = s[j];
                            }
                        }
                        string s;
                        cin >> s;
                        vector<string>words;
                        for (int i = 0; i < s.length(); i++)
                        {
                            int j = i;
                            while (s[j] != ';' && j < s.length())
                            {
                                j++;
                            }
                            words.push_back(s.substr(i, j - i));
                            i = j;
                            j++;
                        }
                        bool x = crossword(arr, words, 0);
                    }
```

## 4-Ass : **Sudoku Solver**

Send Feedback

Coding Ninjas has provided you a 9*9 sudoku board. The board contains non zero and zero values. Non zero values lie in the range: [1, 9]. Cells with zero value indicate that the particular cell is empty and can be replaced by non zero values.

You need to find out whether the sudoku board can be solved. If the sudoku board can be solved, then print true, otherwise print false.

### Input Format:

There are nine lines in input. Each of the nine lines contain nine space separated integers. These nine lines represent the sudoku board.

### Output Format:

The first and only line of output contains boolean value, either true or false, as described in problem statement.

### Constraints:

The cell values lie in the range: [0, 9]

Time Limit: 1 second

### Note:

Input are given in a way that backtracking solution will work in the provided time limit.

### Sample Input 1:

9 0 0 0 2 0 7 5 0

6 0 0 0 5 0 0 4 0

0 2 0 4 0 0 0 1 0

2 0 8 0 0 0 0 0 0

0 7 0 5 0 9 0 6 0

0 0 0 0 0 0 4 0 1

0 1 0 0 0 5 0 8 0

0 9 0 0 7 0 0 0 4

0 8 2 0 4 0 0 0 6

### Sample Output 1:

true

```
1.  #include<iostream>
2.  using namespace std;
3.
4.  bool find_empty(int board[][9],int &row,int &col){
```

```
    for(int i=0;i<9;i++){
        for(int j=0;j<9;j++){
            if(board[i][j]==0){
                row=i;
                col=j;
                return true;
            }
        }
    }
    return false;
}

bool check_row(int board[][9],int i,int j,int num){
    for(int row=0;row<9;row++){
        if(board[row][j]==num){
            return false;
        }
    }
    return true;
}

bool check_col(int board[][9],int i,int j,int num){
    for(int col=0;col<9;col++){
        if(board[i][col]==num){
            return false;
        }
    }
    return true;
}

bool check_block(int board[][9],int i,int j,int num){
    int checki=i>=0&&i<3?0:i>=3&&i<6?3:6;
    int checkj=j>=0&&j<3?0:j>=3&&j<6?3:6;
    for(int row=checki;row<checki+3;row++){
        for(int col=checkj;col<checkj+3;col++){
            if(board[row][col]==num){
                return false;
            }
        }
    }
    return true;
}

bool solve_sudoku(int board[][9]){
    int row,col;
    bool found=find_empty(board,row,col);
    if(found){
        // go from 1 to 9 and
        for(int i=0;i<=9;i++){
```

```
            bool rowcheck=check_row(board,row,col,i);
            bool colcheck=check_col(board,row,col,i);
            bool blockcheck=check_block(board,row,col,i);
            if(rowcheck&&colcheck&&blockcheck){
                board[row][col]=i;
                bool findifvalid=solve_sudoku(board);
                if(findifvalid){
                    return true;
                }
                board[row][col]=0;
            }
        }
        // check row
        // check col
        // check block
        // fill that row,col and call solve_sudoku again
        // if returned true - return true
        // if returned false - try for another num
    }else{
        // no empty means sudoku is filled or solved
        return true;
    }
    return false;
}

bool sudokuSolver(int board[][9]){
    return solve_sudoku(board);

}

int main(){

   int n = 9;
   int board[9][9];
   for(int i = 0; i < n ;i++){
            for(int j = 0; j < n; j++){
            cin >> board[i][j];
        }
   }
   if(sudokuSolver(board)){
            cout << "true";
   }else{
            cout << "false";
   }
}

```

# 5-Ass : Subset Sum

Send Feedback

You are given an array of integers and a target K. You have to find the count of subsets of the given array that sum up to K.

## Note:

1. Subset can have duplicate values.

2. Empty subset is a valid subset and has sum equal to zero.

## Input Format:

The first line of input will contain an integer T, that denotes the value of number of test cases.

Each test case contains two lines. The first line of each test case will contain two space-separated integers N and K, where N is the size of the given array and K is the target value.

The second line of each test case will contain N space separated integers denoting the elements of the input array.

## Output Format ;

For each test case, print the number of subsets that sum upto K in a separate line.

## Constraints

1 <= T <= 50

2 <= N <= 15

0 <= array[i] <= 10^9

0 <= K <= 10^9

Time Limit: 1 second

## Sample Input 1:

1

5 6

2 4 4 3 1

### Sample Output 1:

3

### Explanation:

Following are the three subsets, that sum upto K=6:

1. [2, 4], Element 4 in this subset is the one at index 1

2. [2, 4], Element 4 in this subset is the one at index 2

3. [2, 3, 1]

### Sample Input 2:

2

8 8

1 4 1 3 1 1 2 3

8 2

4 1 4 4 2 4 1 1

### Sample Output 2:

30

4

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;

void countSubsetSum(ll* a, ll n, int index, ll currSum, ll target, ll &count){
    if(index>n) return ;
    if(index==n){
        if(currSum==target) count++;
        return;
    }
    countSubsetSum(a,n,index+1,currSum+a[index],target,count);
    countSubsetSum(a,n,index+1,currSum,target,count);
}
int main(){

    int t;
```

```
    cin>>t;
    while(t--){
        ll n,k;
        cin>>n>>k;
        ll* a = new ll[n];
        for(int i = 0; i<n; i++) cin>>a[i];
        ll ans = 0;
        countSubsetSum(a,n,0,0,k,ans);
        cout << ans << endl;

    }
}
```

# L9 : Modulo Arithmetic Practice Questions

## 1-Tut : Number Of Balanced BTs

Send Feedback

You are given an integer h, find and print the count of all possible balanced binary trees of height h.

This number can be huge, so return the output modulus 10^9 + 7.

### Input Format :

The first line of input contains an integer that denotes the value of the number of test cases. Let us denote it with the symbol T.

Each test case consists of a single integer h, that denotes height of the binary tree, in a separate line.

### Output Format :

For each test case, there is a single line of output, which prints the count of all possible balanced binary trees of height h, modulus 10^9 + 7. The output for each test case is printed in a separate line.

### Constraints :

1 <= T <= 10

1 <= h <= 20

Time Limit: 1 second

### Sample Input 1:

1

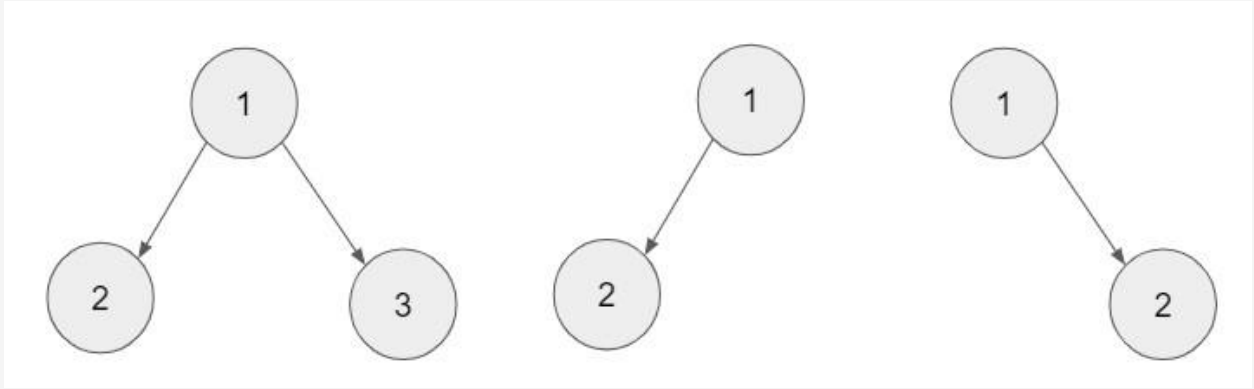
2

### Sample Output 1:

3

### Explanation:

Following are the three balanced binary trees that can be formed with height = 2.




### Sample Input 2:

2

3

4

### Sample Output 2:

15

315

```
#include<bits/stdc++.h>
using namespace std;
#define mymodulo 1000000007
long long int noofBBT(int h){
    if(h == 0 || h == 1){
        return 1;
    }
    long long int x = noofBBT(h-1);
    long long int y = noofBBT(h-2);
    x = x % mymodulo;
    y = y % mymodulo;

    return ( (x*x) % mymodulo + (2*x*y) % mymodulo ) % mymodulo;
}


int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int h; cin >> h;
        long long int output = noofBBT(h);
        cout << output << endl;

    }
    return 0;
}
```

## 2-Ass : Product of first N natural numbers

Send Feedback

You are given an integer N and you have to calculate the product of the first N natural numbers.

As the answer will be large, print the answer modulo 10^9+7.

### Input Format :

The first line of input contains an integer that denotes the value of the number of test cases. Let us denote it with the symbol T.

Each test case consists of a single integer N in a separate line.

### Output Format :

For each test case, print the product of the first N natural numbers modulus 10^9 + 7 in a separate line.

### Constraints :

1 <= T <= 100

1 <= N <= 10^5

Time Limit: 1 second

**Sample Input 1:**

2
3
4

**Sample output 1:**

6

24

```
#include<bits/stdc++.h>
using namespace std;
#define modu 1000000007
int prod_1_to_n(int N){
    int i = 1;
    long long int prod = 1;
    while(i <= N) {
        prod = prod % modu;
        prod = (prod*i) % modu;
        i++;
    }
    return prod;
}

int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int N; cin >> N;
        long long int output = prod_1_to_n(N);
        cout << output << endl;

    }

    return 0;
}
```

## 3-Ass : Power Function

Send Feedback

You are given two integers, X and Y. You have to compute X^Y.

Example:

2^3 evaluates to 8.

Since the result can be large, so calculate the X^Y % M, where M is 10^9 + 7.

Note: Apply the brute force method to solve the problem.

## Input Format:

The first line of input contains an integer that denotes the value of the number of test cases. Let us denote it with the symbol T.

Each test case consists of two space separated integers that denote the value of X and Y. Each test case is given in a separate line.

## Output Format:

For each test case, print the answer modulo 10^9 + 7 in a separate line.

## Constraints:

1 <= T <= 50

1 <= X, Y <= 10^5

Time Limit: 1 second

## Sample Input 1:

2

2 3

3 2

Sample Output 1:

8

9

```
#include<bits/stdc++.h>
using namespace std;
#define modulo 1000000007
long long int power(int x, int y){
    int i = 1;
    long long int prod = 1;
    while(i <= y){
        prod = prod % modulo;
        prod = (prod * x ) % modulo;
        i++;
    }
    return prod;
}


int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int x,y;
        cin >> x >> y;
        long long int result = power(x,y);
        cout << result << endl;
    }
    return 0;
}
```

# L10: Ad Hoc Problem Practice Questions

a-Live: https://codeforces.com/problemset/problem/1037/C

## 1-Tut : Rectangular Area

Send Feedback

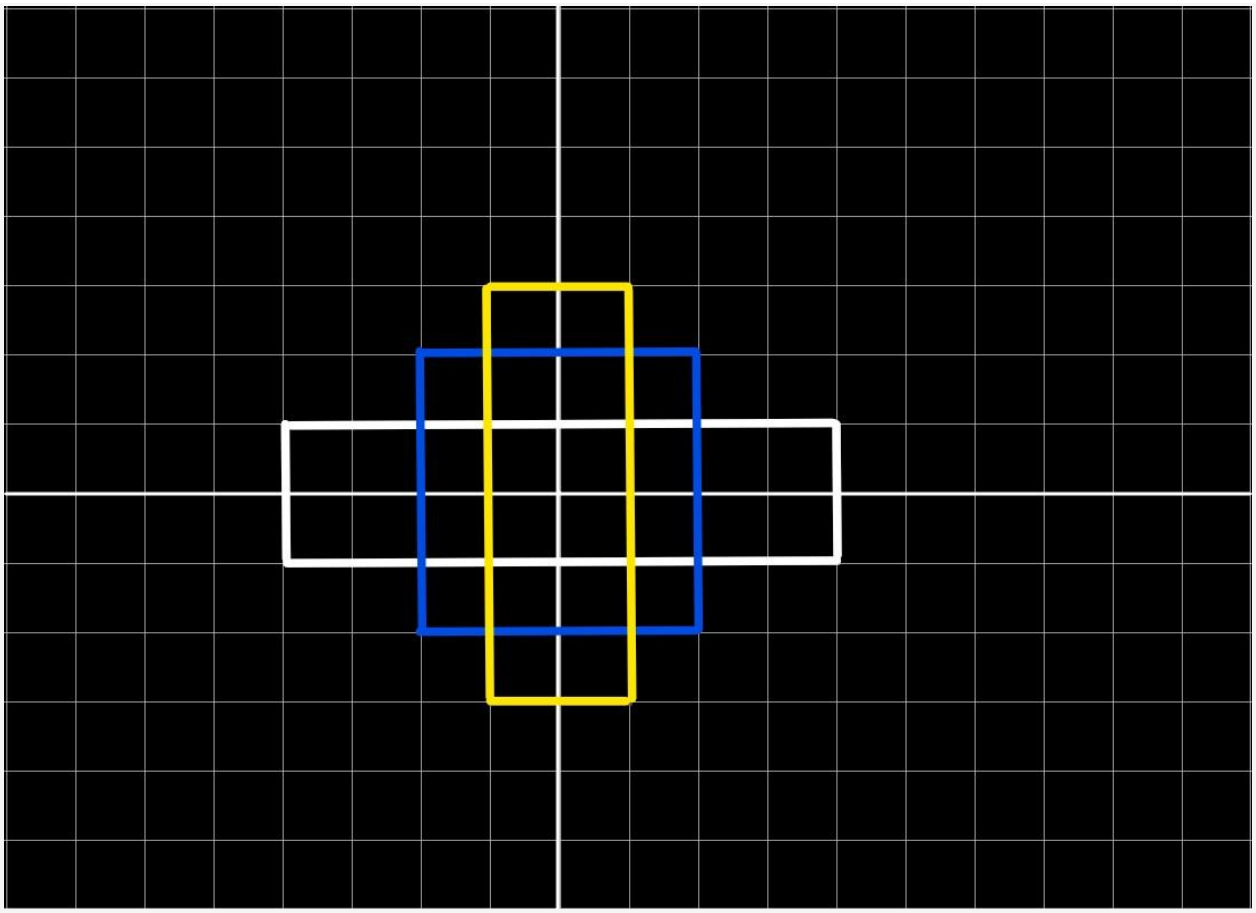
You are given N rectangles, which are centered in the center of the Cartesian coordinate system and their sides are parallel to the coordinate axes. Each rectangle is uniquely identified with its width (along the x-axis) and height (along the y-axis). Navdeep has coloured each rectangle in a certain colour and now wants to know the area of the coloured part of the paper. Please refer to the sample test case 1 and image used in it for better understanding.

### Sample Input 1 :

3
8 2 (represented by white coloured rectangle)
4 4 (represented by blue coloured rectangle)
2 6 (represented by yellow coloured rectangle)

### Sample Output 1 :

28



In other words, he wants to know the number of unit squares that belong to at least one rectangle. This can also be seen in the image. There are 28 unit squares, which come in the bounds of at least one rectangle.

### Input Format :

The first line of input contains the integer N, that denotes the number of rectangles.
Each of the following N lines contains even integers X and Y, dimensions (width and height, respectively) of the corresponding rectangles.

### Output Format :

The first and only line of output must contain the required area, as described in the task.

### Constraints:

1 <= N <= 10^5
1 <= X,Y <= 10^5
All values of X and Y are even
Time Limit: 1 second

### Sample Input 1 :

3
8 2
4 4
2 6

### Sample Output 1 :

28

### Sample Input 2 :

5
2 10
4 4
2 2
8 8
6 6

### Sample Output 2 :

68

```
#include<bits/stdc++.h>
using namespace std;
int main(){

    // write your code here
    int n; cin >> n;
    int *height = new int[1000000]();
    int max_x = 0;
    int x,y;
    for(int i = 0; i < n; i++){
        cin >> x >> y;
        if(height[x/2] < y){
            height[x/2] = y;
        }
```

```
        if(x/2 > max_x){
            max_x = x/2;
        }

    }
    long area = 0;
    for(int i = max_x; i > 0; i--){
        if(height[i] < height[i+1]){
            height[i] = height[i+1];
        }
        area += height[i];
    }

    cout << 2*area << endl;
    return 0;
}
```

## 2-Ass : Light Up the Bulbs

Send Feedback

Navdeep is given a binary string (string consists of only 0's and 1's) of size 'n'. The string represents the state of n bulbs. '0' represents that the bulb is in 'OFF' mode and '1' represents the bulb is in 'ON' mode. Navdeep is given the task to change all the characters to 1. In other words, Navdeep has to light up all the bulbs.

He can perform two operations on the given string:

1. In the first operation, he can choose any substring and reverse it. For example, in the binary string: 0110001, if we reverse substring from 1st to 3rd index in the given string, then the resultant string will be: 0011001. Here, 110 is reversed to form 011. This operation will cost Rs. 'X'.
2. In the second operation, he can choose any substring and toggle each character of that substring. For example, in the binary string: 0110001, if we toggle characters of the substring from 1st to 3rd index in the given string, then the resultant string will be: 0001001. Here, 110 is converted to form 001. This operation will cost Rs. 'Y'.

Can you help Navdeep to complete the task in the minimum amount possible.

### Input Format:

The first line will contain three space separated integers: 'n', 'X', 'Y', denoting the number of bulbs, cost of first operation and cost of second operation respectively.

The second line contains a binary string of length 'n', consisting of 0's and 1's, representing whether the bulb is 'OFF' or 'ON'.

### Output Format:

Print a single integer denoting the minimum cost required to light up all the bulbs.

### Constraints:

1 <= n <= 3,00,000

0 <= X, Y <= 10^9

Time Limit: 1 second

### Sample Input:

5 1 10
01000

**Sample Output:**

11

**Explanation:**

First, Reverse substring (0, 1): “01000” -> “10000”, COST = 1
Second, Invert substring (1, 4): “10000” -> “11111”, COST = 10

Total cost = 1+10 => 11

```
#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
int main()
{
        long long n, x, y;
        cin >> n >> x >> y;
        string s;
        cin >> s;
        long long count = 0;
        if(s[0]=='0')
        {
                count = 1;
        }
        for(long long i=1; i<n; i++)
        {
                if((s[i-1]=='1') && (s[i]=='0'))
                {
                        count += 1;
                }
        }

        if(count==0)
        {
                cout << 0;
        }
        else
        {
                cout << (count-1) * min(x, y) + y;
        }
}
```

# 3-Ass : Interesting Sequences

Send Feedback

Professor Jain has a class full of notorious students. To get anything done from them is a herculean task. Prof Jain wanted to organize a test. He gave this responsibility to Aahad. Aahad did an excellent job of organizing the test. As a reward, the professor gave him an sequence of numbers to play with. But Aahad likes playing with "interesting" sequence of numbers, which are sequences that have equal elements.

Now, the problem is - Prof Jain has a sequence of numbers, but that sequence isn't always "interesting".

To ensure sequence has equal elements, Prof Jain has 2 options:

1) Choose two elements of sequence. Decrease the first element by 1 and increase the second element by 1. This operation costs 'k' coins.

2) Choose one element of array and increase it by 1. This operation costs 'l' coins.

You have to find the minimum number of coins that Prof Jain needs to turn his sequence into a “interesting" sequence for Aahad?

## Input Format

The first line of input contains three space-separated integers: n, k, l. Integer n is the size of array. Integer k is the number of coins needed to perform the first operation. Integer l is the number of coins needed to perform the second operation.

The second line of input contains n space separated integers: (a1, a2, a3... an), that denote the numbers of the sequence.

## Constraints:

1 <= n <= 1000

1 <= k, l <= 10^9

1 <= a[i] <= 1000

Time Limit: 1 second

## Output Format

The first and only line of output prints an integer, that denotes minimum number of coins required to make "interesting" sequence.

## Sample Input 1:

4 1 2

3 4 2 2

## Sample Output 1:

3

## Explanation Output 1 :

The professor has a sequence with 4 elements. To perform the first operation, he must pay 1 coin and to perform the second operation, he must pay 2 coins. The optimal strategy is:

-Perform the second operation on the fourth element. Now the sequence is {3, 4, 2, 3}. This costs 2 coins.

-Perform the first operation on the second and third element. The sequence is now "interesting", and it looks like {3, 3, 3, 3}. This costs 1 coin.

The total amount of coins needed is 2 + 1 = 3.

## Sample Input 2:

3 2 1

5 5 5

## Sample Output 2:

0

### Explanation Output 2 :

The given sequence is already "interesting". The professor would spend 0 coins.

### Sample Input 3:

5 2 1
1 2 3 4 5

### Sample Output 3:

6

### Explanation Output 3 :

The professor has a sequence with 5 elements. To perform the first operation, he must pay 2 coins and to perform the second operation, he must pay 1 coin. The optimal strategy is:
-Perform the first operation on the first and last element. Now the sequence is {2, 2, 3, 4, 4}. This costs 2 coins.
-Perform the first operation again on the first and last element. Now the sequence is {3, 2, 3, 4, 3}. This costs 2 coins.
-Perform the first operation on the second and second last element. Now the sequence is {3, 3, 3, 3, 3}. This costs 2 coins.

The total amount of coins needed is 2 + 2 + 2 = 6.

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

int minCost(const vector<int>& a, const int& k, const int& l, int curr) {
	int inc = 0, dec = 0;
	for(int x : a) {
		if(curr > x) {
			inc += (curr - x);
		}
		else {
			dec += (x - curr);
		}
	}
	if(inc >= dec) {
		return (dec * k) + ((inc - dec) * l);
	}
	return (int) 1e15;
}

int32_t main()
{
	ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

	int n; cin >> n;
	int k, l; cin >> k >> l;

```

```
29.         vector<int> a(n);
30.         for(int i = 0 ; i < n ; ++i) {
31.                 cin >> a[i];
32.         }
33.
34.         int rangeMax = *max_element(a.begin(), a.end());
35.         int rangeMin = *min_element(a.begin(), a.end());
36.
37.         int ans = (int) 1e15;
38.         for(int i = rangeMin ; i <= rangeMax ; ++i) {
39.                 ans = min(ans, minCost(a, k, l, i));
40.         }
41.
42.         cout << ans << '\n';
43.         return 0;
44. }
```

## 4-Ass : Winning Strategy

Send Feedback

The college team, along with their coach, is going to the sports fest to play a football match. There are n players in the team, numbered from 1 to n.

Someone gives a paper to the coach. The paper elaborates on the positions and strategies of the opponent team. Based on it, the coach creates a winning strategy. In that strategy, he decides and gives a particular position to every player.

After this, the coach starts swapping two players at a time to make them stand according to new positions decided on paper.

He swaps players by applying following rules:

1. Any player can swap with the player standing next to him.
2. One player can swap with at most two other players.

Given that initially all the players are standing linearly, numbered from 1 to n, you have to tell whether it is possible for the coach to create new positions by swapping within the constraints defined in the task.

### Input Format

The first line of input will contain an integer, that denotes the value of the number of test cases. Let us denote the number of test cases by the symbol T.

Each of the following T test cases consists of two lines. The first line of each test case contains an integer n, that denotes the number of players in the team. The following line contains n space separated integers, denoting the specific position of players in winning strategy.

### Output Format

For each test case, if it is possible to create winning strategy positions, then print "YES" (without quotes) and in the next line, print the minimum numbers of swaps required to form the winning strategy order, otherwise print "NO"(without quotes) in a new line.

### Constraints

1 <= T <= 50

1 =< N <= 10^5
1 <= A[i] <= n
Time Limit: 1 second

### Sample Input 1:

1
5
2 1 5 3 4

### Sample Output 1:

YES
3

### Explanation

In this case, we can achieve winning strategy positions in 3 swaps. Initial state of positions: 1 2 3 4 5
Three moves required to form winning strategy positions:
1 2 3 4 5 -> 1 2 3 5 4 -> 1 2 5 3 4 -> 2 1 5 3 4

### Sample Input 2:

1
5
2 5 1 3 4

### Sample Output 2:

NO

### Explanation:

In the second case, there is no way to form the specific winning strategy positions by swapping within the constraints mentioned in the task.

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

void solve(vector<int> a, int n) {
        for(int i = 0 ; i < n ; ++i) {
                if(abs(a[i]-i-1) > 2) {
                        cout << "NO\n";
                        return;
                }
        }

        int count = 0;
        for(int i = n-1 ; i >= 0 ; --i) {
                if(a[i] == i+1) {
                        continue;
                }

                if(i-1 >= 0 && a[i-1] == i+1) {
                        ++count;
                        swap(a[i], a[i-1]);
                }
```

```
                else if(i-2 >= 0 && a[i-2] == i+1) {
                        count += 2;
                        a[i-2] = a[i-1];
                        a[i-1] = a[i];
                        a[i] = i+1;
                }
                else {
                        cout << "NO\n";
                        return;
                }
        }

        cout << "YES\n" << count << '\n';
}

int32_t main()
{
        ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

        int t; cin >> t;
        while(t--)
        {
                int n; cin >> n;
                vector<int> a(n);
                for(int i = 0 ; i < n ; ++i) {
                        cin >> a[i];
                }

                solve(a, n);
        }
        return 0;
}
```

# L11 : Dynamic Programming 1 Practice Questions

## 1-Tut : AlphaCode-Question

Send Feedback

Alice and Bob need to send secret messages to each other and are discussing ways to encode their messages:

Alice: “Let’s just use a very simple code: We’ll assign ‘A’ the code word 1, ‘B’ will be 2, and so on down to ‘Z’ being assigned 26.”
Bob: “That’s a stupid code, Alice. Suppose I send you the word ‘BEAN’ encoded as 25114. You could decode that in many different ways!”
Alice: “Sure you could, but what words would you get? Other than ‘BEAN’, you’d get ‘BEAAD’, ‘YAAD’, ‘YAN’, ‘YKD’ and ‘BEKD’. I think you would be able to figure out the correct decoding. And why would you send me the word ‘BEAN’ anyway?”
Bob: “OK, maybe that’s a bad example, but I bet you that if you got a string of length 5000 there would be tons of different decodings and with that many you would find at least two different ones that would make sense.”
Alice: “How many different decodings?”
Bob: “Jillions!”
For some reason, Alice is still unconvinced by Bob’s argument, so she requires a program that will determine how many decodings there can be for a given string using her code.

### Input Format:

First line will contain T (number of test case).
Each input is consists of a single line containing the message string

### Output Format:

For each test case print the answer % mod (10^9 +7)

### Constraints:

1 <= T <= 100
1 <= |S| <= 10^5
sum of length of all string doesn't exceed 5*10^6

### Sample Input 1:

```
3
47974
6349988978
1001
```

### Sample Output 1:

```
1
1

0
```

```cpp
#include <bits/stdc++.h>
using namespace std;
#define mod 1000000007

int alpha_code(int * input, int size) {
    vector<int> output(size + 1,0);
    output[0] = 1;
    output[1] = 1;

    for (int i = 2; i <= size; i++) {
        if (input[i-1] != 0)
        {
            output[i] = output[i - 1] % mod;
        }

        if (input[i-2] *10 + input[i - 1] <= 26)
        {
            if (input[i-2] != 0)
            {
                output[i] = (output[i - 2] + output[i])%mod;
            }
        }
    }
    int ans = output[size];
    return ans;
}

int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        string input; cin >> input;
        if(input[0] == '0') return 0;
        int len = input.length();
        int arr[len];
        for(int i=0; i<len; i++)
            arr[i] = input[i] - 48;
        cout << alpha_code(arr,len) << endl;
    }
    return 0;
}
```

## 2-Tut : **Largest Bitonic Subsequence**

Send Feedback

You are given an array of positive integers as input. Write a code to return the length of the largest such subsequence in which the values are arranged first in strictly ascending order and then in strictly descending order.

Such a subsequence is known as bitonic subsequence. A purely increasing or purely decreasing subsequence will also be considered as a bitonic sequence with the other part empty.

**Note that the elements in bitonic subsequence need not be consecutive in the given array but the order should remain same.**

**Input Format:**

First line will contain T (number of test case), each test is consists of two lines.
Line 1 : A positive Integer N, i.e., the size of array
Line 2 : N space-separated integers as elements of the array

**Output Format:**

Length of Largest Bitonic subsequence for each test case in a newline.

**Input Constraints:**

1 <= T <= 10
1<= N <= 5000

**Sample Input 1:**

```
1
6
15 20 20 6 4 2
```

**Sample Output 1:**

```
5
```

**Sample Output 1 Explanation:**

Here, longest Bitonic subsequence is {15, 20, 6, 4, 2} which has length = 5.

**Sample Input 2:**

```
1
2
1 5
```

**Sample Output 2:**

```
2
```

**Sample Input 3:**

```
1
2
5 1
```

**Sample Output 3:**

```
2
```

```
#include<iostream>
using namespace std;
#include<algorithm>

int* LISfront(int* arr,int n){
    int* output = new int[n];
    output[0] = 1;
    for(int i=1;i<n;i++){
        output[i] = 1;
        for(int j=i-1;j>=0;j--){
            if(arr[j]>=arr[i]){
```

```
            continue;
        }
        int currAns = output[j] + 1;
        if(currAns > output[i]){
            output[i] = currAns;
        }

    }
  }
  return output;
}

int* LISback(int* arr,int n){
  int* output = new int[n];
  output[n-1] = 1;
  for(int i=n-2;i>=0;i--){
    output[i] = 1;
    for(int j=i+1;j<n;j++){
      if(arr[j] >= arr[i]){
        continue;
      }
      int currAns = output[j]+1;
      if(currAns > output[i]){
        output[i] = currAns;
      }
    }

  }
  return output;
}


int longestBitonicSubarray(int *input, int n) {

  int* LISbegin = LISfront(input,n);
  int* LISend = LISback(input,n);
  int maximum = 1;
  for(int i=0;i<n;i++){
    maximum = max(maximum,LISbegin[i]+LISend[i]-1);
  }
  return maximum;
}

int main()
{
  int T; cin >>T;
  while(T--)
  {
    int n, input[100000];
```

```
        cin>>n;
        for(int i=0; i<n; i++)
        {
           cin>>input[i];
        }
      cout << longestBitonicSubarray(input, n) << endl;
    }
  return 0;
}
```

## 3-Tut : StairCase Problem

Send Feedback

A child is running up a staircase with n steps and can hop either 1 step, 2 steps or 3 steps at a time. Implement a method to count how many possible ways the child can run up to the stairs. You need to return all possible number of ways.

Time complexity of your code should be O(n).

Since the answer can be pretty large print the answer % mod(10^9 +7)

### Input Format:

First line will contain T (number of test case).
Each test case is consists of a single line containing an integer N.

### Output Format:

For each test case print the answer in new line

### Constraints :

1 <= T < = 10
1 <= N <= 10^5
Sample input 1

2
2
3
Sample output 1

2
4
Explanation of sample input 1:

Test case 1:
To reach at top of 2nd stair, we have only two options i.e (2), (1,1)

Test case 2:
To reach at top of 3rd stair, we have four options i.e (1,1,1), (1,2) ,(2,1), (3)
Sample input 2:

2
5

10
Sample output 2:

13

274

```
#include<bits/stdc++.h>
using namespace std;
#include <iostream>
long long mod =1e9+7;

long long int staircase(int n){
    if(n<=1)
        return 1;

    long long a = 1, b = 1, c = 2; for (int i = 0; i <= n - 3; ++i) { long long d = (a + b + c) % mod; a = b; b = c; c = d; } return c; }


int main()
{
    int T;
    cin >> T;
    while(T--)
    {
        int N; cin >> N;
        cout << staircase(N) << endl;

    }
    return 0;
}
/*aur mod me 1e9+7 liya


yeh liya karo jab aisa hu kare
*/
```

## 4-Tut : Coin Change Problem

Send Feedback

You are given an infinite supply of coins of each of denominations D = {D0, D1, D2, D3, ..... Dn-1}. You need to figure out the total number of ways W, in which you can make a change for Value V using coins of denominations D.

Note : Return 0, if change isn't possible.

W can be pretty large so output the answer % mod(10^9 + 7)

## Input Format

First line will contain T (number of test case), each test case is consists of 3 three lines.
Line 1 : Integer n i.e. total number of denominations
Line 2 : N integers i.e. n denomination values
Line 3 : Value V

## Output Format

For each test case print the number of ways (W) % mod(10^9 +7) in new line.
Constraints :

1 <= T <= 10
1 <= N <= 10
1 <= V <= 5000

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
long long mod =1e9+7;
using namespace std;
int getCount(int* arr,int n,int value,vector<vector<int>>&temp){
    if(value == 0){
        return 1;
    }
    if(n == 0){
        return 0;
    }
    if(value < 0){
        return 0;
    }
    if(temp[n][value] > -1){
        return temp[n][value];
    }
    int first = getCount(arr,n,value-arr[n-1],temp) % mod;
    int second = getCount(arr,n-1,value,temp) % mod;
    return temp[n][value] = (first+second) % mod;
}

int countWaysToMakeChange(int denominations[], int n, int value){

    vector<vector<int>>temp(n+1,vector<int>(value+1,-1));
            return getCount(denominations,n,value,temp);

}


signed main(){

    // write your code here
    int T; cin >> T;
```

```
    while(T--){
            int numDenominations;
            cin >> numDenominations;
            int* denominations = new int[numDenominations];
        for(int i = 0; i < numDenominations; i++)
        {
          cin >> denominations[i];
        }
        int value;
        cin >> value;
        cout << countWaysToMakeChange(denominations, numDenominations, value) << endl;
    }
    return 0;
}
```

## 5-Tut : Magic Grid Problem

Send Feedback

You are given a magrid S ( a magic grid ) having R rows and C columns. Each cell in this magrid has either a Hungarian horntail dragon that our intrepid hero has to defeat, or a flask of magic potion that his teacher Snape has left for him. A dragon at a cell (i,j) takes away |S[i][j]| strength points from him, and a potion at a cell (i,j) increases Harry's strength by S[i][j]. If his strength drops to 0 or less at any point during his journey, Harry dies, and no magical stone can revive him.

Harry starts from the top-left corner cell (1,1) and the Sorcerer's Stone is in the bottom-right corner cell (R,C). From a cell (i,j), Harry can only move either one cell down or right i.e., to cell (i+1,j) or cell (i,j+1) and he can not move outside the magrid. Harry has used magic before starting his journey to determine which cell contains what, but lacks the basic simple mathematical skill to determine what minimum strength he needs to start with to collect the Sorcerer's Stone. Please help him once again.

Input Format :

The first line contains the number of test cases T. T cases follow. Each test case consists of R C in the first line followed by the description of the grid in R lines, each containing C integers. Rows are numbered 1 to R from top to bottom and columns are numbered 1 to C from left to right. Cells with S[i][j] < 0 contain dragons, others contain magic potions.

Output Format :

Output T lines, one for each case containing the minimum strength Harry should start with from the cell (1,1) to have a positive strength through out his journey to the cell (R,C).

Constraints:

$1 \leq T \leq 5$

$2 \leq R, C \leq 500$

$-10^3 \leq S[i][j] \leq 10^3$

S[1][1] = S[R][C] = 0

Sample Input

```
3
2 3
0 1 -3
1 -2 0
2 2
0 1
2 0
3 4
0 -2 -3 1
-1 4 0 -2
1 -2 -3 0
```

Sample Output

```
2
1

2
```

```
#include<bits/stdc++.h>
using namespace std;

int getStrength(int** arr,int r,int c){
    int** temp = new int*[r];
    for(int i=0;i<r;i++){
        temp[i] = new int[c]();
    }
    temp[r-1][c-2] = 1;
    temp[r-2][c-1] = 1;
    temp[r-1][c-1] = 1;

    //setting last row
    for(int i=c-3;i>=0;i--){
        if(arr[r-1][i+1] < 0){
            temp[r-1][i] = temp[r-1][i+1] - arr[r-1][i+1];
        }else{
            temp[r-1][i] = max(1,temp[r-1][i+1] - arr[r-1][i+1]);
        }
    }

    for(int i=r-3;i>=0;i--){
        if(arr[i+1][c-1] < 0){
            temp[i][c-1] = temp[i+1][c-1] - arr[i+1][c-1];
        }else{
            temp[i][c-1] = max(1,temp[i+1][c-1] - arr[i+1][c-1]);
```

```
        }
    }

    for(int i=r-2;i>=0;i--){
        for(int j=c-2;j>=0;j--){
            int right,down;
            if(arr[i][j+1] < 0){
                right = temp[i][j+1] - arr[i][j+1];
            }else{
                right = max(1,temp[i][j+1] - arr[i][j+1]);
            }

            if(arr[i+1][j] < 0){
                down = temp[i+1][j] - arr[i+1][j];
            }else{
                down = max(1,temp[i+1][j]  - arr[i+1][j]);
            }

            temp[i][j] = min(right,down);
        }
    }

    int ans = temp[0][0];
    return ans;

}


int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int R,C; cin >> R >> C;
        int **arr = new int *[R];
        for(int i = 0; i < R; i++){
            arr[i] = new int[C];
        }
        for(int i=0;i<R;i++){
            for(int j=0;j<C;j++){
                cin >> arr[i][j];
            }
        }
        int ans = getStrength(arr,R,C);
        cout << ans << endl;
    }
    return 0;
}
```

Live Question 1 : https://www.hackerrank.com/challenges/construct-the-array/problem

Live Question 2 : https://www.hackerrank.com/challenges/sam-and-substrings/problem

## 6-Ass : Loot Houses

Send Feedback

A thief wants to loot houses. He knows the amount of money in each house. He cannot loot two consecutive houses. Find the maximum amount of money he can loot.

### Input Format :

The first line of input contains a single integer N denoting the total number of houses.
The second line of input contains N single space-separated integers, denoting the amount of money in every i-th house.

### Output Format :

The only line of output will print the maximum amount of loot that is possible.

### Input Constraints

0 <= N <= 10^5
0 <= A[i] <= 10^4
Where N is the total number of houses.
A[i] represents the money present in the i-th house.
Time limit: 1sec

### Sample Input 1:

6
5 5 10 100 10 5

### Sample Output 1 :

110

### Sample Input 2:

4
10 2 3 11

### Sample Output 2 :

21

### Explanation to Sample Input 2:

Since the thief cant loot two consecutive houses, the ways in which he may loot are:

1. [10, 3]: a total loot of 13
2. [10, 11]: a total loot of 21
3. [2, 11]: a total loot of 13
4. [10]: a total loot of 10
5. [2]: a total loot of 2
6. [3]: a total loot of 3
7. [11]: a total loot of 11

We can't neglect the option to loot just either of the houses if it yields the maximum loot.

From all the possible seven ways, the second option yields the maximum loot amount and hence the answer.

```
#include<bits/stdc++.h>
using namespace std;
int loottotal(int *arr,int n, int si, int ei){
    // Recursive without DP correct solution but giving TLE
   if(n == 0){
      return 0;
   }

    if(si == ei){
      return arr[si];
    }
    if(si == ei-1){
      return max(arr[si], arr[ei]);
    }

    int loot1 = arr[si] + loottotal(arr,n,si+2,ei);
    int loot2 = loottotal(arr,n,si+1,ei);

    return max(loot1,loot2);

}
int main(){

    // write your code here
    int N; cin >> N;
    int *arr = new int[N];
    for(int i = 0 ; i < N;i++){
      cin >> arr[i];
    }

    cout << loottotal(arr,N,0,N-1) << endl;

    return 0;
}
```

```
int loottotal(int *arr,int n, int si, int ei){
    // iterative dp
    if(n == 0){
      return 0;
    }
    if(n == 1){
      return arr[0];
    }

    int *dp = new int[n];
```

```
    dp[0] = arr[0];
    dp[1] = max(arr[0],arr[1]);

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

        int loot1 = arr[i] + dp[i-2];
        int loot2 = dp[i-1];

        dp[i] = max(loot1,loot2);

    }
    return dp[n-1];
}

```

**Solution :**

1. We can do it without declaring the dp array (space complexity reduced by dp array size) bu using 3 variables

```
int loottotal(int *arr,int n, int si, int ei){
    // iterative
    if(n == 0){
        return 0;
    }
    if(n == 1){
        return arr[0];
    }

    if(n == 2){
        return max(arr[0],arr[1]);
    }

    int curr;
    int prev1 = max(arr[0],arr[1]);
    int prev2 = arr[0];

    for(int i = 2; i < n; i++){
            curr = max(prev1,arr[i] + prev2);
            prev2 = prev1;
        prev1 = curr;

    }
    return curr;
}

```

## 7-Ass : Boredom

Send Feedback

Gary is bored and wants to play an interesting but tough game . So he figured out a new board game called "destroy the neighbours" . In this game there are N integers on a board. In one move, he can pick any integer x from the board and then all the integers with value x+1 or x-1 gets destroyed .This move will give him x points.

He plays the game until the board becomes empty . But as he want show this game to his friend Steven, he wants to learn techniques to maximise the points to show off . Can you help Gary in finding out the maximum points he receive grab from the game ?

### Input Format :

First line will contain T (number of test case), each test case is consists of two line.
Line 1: Integer N
Line 2: A list of N integers

### Output Format :

For each test case print maximum points, Gary can receive from the Game setup in a newline.

### Constraints :

1 <= T <= 50
1 <= N <= 10^5
1 <= A[i] <= 1000

### Sample Input :

1
2
1 2

### Sample Output :

2

### Explanation:

Gary can receive a maximum of 2 points, by picking the integer 2.

```
#include<bits/stdc++.h>
using namespace std;
int destroyneighbour(int *arr, int N){

    unordered_map<int,int> m;

    for(int i=0;i<N;i++){
        m[arr[i]] += 1;
    }
    int* output = new int[1005]();
    output[1] = 1*m[1];

    for(int i=2;i<=1000;i++){
        output[i] = max(output[i-1],((i)*(m[i]))+output[i-2]);
    }
    int ans = output[1000];
```

```
    return ans;
}

int main(){

    // write your code here
    int T; cin >> T;
    while(T--){

        int N; cin >> N;
        int *arr = new int[N];
        for(int i = 0; i < N;i++){
            cin>>arr[i];
        }

        cout << destroyneighbour(arr,N) <<endl;
    }
    return 0;
}
```

**Solution :**

```
int destroyneighbour(int *arr, int n){

                        int freq[1001];
                        int dp[1001];
                        for (int i = 0; i <= 1000; i++)
                                {
            freq[i] = 0;
            dp[i] = 0;
            }
        for (int i = 0; i < n; i++)
         {
            freq[arr[i]]++;
         }
            dp[1] = freq[1];
        for (int i = 2; i <= 1000; i++)
        {
            dp[i] = max(dp[i - 1], dp[i - 2] + i * freq[i]);

        }
    return dp[1000];
}


```

# 8-Ass : Minimum Number of Chocolates

Send Feedback

Noor is a teacher. She wants to give some chocolates to the students in her class. All the students sit in a line and each of them has a score according to performance. Noor wants to give at least 1 chocolate to each student. She distributes chocolates to them such that If two students sit next to each other then the one with the higher score must get more chocolates. Noor wants to save money, so she wants to minimise the total number of chocolates.

Note that when two students have equal score they are allowed to have different number of chocolates.

## Input Format:

First line will contain T(number of test case), each test case consists of two lines.
First Line: Integer N, the number of students in Noor's class.
Second Line: Each of the student's score separated by spaces.

## Output Format:

Output the minimum number of chocolates Noor must give for each test case in a newline.

## Input Constraints

1 <= T <= 50
1 <= N <= 50000
1 <= score <= 10^9

## Sample Input:

1
4
1 4 4 6

## sample Output:

6

## Explanation:

The number of chocolates distributed could be:
1 2 1 2

## Sample Input:

1
3
8 7 5

## sample Output:

6

```
#include<bits/stdc++.h>
using namespace std;

int getMin(int *arr, int n){

    int* output = new int[1000005]();
    output[0] = 1;
    int count = 1;
    //left to right assigning with min possible value
        for(int i=1;i<n;i++){
```

```
        if(arr[i] > arr[i-1]){
            output[i] = ++count;
        }else{
            output[i] = 1;
            count = 1;
        }
    }
    //right to left re-traversing to make sure two person adjacent get correct chocolates (ex : 8 7 5) (1,1,1) -> (3,2,1)
    for(int i=n-1;i>=0;i--){
        if(arr[i-1] > arr[i] && output[i-1] <= output[i]){
            output[i-1] = output[i]+1;
        }
    }
    int sum = 0;
    for(int i=0;i<=n;i++){
        sum += output[i];
    }
   return sum;
}

int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int n;
        cin >> n;
        int *arr = new int[n];
        for(int i = 0; i < n; i++){
            cin >> arr[i];
        }
       cout << getMin(arr, n) << endl;
    }

    return 0;
}
```

**Solution :**

```
int getMin(int *arr, int n){

            int dp[n];
                        dp[0] = 1;
                        int i = 0;
          int sum = 0;
        for (i = 1; i < n; i++)
          {
            if (arr[i] > arr[i - 1])
                        {
```

```
            dp[i] = dp[i - 1] + 1;
        }
        else
        dp[i] = 1;
    }
  for (i = n - 2; i >= 0; i--)
    {
        if (arr[i] > arr[i + 1] && dp[i] <= dp[i + 1])
        {
        dp[i] = dp[i + 1] + 1;
        }
    }
        for (i = 0; i < n; i++)
      sum += dp[i];
  return sum;
}
```

## 9-Ass : Minimum Count

Send Feedback

Given an integer N, find and return the count of minimum numbers, sum of whose squares is equal to N. That is, if N is 4, then we can represent it as : {1^2 + 1^2 + 1^2 + 1^2} and {2^2}. Output will be 1, as 1 is the minimum count of numbers required.

**Note : x^y represents x raise to the power y.**

**Input Format :**

First line will contain T(number of test case), each test case consists of a single line containing an integer N.

**Output Format :**

For each test case print the required minimum count in a newline.

**Constraints :**

1 <= T <= 1000

1 <= N <= 1000

**Sample Input 1 :**

1

12

**Sample Output 1 :**

3

**Sample Output 1 Explanation :**

12 can be represented as :

1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 1^1

1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 1^1 + 2^2

1^1 + 1^1 + 1^1 + 1^1 + 2^2 + 2^2

2^2 + 2^2 + 2^2

As we can see, the output should be 3.

### Sample Input 2 :

```
1
9
```

### Sample Output 2 :

```
1
```

```
#include<bits/stdc++.h>
using namespace std;
int minCount(int n)
{

        int dp[n+1];

        dp[0] = 0;
        dp[1] = 1;
  for (int i = 2; i <= n; ++i)
  {
        dp[i] = INT_MAX;
        for (int j = 1; i-(j*j) >= 0; j++)
        {
                dp[i] = min(dp[i],dp[i-(j*j)]);
        }
        dp[i] +=1;
  }

  return dp[n];

}
int main(){

  // write your code here
  int T;cin>>T;
  while(T--){
     int n; cin>>n;
     cout << minCount(n)<<endl;
  }
  return 0;
}
```

**Solution : Same**

## 10-Ass : Hasan and Trip

Send Feedback

Hasan has finally finished his final exams and he decided to go in a trip among cities in Syria.

There are N cities in Syria and they are numbered from 1 to N, each city has coordinates on plane, i-th city is in (Xi, Yi).

Hasan is in first city and he wants to visit some cities by his car in the trip but the final destination should be N-th city and the sequence of cities he will visit should be increasing in index (i.e. if he is in city i he can move to city j if and only if i < j ).

Visiting i-th city will increase Hasan's happiness by Fi units (including first and last cities), also Hasan doesn't like traveling too much, so his happiness will decrease by total distance traveled by him.

Help Hasan by choosing a sequence of cities to visit which maximizes his happiness.

### Input format:

First line will contain T(number of test case).
First line of each test case will contain an integer N
Next N lines of that test case will contain three space-separated integers Xi, Yi, Fi (coordinates and happiness)
Output format:

For each test Output one number rounded to 6 digits after floating point, the maximum possible happiness in newline, Hasan can get.
Note: If answer is 2 print 2.000000
Constraints:

1 <= T <= 50
1 <= N <= 500
0 <= Xi, Yi, Fi <= 100,000
Sample Input

```
1
3
0 0 1
3 1 1
6 0 9
```

Sample Output

```
4.675445
```

```
1. #include<bits/stdc++.h>
2. using namespace std;
3. double inf = 1e15;
4.
5. double disc(pair<double, double> p1, pair<double, double> p2)
6. {
```

```
    double a, b;
    a = (p2.first - p1.first) * (p2.first - p1.first);
    b = (p2.second - p1.second) * (p2.second - p1.second);
    return sqrt(a+b);
}
void solve(pair<double, double> arr[], double happy[], int n)
{
    double dp[n];
    dp[0] = happy[0];
    for(int i=1;i<n;i++)
    {
        dp[i] = -inf;
        for(int j=0;j<i;j++)
        {
            double x = disc(arr[i], arr[j]);
            dp[i] = max(dp[i], dp[j]-x);
        }
        dp[i] += happy[i];
    }
    cout<<fixed;
    cout<<setprecision(6)<<dp[n-1]<<endl;
    return;
}
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        pair<double, double> arr[n];
        double a,b;
        double happy[n];
        for(int i=0;i<n;i++)
        {
            cin>>a>>b;
            arr[i] = make_pair(a,b);
            cin>>happy[i];
        }
        solve(arr,happy,n);
    }
}
```

**Solution : Same**

## 11-Ass : Vanya and GCD

Send Feedback

Vanya has been studying all day long about sequences and other Complex Mathematical Terms. She thinks she has now become really good at it. So, her friend Vasya decides to test her knowledge and keeps the following challenge it front of her:

Vanya has been given an integer array A of size N. Now, she needs to find the number of increasing sub-sequences of this array with length ≥1 and GCD=1. A sub-sequence of an array is obtained by deleting some (or none) elements and maintaining the relative order of the rest of the elements. As the answer may be large, print it Modulo 109+7

She finds this task really easy, and thinks that you can do it too. Can you?

Input Format:

First line will contain T(number of test case), each test consists of two line.
The first line contains a single integer N denoting size of array A.
The next line contains N space separated integers denoting the elements of array A

Output Format:

Print the required answer Modulo 10^9+7 for each test case in new line

Constraints:

1 <= T <= 50
1 <= N <= 200
1 <= A[i] <= 100

Sample Input

```
1
3
1 2 3
```

Sample Output

```
5
```

```
#include<bits/stdc++.h>
using namespace std;
int mod = 1e9 + 7;
int solve(int arr[],int n)
{
    int maxVal = INT_MIN;
    int dp[n+1][105];

    for(int i=0;i<n;i++)
    {
        for(int j=0;j<=100;j++)
        {
```

```
            dp[i][j] = 0;
        }
    }

    for(int i=0;i<n;i++)
    {
        maxVal = max(arr[i], maxVal);
        dp[i][arr[i]] = 1;
    }

    for(int i=1;i<n;i++)
    {
        for(int j=0;j<i;j++)
        {
            if(arr[i] > arr[j])
            {
                for(int k=1;k<=maxVal;k++)
                {
                    int x = __gcd(k,arr[i]);
                    dp[i][x] += dp[j][k];
                    if(dp[i][x] >= mod)
                        dp[i][x] -= mod;
                }
            }
        }
    }

    int ans = 0;
    for(int i=0;i<n;i++)
    {
        ans += dp[i][1];
        if(ans >= mod)
            ans -= mod;
    }
    return ans;
}

int main(){

    // write your code here
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        int arr[n];
        for(int i=0;i<n;i++)
        {
```

```
            cin>>arr[i];
        }
        cout<<solve(arr,n)<<endl;
    }
    return 0;
}
```

**Solution : Same**

## 12-Ass : Roy and Coin Boxes

Send Feedback

Roy has N coin boxes numbered from 1 to N.

Every day he selects two indices [L,R] and adds 1 coin to each coin box starting from L to R (both inclusive).

He does this for M number of days.

After M days, Roy has a query: How many coin boxes have at least X coins.

He has Q such queries.

### Input Format:

First line will contain T (number of test case), format of each test case follows
First line contains two space separated integers N and M (N - number of coin boxes, M - number of days).
Each of the next M lines consists of two space separated integers L and R. Followed by integer Q - number of queries.
Each of next Q lines contain a single integer X.

### Output Format:

For each query of each test case output the result in a new line.

### Constraints:

1 <= T <= 10
1 ≤ N ≤ 10000
1 ≤ M ≤ min(10000, N)
1 ≤ L ≤ R ≤ N
1 ≤ Q ≤ 10000
1 ≤ X ≤ N
Sample Input

```
1
7
4
1 3
2 5
1 2
5 6
4
1
7
4
```

2
Sample Output

6
0
0

4

```
#include<bits/stdc++.h>
using namespace std;
int main(){

    // write your code here
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    int t;
    cin>>t;
    while(t--)
    {
        int n,m,q,l,r;
        cin>>n>>m;
        int start[n+1], end[n+1], coins[n+1], ans[n+1];
        for(int i=0;i<=n;i++)
        {
            start[i] = 0;
            end[i] = 0;
            coins[i] = 0;
            ans[i] = 0;
        }
        for(int i=0;i<m;i++)
        {
            cin>>l>>r;
            start[l]++; //noting the times the starting box was kth box
                                    end[r]++; //noting the times the ending box was kth box
        }
        int temp = 0;
        for(int i=1;i<=n;i++)
        {
            temp += start[i];
            coins[i] = temp;
            temp -= end[i];
        }

        //now calculate how many boxes have exact i coins
        for(int i=1;i<=n;i++)
        {
            ans[coins[i]]++;
        }
```

```
41.         /*now calculate how many boxes have atleast i coins which is same as
42.          number of boxes having atleast i+1 coins + number of boxes having exact i coins
43.          Note: box with atleast max coins is same as box with exact k coins
44.          In other words we can take max as n and start iterating from n-1*/
45.         for(int i=n-1;i>0;i--)
46.         {
47.             ans[i] = ans[i]+ans[i+1];
48.         }
49.         cin>>q;
50.         while(q--)
51.         {
52.             cin>>temp;
53.             cout<<ans[temp]<<endl;
54.         }
55.     }
56.     return 0;
57. }
```

## 13-Ass : Alyona and Spreadsheet

Send Feedback

During the lesson small girl Alyona works with one famous spreadsheet computer program and learns how to edit tables.

Now she has a table filled with integers. The table consists of n rows and m columns. By ai, j we will denote the integer located at the i-th row and the j-th column. We say that the table is sorted in non-decreasing order in the column j if ai, j ≤ ai + 1, j for all i from 1 to n - 1.

Teacher gave Alyona k tasks. For each of the tasks two integers l and r are given and Alyona has to answer the following question: if one keeps the rows from l to r inclusive and deletes all others, will the table be sorted in non-decreasing order in at least one column? Formally, does there exist such j that ai, j ≤ ai + 1, j for all i from l to r - 1 inclusive.

Alyona is too small to deal with this task and asks you to help!

### Input Format:

First line of input will contain T(number of test case), each test case is described as.
The first line of the each test case contains two positive integers n and m the number of rows and the number of columns in the table respectively.
Each of the following n lines contains m integers. The j-th integers in the i of these lines stands for ai, j.

The next line of the input contains an integer k, the number of task that teacher gave to Alyona.

The i-th of the next k lines contains two integers li and ri

### Output Format:

For each test case, print "Yes" to the i-th line of the output if the table consisting of rows from li to ri inclusive is sorted in non-decreasing order in at least one column. Otherwise, print "No".

### Constriants:

1 <= T <= 10

1 <= N, M <=  20000
1 <= N*M <= 20000
1 <= arr[i][j] <= 10^9
1 <= K <= 10000
1 <= l <= r <= N

**Sample Input :**

1
3 11
5 1 3 4 5 1 5 5 3 3 2
5 8 2 10 1 9 8 4 4 3 4
15 6 9 2 7 1 3 13 7 7 5
1
1 3

**Sample Output :**

Yes

```
#include<bits/stdc++.h>
using namespace std;
vector<int> solve(vector<vector<int>> &mat, int n, int m)
{
    vector<int> dp(n, 0);
    int index[n][m];
    for(int i=0;i<m;i++)
    {
        index[0][i] = 0;
        for(int j=1;j<n;j++)
        {
            if(mat[j][i] >= mat[j-1][i])
                index[j][i] = index[j-1][i];
            else
                index[j][i] = j;
        }
    }
    for(int i=0;i<n;i++)
    {
        dp[i] = i;
        for(int j=0;j<m;j++)
        {
            dp[i] = min(dp[i], index[i][j]);
        }
    }
    return dp;
}

int main(){

    // write your code here
    int t;
```

```
    cin>>t;
    while(t--)
    {
        int n,m;
        cin>>n>>m;
        vector<vector<int>> mat(n,vector<int>(m,0));
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
                cin>>mat[i][j];
            }
        }
            vector<int> dp = solve(mat, n, m);
            int q, l, r;
            cin>>q;
            while(q--)
            {
                cin>>l>>r;
                l--;
                r--;
                if(dp[r] <= l)
                    cout<<"Yes\n";
                else
                    cout<<"No\n";
            }
    }


    return 0;
}
```

**Solution :**

## 14-Ass : Angry Children

Send Feedback

Bill Gates is on one of his philanthropic journeys to a village in Utopia. He has N packets of candies and would like to distribute one packet to each of the K children in the village (each packet may contain different number of candies). To avoid a fight between the children, he would like to pick K out of N packets such that the unfairness is minimized.

Suppose the K packets have (x1, x2, x3,....xk) candies in them, where xi denotes the number of candies in the ith packet, then we define unfairness as

```
unfairness=0;
for(i=0;i<n;i++)
   for(j=i;j<n;j++)
```

unfairness+=abs(xi-xj)
abs(x) denotes absolute value of x.

## Input Format:

First line will contain T(number of test cases), and each test case consists of two lines.
The first line contains two space-separated integers N and K.
The second line will contain N space-separated integers, where Ith integer denotes the candy in the Ith packet.

Output Format:

For each test case print a single integer which will be minimum unfairness in newline.
Constraints

1 <= T <= 10
2<=N<=10^5
2<=K<=N
0<= number of candies in each packet <=10^6
Sample Input

1
7 3
10 100 300 200 1000 20 30
Sample Output

40
Explanation

Bill Gates will choose packets having 10, 20 and 30 candies.So unfairness will be |10-20| + |20-30| + |10-30| = 40. We can verify that it will be minimum in this way.

```
#include<bits/stdc++.h>
using namespace std;

long long int solve(int arr[], int n, int k)
{
    sort(arr,arr+n);
    vector<long long> sum(n+1,0);
    long long curr=0,ans;
    sum[0] = 0;
    for(int i=0;i<n;i++)
    {
        sum[i+1] = sum[i] + arr[i];
    }
    for(int i=0;i<k;i++)
    {
        curr += (1ll*i*arr[i]-sum[i]);
```

```
    }
    ans = curr;
    for(int i=k;i<n;i++)
    {
        curr = curr-2ll*(sum[i]-sum[i-k+1]) + 1ll*(k-1)*(arr[i]+arr[i-k]);
        ans = min(ans,curr);
    }
    return ans;
}


int main(){

    // write your code here
     int t;
    cin>>t;
    while(t--)
    {
        int n,k;
        cin>>n>>k;
        int arr[n];
        for(int i=0;i<n;i++)
        {
            cin>>arr[i];
        }
        cout<<solve(arr,n,k)<<endl;
    }
    return 0;
}
```

# L12 : Dynamic Programming-2

## 1-Tut : LCS - Problem

Send Feedback

Given two strings S1 and S2 with lengths M and N respectively, find the length of the longest common subsequence.

A subsequence of a string S whose length is K, is a string containing characters in same relative order as they are present in S, but not necessarily contiguous. Subsequences contain all the strings of length varying from 0 to K. For example, subsequences of string "abc" are -- ""(empty string), a, b, c, ab, bc, ac, abc.

### Input Format :

First line will contain T(number of test case), each test case will consist of two lines.
Line 1: String S1
Line 2: String s2

### Output Format :

Length of the longest common subsequence for each test case in a newline.

### Constraints :

1 <= T <= 100
1 <= M <= 100
1 <= N <= 100
Time Limit: 1 sec

### Sample Input 1:

1
adebc
dcadb

### Sample Output 1: 3

### Explanation of Sample Input 1:

"a", "d", "b", "c", "ad", "ab", "db", "dc" and "adb" are present as a subsequence in both the strings in which "adb" has the maximum length. There are no other common subsequence of length greater than 3 and hence the answer.

### Sample Input 2:

1
abcd
acbdef

### Sample Output 2: 3

### Explanation of Sample Input 2:

"a", "b", "c", "d", "ab", "ac", "ad", "bd", "cd", "abd" and "acd" are present as a subsequence in both the strings S1 and S2 in which "abd" and "acd" are of the maximum length. There are no other common subsequence of length greater than 3 and hence the answer.

```
#include<bits/stdc++.h>
using namespace std;
int lcs(string &str1, string &str2)
{
    int n = str1.size();
    int m = str2.size();
    int dp[n+1][m+1];
    for(int i=0;i<=n;i++)
    {
        for(int j=0;j<=m;j++)
        {
            dp[i][j] = 0;
        }
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            if(str1[i-1] == str2[j-1])
            {
                dp[i][j] = 1+dp[i-1][j-1];
            }
            dp[i][j] = max(dp[i][j], max(dp[i-1][j], dp[i][j-1]));
        }
    }

    int ans = dp[n][m];
    return ans;
}
int main(){
    // write your code here
    int t;
    cin>>t;
    while(t--)
    {
        string str1, str2;
         cin >> str1 >> str2;
        cout << lcs(str1, str2)<<endl;
    }
    return 0;
}
```

## 2-Tut : Edit Distance - Problem

Send Feedback

Given two strings s and t of lengths m and n respectively, find the Edit Distance between the strings. Edit Distance of two strings is minimum number of steps required to make one string equal to other. In order to do so you can perform following three operations only :

1. Delete a character
2. Replace a character with another one
3. Insert a character

**Note - Strings don't contain spaces**

**Input Format :**

First line of input will contain T (number of test cases), each test case consists of two lines.
Line 1 : String s
Line 2 : String t

**Output Format :**

For each test case print the Edit Distance value in new line.

**Constraints:**

1 <= T <= 100
1<= m,n <= 100

**Sample Input 1 :**

```
1
abc
dc
```

**Sample Output 1 :**

```
2
```

```cpp
#include<bits/stdc++.h>
using namespace std;

int Edit_Distance(string &s1,string &s2){
    int n=s1.size();
    int m=s2.size();
    int dp[n+1][m+1];

    for(int i=0;i<=n;i++){
        for(int j=0;j<=m;j++){
            if(i==0){
                dp[i][j]=j;
            }
            else if(j==0){
                dp[i][j]=i;
            }
            else{
                if(s1[i-1]==s2[j-1]){
```

```
                dp[i][j]=dp[i-1][j-1];
            }
            else{
                int f1=1+dp[i-1][j-1];
                int f2=1+dp[i-1][j];
                int f3=1+dp[i][j-1];
                dp[i][j]=min(f1,f2);
                dp[i][j]=min(dp[i][j],f3);
            }
        }
    }
    return dp[n][m];
}

int main(){

    // write your code here
    int t;
    cin>>t;
    while(t--){
        string s1,s2;
        cin>>s1>>s2;
        cout<<Edit_Distance(s1,s2)<<endl;
    }
    return 0;
}
```

**Solution :**

```
/* Time complexity: O(N*M) Space complexity : O(N*M) where N and M are the size of input strings */
#include using namespace std; int helper(string &s1, string &s2, int dp[][102], int len1, int len2) { if (len1 == 0 || len2 == 0) { dp[len1][len2] = max(len1, len2); return dp[len1][len2]; } if (dp[len1][len2] != -1) { return dp[len1][len2];
```

## 3-Tut : Knapsnack - Problem

Send Feedback

A thief robbing a store and can carry a maximal weight of W into his knapsack. There are N items and ith item weigh wi and is of value vi. What is the maximum value V, that thief can take ?

Note: Space complexity should be O(W).

### Input Format :

Line 1 : N i.e. number of items
Line 2 : N Integers i.e. weights of items separated by space
Line 3 : N Integers i.e. values of items separated by space
Line 4 : Integer W i.e. maximum weight thief can carry

### Output Format :

Line 1 : Maximum value V

### Constraints

1 <= N <= 10^4
1<= wi <= 100
1 <= vi <= 100
1 <= W <= 1000

### Sample Input 1 :

```
4
1 2 4 5
5 4 8 6
5
```

### Sample Output :

```
13
```

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

ll helper_(int* weights, int* values, int n, int maxWeight,ll **storage){
    if(n<0){
      return 0;
    }
    if(storage[n][maxWeight]!=-1){
      return storage[n][maxWeight];
    }
    if(weights[n]<=maxWeight){
      ll inc=values[n]+helper_(weights,values,n-1,maxWeight-weights[n],storage);
      // not including last num
      ll notinc=helper_(weights,values,n-1,maxWeight,storage);
      ll ans=max(inc,notinc);
      storage[n][maxWeight]=ans;
      return ans;
```

```
19.
20.     }else{
21.
22.       ll ans=helper_(weights,values,n-1,maxWeight,storage);
23.       storage[n][maxWeight]=ans;
24.       return ans;
25.
26.     }
27. }
28.
29. int knapsack(int* weights, int* values, int n, int maxWeight){
30.             ll **storage=new ll*[n+1];
31.             for(int i=0;i<=n;i++){
32.               storage[i]=new ll[maxWeight+1];
33.             }
34.             for(int i=0;i<=n;i++){
35.               for(int j=0;j<=maxWeight;j++){
36.                 storage[i][j]=-1;
37.               }
38.             }
39.     ll ans=helper_(weights,values,n-1,maxWeight,storage);
40.     ans=(int)ans;
41.     for(int i=0;i<=n;i++){
42.         delete storage[i];
43.     }
44.     delete storage;
45.     return ans;
46. }
47.
48. int main(){
49.     // write your code here
50.   int n;
51.   cin >> n;
52.   int* weights = new int[n];
53.   int* values = new int[n];
54.
55.   for(int i = 0; i < n; i++){
56.     cin >> weights[i];
57.   }
58.
59.   for(int i = 0; i < n; i++){
60.     cin >> values[i];
61.   }
62.
```

```
63.  int maxWeight;
64.  cin >> maxWeight;
65.
66.  cout << knapsack(weights, values, n, maxWeight);
67.    return 0;
68. }
```

## 4-Tut : PARTY - Problem

Send Feedback

You just received another bill which you cannot pay because you lack the money.

Unfortunately, this is not the first time to happen, and now you decide to investigate the cause of your constant monetary shortness. The reason is quite obvious: the lion's share of your money routinely disappears at the entrance of party localities.

You make up your mind to solve the problem where it arises, namely at the parties themselves. You introduce a limit for your party budget and try to have the most possible fun with regard to this limit.

You inquire beforehand about the entrance fee to each party and estimate how much fun you might have there. The list is readily compiled, but how do you actually pick the parties that give you the most fun and do not exceed your budget?

Write a program which finds this optimal set of parties that offer the most fun. Keep in mind that your budget need not necessarily be reached exactly. Achieve the highest possible fun level, and do not spend more money than is absolutely necessary.

Input Format:

First line of input will contain an integer N (number of parties).
Next line of input will contain N space-separated integers denoting the entry fee of Ith party.
Next line will contain N space-separated integers denoting the amount of fun Ith party provide.
Last line of input will contain an integer W party budget.

Output Format:

For each test case your program must output the sum of the entrance fees and the sum of all fun values of an optimal solution. Both numbers must be separated by a single space.
Note: In case of multiple cost provides the maximum fun output the minimum total cost.

**Sample Input:**

```
5
1 7 9 7 2
5 5 2 4 7
12
```

**Sample Output:**

```
10 17
```

```
1. #include<bits/stdc++.h>
2. #include<utility>
3. using namespace std;
4.
```

```
5.  pair<int, int> most_fun(int* enterance_fees, int* fun, int budget, int n, pair<int, int>**dp)
6.  {
7.          if (n == 0)
8.          {
9.                  pair<int, int> p;
10.                 p.first = 0;
11.                 p.second = 0;
12.                 return p;
13.         }
14.         if (dp[budget][n].first != -1 && dp[budget][n].second != -1)
15.         {
16.                 return dp[budget][n];
17.         }
18.         pair<int, int> ans;
19.         if (enterance_fees[0] <= budget)
20.         {
21.                 pair<int, int>option1 = most_fun(enterance_fees + 1, fun + 1, budget -
    enterance_fees[0], n - 1, dp);
22.                 option1.first += enterance_fees[0];
23.                 option1.second += fun[0];
24.                 pair<int, int>option2 = most_fun(enterance_fees + 1, fun + 1, budget, n -
    1, dp);
25.                 if (option1.second > option2.second)
26.                 {
27.                         ans= option1;
28.                 }
29.                 else if (option2.second > option1.second)
30.                 {
31.                         ans= option2;
32.                 }
33.                 else
34.                 {
35.                         if (option1.first < option2.first)
36.                         {
37.                                 ans= option1;
38.                         }
39.                         else
40.                         {
41.                                 ans= option2;
42.                         }
43.                 }
44.         }
45.         else
46.         {
```

```
                ans= most_fun(enterance_fees + 1, fun + 1, budget, n - 1, dp);
        }
        dp[budget][n] = ans;
        return ans;
}


int main(){

    // write your code here
    while (true)
        {
            int budget;
            int n;
            cin >> n;
            if (budget == 0 && n == 0)
            {
                break;
            }
            int* enterance_fees = new int[n];
            int* fun = new int[n];
            for (int i = 0; i < n; i++)
            {
                cin >> enterance_fees[i];
            }
        for(int i=0;i<n;i++){
       cin>> fun[i];
    }
            cin >> budget;

            pair<int, int>** dp = new pair<int, int> * [budget+1];
            for (int i = 0; i < budget+1; i++)
            {
                dp[i] = new pair<int, int>[n + 1];
                for (int j = 0; j < n + 1; j++)
                {
                    dp[i][j].first = -1;
                    dp[i][j].second = -1;
                }
            }


            pair<int, int> p;
            p = most_fun(enterance_fees, fun, budget, n, dp);
```

```
                cout << p.first << ' ' << p.second << endl;
        exit(0);



                for (int i = 0; i < budget + 1; i++)
                {
                        delete[]dp[i];
                }
                delete[]dp;
          }
        return 0;
    }
```

## 5-Tut : Subset Sum - Problem

Send Feedback

Given an array of n integers, find if a subset of sum k can be formed from the given set. Print Yes or No.

### Input Format

First-line will contain T(number of test cases), each test case consists of three lines.
First-line contains a single integer N(length of input array).
Second-line contains n space-separated integers denoting the elements of array.
The last line contains a single positive integer k.

### Output Format

Output Yes if there exists a subset whose sum is k, else output No for each test case in new line.

### Constraints:

1 <= T <= 100
1 <= N <= 500
1 <= arr[i] <= 10^4
1 <= K <= 500
Sample Input

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

Sample Output

```
Yes
```

```
1. #include<bits/stdc++.h>
2. using namespace std;
3.
4. bool targetSumSubset(int n, vector<int> arr, int tar) {
5.     bool dp[n+1][tar+1]={true};
```

```
    for(int i=0;i<n+1;i++){
        for(int j=0;j<tar+1;j++){
            if(i==0 and j==0){
                dp[i][j]=true;
            }
            else if(i==0){
                dp[i][j]=false;
            }
            else if(j==0){
                dp[i][j]=true;
            }
            else{
                if(dp[i-1][j]==true){
                    dp[i][j]=true;
                }
                else{
                    int val=arr[i-1];
                    if(j>=val){
                        if(dp[i-1][j-val]==true){
                            dp[i][j]=true;
                        }
                    }
                }
            }
        }
    }
   return dp[arr.size()][tar];
}

int main(){

    // write your code here

    int t;
    cin>>t;
    while(t--)
    {
            int n;
      cin >> n;
      vector<int> arr(n);
      for (int i = 0; i < arr.size(); i++) {
        cin >> arr[i];
            }
            int tar;
```

```
        cin >> tar;
    bool a=  targetSumSubset(n, arr, tar);
    if(a==1)
    {
        cout<<"Yes"<<endl;
    }
    else{
        cout<<"No"<<endl;
                    }
    }
    return 0;
}
```

## 6-Tut : Maximum Sum Rectangle

Send Feedback

Given a 2D array, find the maximum sum rectangle in it. In other words find maximum sum over all rectangles in the matrix.

### Input Format:

First line of input will contain T(number of test case), each test case follows as.
First line contains 2 numbers n and m denoting number of rows and number of columns. Next n lines contain m space separated integers denoting elements of matrix nxm.
Output Format:

Output a single integer, maximum sum rectangle for each test case in a newline.
Constraints

1 <= T <= 50
1<=n,m<=100
-10^5 <= mat[i][j] <= 10^5

### Sample Input

```
1
4 5
1 2 -1 -4 -20
-8 -3 4 2 1
3 8 10 1 3
-4 -1 1 7 -6
```

### Sample Output

```
29
```

```
#include<bits/stdc++.h>
using namespace std;

int kadane(vector<int>v,int n){
    int ma=INT_MIN;
```

```
    int u=0;
    for(int i=0;i<n;i++){
        u+=v[i];
        if(u>ma){
            ma=u;
        }
        if(u<0){
            u=0;
        }
    }
    return ma;
}

int main(){

    // write your code here
    int t;
    cin>>t;
    while(t--){
        int n,m;
        cin>>n>>m;
        vector<vector<int>>mat(n,vector<int>(m));
                    for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                cin>>mat[i][j];
            }
            }


        int ma=INT_MIN;
        for(int i=0;i<n;i++){
            vector<int>ans(m);
            for(int j=i;j<n;j++){
                for(int col=0;col<m;col++) {
                    ans[col]+=mat[j][col];
                }
                ma=max(ma,kadane(ans,m));
            }
        }
        cout<<ma<<endl;

            }
    return 0;
}
```

# 7-Tut : Mehta and Bank Robbery - Problem

Send Feedback

One fine day, when everything was going good, Mehta was fired from his job and had to leave all the work. So, he decided to become a member of gangster squad and start his new career of robbing. Being a novice, mehta was asked to perform a robbery task in which he was given a bag having a capacity W units. So, when he reached the house to be robbed, there lay N items each having particular weight and particular profit associated with it. But, theres a twist associated, He has first 10 primes with him, which he can use atmost once, if he picks any item x, then he can multiply his profit[x] with any of the first 10 primes and then put that item into his bag. Each prime can only be used with one particular item and one item can only have atmost one prime multiplied with its profit. Its not necessary to pick all the items. If he doesnt want to use a prime with any particular item, he can simply add the profit as it is, more specifically, 1*profit[x] for xth item will get added to its total profit, and that he can do with as many items as he wants. He cannot fill his bag more than weight W units. Each item should be picked with its whole weight, i.e it cannot be broken into several other items of lesser weight. So, now to impress his squad, he wishes to maximize the total profit he can achieve by robbing this wealthy house.

### Input Format:

The first line of input will contain T(number of test cases), each test will follow as.
First Line will contain two integers N and W (number of items and maximum weight respectively).
Second-line will contain N space-separated integers denoting the profit associated with the Ith item.
The third line will contain N space-separated integers denoting the weight of the Ith item.

### Output Format:

Output the maximum profit obtainable for each test case in a new line.

### Constraints:

1 <= T <= 20
1 <= N, W <= 500
1 <= profit[i] <= 10^4
1 <= weight[i] <= 10^4

### Sample Input:

```
1
7 37
33 5 14 14 16 25 15
5 19 30 4 15 31 25
```

### Sample output:

```
1591
```

```
1. #include<bits/stdc++.h>
2. using namespace std;
3.
4. int solve(vector<pair<int, int>> &arr, int n,int w)
5. {
6.     int dp[2][n+1][w+1];
```

```
7.      memset(dp, 0, sizeof(dp));
8.      sort(arr.begin(), arr.end());
9.
10.     int primes[11] = {1,2,3,5,7,11,13,17,19,23,29};
11.
12.     for(int i=1;i<=n;i++)
13.     {
14.         for(int j=1;j<=w;j++)
15.         {
16.             dp[0][i][j] = dp[0][i-1][j];
17.
18.             if(j>=arr[i-1].second)
19.             {
20.                 dp[0][i][j] = max(dp[0][i][j], dp[0][i-1][j-arr[i-1].second] + arr[i-1].first);
21.             }
22.         }
23.     }
24.
25.     for(int prime = 1;prime<=10;prime++)
26.     {
27.         int p = prime%2;
28.         for(int i=1;i<=n;i++)
29.         {
30.             for(int j=1;j<=w;j++)
31.             {
32.                 dp[p][i][j] = dp[p][i-1][j];
33.                 if(j>=arr[i-1].second)
34.                 {
35.                     int temp = max(dp[p][i-1][j-arr[i-1].second] + arr[i-1].first,
36.                                    dp[p^1][i-1][j-arr[i-1].second] + arr[i-1].first * primes[prime]);
37.
38.                     dp[p][i][j] = max(dp[p][i][j], temp);
39.                 }
40.             }
41.         }
42.     }
43.     return dp[0][n][w];
44. }
45.
46. int main(){
47.
48.     // write your code here
49.      int t;
50.     cin>>t;
```

```
    while(t--)
    {
        int n,w;
        cin>>n>>w;
        vector<pair<int, int>> arr(n);
        for(int i=0;i<n;i++)
        {
            cin>>arr[i].first;
        }
        for(int i=0;i<n;i++)
        {
            cin>>arr[i].second;
        }
        cout<<solve(arr, n, w)<<endl;
    }

    return 0;
}
```

## 8-Ass : Miser Man

Send Feedback

Jack is a wise and miser man. Always tries to save his money.

One day, he wants to go from city A to city B. Between A and B, there are N number of cities(including B and excluding A) and in each city there are M buses numbered from 1 to M. And the fare of each bus is different. Means for all N*M busses, fare (K) may be different or same. Now Jack has to go from city A to city B following these conditions:

1. At every city, he has to change the bus.

2. And he can switch to only those buses which have number either equal or 1 less or 1 greater to the previous.

You are to help Jack to go from A to B by spending the minimum amount of money.

### Input Format:

First-line will contain T(number of the test case), each test case follows as.
First-line will contain two space-separated integers N and M.
Next, N rows will contain M space-separated integers denoting the elements of the grid.
Each row lists the fares the M busses to go form the current city to the next city.

### Output Format:

For each test case print the minimum amount of fare that Jack has to give in a newline.

### Constraints:

1 <= T <= 100
1 <= N, M <= 100
1 <= arr[i][j] <= 10^5

**Sample Input**

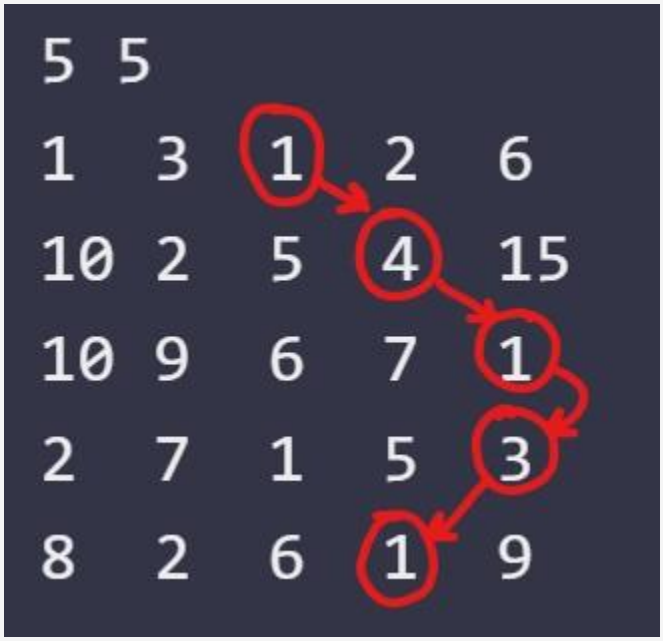
```
1
5 5
1 3 1 2 6
10 2 5 4 15
10 9 6 7 1
2 7 1 5 3
8 2 6 1 9
```

**Sample Output**

```
10
```

**Explanation:**

1 -> 4 -> 1 -> 3 -> 1: 10
This is marked and shown in the following image:




```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.
4.  int save[105][105];
5.
6.  int main(){
7.
8.      // write your code here
9.
10.     int t;
11.    cin>>t;
12.    while(t--){
```

```
13.         int n,m;
14.         cin>>n>>m;
15.         int i,j;
16.
17.         for(i=0;i<n;i++)
18.                 for(j=0;j<m;j++)
19.                         cin>>save[i][j];
20.         int res[n][m];
21.         for(i=0;i<m;i++)
22.                 res[0][i]=save[0][i];
23.         for(i=1;i<n;i++)
24.         {
25.                 for(j=0;j<m;j++)
26.                 {
27.                         int l,u,r;
28.                         l=(j>0)?(res[i-1][j-1]):INT_MAX;
29.
30.                         u=res[i-1][j];
31.                         r=(j<m-1)?res[i-1][j+1]:INT_MAX;
32.                         res[i][j]=save[i][j]+min(u,min(l,r));
33.                 }
34.         }
35.         int ans=INT_MAX;
36.         for(i=0;i<m;i++)
37.                 if(res[n-1][i]<ans)
38.                         ans=res[n-1][i];
39.         cout<<ans<<endl;
40.     }
41.
42.     return 0;
43. }
```

## 9-Ass : Trader Profit

Send Feedback

Mike is a stock trader and makes a profit by buying and selling stocks. He buys a stock at a lower price and sells it at a higher price to book a profit. He has come to know the stock prices of a particular stock for n upcoming days in future and wants to calculate the maximum profit by doing the right transactions (single transaction = buying + selling). Can you help him maximize his profit?

Note: A transaction starts after the previous transaction has ended. Two transactions can't overlap or run in parallel.

The stock prices are given in the form of an array A for n days.

Given the stock prices and a positive integer k, find and print the maximum profit Mike can make in at most k transactions.

### Input Format:

The first line of input contains an integer T(number of test cases).
The first line of each test case contains a positive integer k, denoting the number of transactions.
The second line of each test case contains a positive integer n, denoting the length of the array A.
The third line of each test case contains n space-separated positive integers, denoting the prices of each day in the array A.

### Output Format

For each test case print the maximum profit earned by Mike on a new line.

### Constraints:

1 <= T <= 10^3
0 < k <= 10
2 <= n <= 10^4
0 <= elements of array A <= 10^5

### Sample Input

```
3
2
6
10 22 5 75 65 80
3
4
20 580 420 900
1
5
100 90 80 50 25
```

### Sample Output

```
87
1040
0
```

### Explanation

Output 1: Mike earns 87 as the sum of 12 and 75 i.e. Buy at price 10, sell at 22, buy at 5 and sell at 80.
Output 2: Mike earns 1040 as the sum of 560 and 480 i.e. Buy at price 20, sell at 580, buy at 420 and sell at 900.

Output 3: Mike cannot make any profit as the selling price is decreasing day by day. Hence, it is not possible to earn anything.

```
1. #include<bits/stdc++.h>
2. using namespace std;
3.
4. int max_profit(int n,int k,int *prices,int curr_i,int ongoing,int ***dp){
5.     //if end of array reached
6.     if(curr_i==n||k==0){
7.         return 0;
8.     }
```

```
    if(dp[curr_i][k][ongoing]!=-1){
        return dp[curr_i][k][ongoing];
    }
    //ignore current num
    int opt1=max_profit(n,k,prices,curr_i+1,ongoing,dp);
    //buy or sell
    int opt2=INT_MIN;
    if(ongoing==1){
        //we can sell
                    opt2=prices[curr_i]+max_profit(n,k-1,prices,curr_i+1,0,dp);
    }else{
        //we can buy
        //buy only if k>0
        if(k>0){
            opt2=max_profit(n,k,prices,curr_i+1,1,dp)-prices[curr_i];
        }
    }
    int ans=max(opt1,opt2);
    dp[curr_i][k][ongoing]=ans;
    return ans;
}

int main(){

    // write your code here

     int q;
    cin>>q;
    while(q--){
        int k,n;
        cin>>k;
        cin>>n;
        int *prices=new int[n+1];
        for(int i=0;i<n;i++){
            cin>>prices[i];
        }
        int ***dp=new int**[n+1];
        for(int i=0;i<=n;i++){
            dp[i]=new int*[k+1];
            for(int j=0;j<=k;j++){
                dp[i][j]=new int[2];
                dp[i][j][0]=-1;
                dp[i][j][1]=-1;
            }
```

```
        }
        cout<<max_profit(n,k,prices,0,0,dp)<<endl;
        for(int i=0;i<=n;i++){
            delete dp[i];
        }
        delete dp;
        delete prices;
    }

    return 0;
}
```

## 10-Ass : Charlie and Pilots

Send Feedback

Charlie acquired airline transport company and to stay in business he needs to lower the expenses by any means possible. There are N pilots working for his company (N is even) and N/2 plane crews needs to be made. A plane crew consists of two pilots - a captain and his assistant. A captain must be older than his assistant. Each pilot has a contract granting him two possible salaries - one as a captain and the other as an assistant. A captain's salary is larger than assistant's for the same pilot. However, it is possible that an assistant has larger salary than his captain. Write a program that will compute the minimal amount of money Charlie needs to give for the pilots' salaries if he decides to spend some time to make the optimal (i.e. the cheapest) arrangement of pilots in crews.

### Input Format:

First line will contain T(number of test case), each test case as follow.
The first line of each test case contains integer N, N is even, the number of pilots working for the Charlie's company.
The next N lines of input contain pilots' salaries. The lines are sorted by pilot's age, the salaries of the youngest pilot are given the first. Each of those N lines contains two integers separated by a space character, a salary as a captain (X) and a salary as an assistant (Y).

### Constraints

1 <= T <= 20
2 ≤ N ≤ 1000
1 ≤ Y < X ≤ 100000

### Output Format:

For each test case print the minimal amount of money Charlie needs to give for the pilots' salaries. in newline.

### Sample Input

```
1
4
5000 3000
6000 2000
8000 1000
9000 6000
```

**Sample Output**

19000

**Explanation**

Out of various possible, optimal arrangements will be:
Plane Crew 1 will have Pilot1 as an assistant and Pilot2 as a Captain
Plane Crew2 will have Pilot3 as an assistant and Pilot4 as a Captain

Amount of money required= 3000+6000+1000+9000 =19000.

```
#include<bits/stdc++.h>
using namespace std;

int dp[5001][5001];

int charlie_pilot(int cap,int ast,int n,vector<int>&cpt,vector<int>&ass)
{
    if(cap+ast>=n){
        return 0;
    }
    if(dp[cap][ast]!=-1){
        return dp[cap][ast];
    }
    int ans=INT_MAX;
    if(cap<n/2 and cap<ast){
        ans=min(ans,charlie_pilot(cap+1,ast,n,cpt,ass)+cpt[ast+cap]);
    }
    if(ast<n/2){
        ans=min(ans,charlie_pilot(cap,ast+1,n,cpt,ass)+ass[ast+cap]);
    }
    dp[cap][ast]=ans;
    return dp[cap][ast];
}

int main(){

    // write your code here

     int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        vector<int>cpt(n);
        vector<int>ass(n);
```

```
        for(int i=0;i<n;i++){
            cin>>cpt[i]>>ass[i];
        }
        memset(dp,-1,sizeof(dp));
        cout<<charlie_pilot(0,0,n,cpt,ass)<<endl;
    }

    return 0;
}
```

## 11-Ass : Square Brackets

Send Feedback

### You are given:

a positive integer n,
an integer k, 1<=k<=n,
an increasing sequence of k integers 0 < s1 < s2 < ... < sk <= 2n.
What is the number of proper bracket expressions of length 2n with opening brackets appearing in positions s1, s2,...,sk?

Illustration

### Several proper bracket expressions:

[[]][[[]][]]
[[[][]]][][[]]

### An improper bracket expression:

[[[][]]][]][[]]
There is exactly one proper expression of length 8 with opening brackets in positions 2, 5 and 7.

Task

Write a program which for each data set from a sequence of several data sets:

1. reads integers n, k and an increasing sequence of k integers from input,

2. computes the number of proper bracket expressions of length 2n with opening brackets appearing at positions s1,s2,...,sk,

3. writes the result to output.

Note: since result can be pretty large output the answer % mod (10^9 + 7).

### Input Format:

The first line of the input file contains one integer T(number of test cases), each test case follows as.
The first line contains two integers n and k separated by single space.
The second line contains an increasing sequence of k integers from the interval [1;2n] separated by single spaces.

### Output Format:

For each test case print the number of balanced square bracket sequence % mod (10^9 + 7), that can be formed using the above rules in a new line.

**Constraints:**

1 <= T <= 100
1 <= N <= 100
1 <= K <= N

Sample Input

5
1 1
1
1 1
2
2 1
1
3 1
2
4 2
5 7

Sample Output

1
0
2
3
2

Explanation

Output1: Proper bracket expressions of length 2 with opening brackets appearing in position 1- [].
Output2: Proper bracket expressions of length 2 with opening brackets appearing in position 2 - none.

Output3: Proper bracket expressions of length 4 with opening brackets appearing in position 1 - [][], [[]].

```
#include<bits/stdc++.h>
using namespace std;

int mod = 1e9 + 7;
bool openings[201];
int dp[201][201];

int possibleBrackets(int open,int close,int n)
{
    if(openings[0])
    {
        return 0;
    }

    if(dp[open][close] != -1)
    {
```

```
17.         return dp[open][close];
18.     }
19.
20.     if(open > n || close > n)
21.     {
22.         return 0;
23.     }
24.
25.     if(open == n && close == n)
26.     {
27.         dp[open][close] = 1;
28.         return 1;
29.     }
30.
31.     int currIndex = (open+close+1);
32.
33.     if(open == close || openings[currIndex])
34.     {
35.         dp[open][close] = possibleBrackets(open+1,close,n);
36.     }
37.     else if(open == n)
38.     {
39.         dp[open][close] = possibleBrackets(open,close+1,n);
40.     }
41.     else
42.     {
43.         dp[open][close] = (possibleBrackets(open+1,close,n) +
    possibleBrackets(open,close+1,n))%mod;
44.     }
45.     return dp[open][close];
46. }
47.
48. int main(){
49.
50.     // write your code here
51.
52.     int d;
53.     cin >> d;
54.
55.     while(d--)
56.     {
57.         int n,k;
58.         cin >> n >> k;
59.         memset(openings,0,sizeof(openings));
```

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

        for(int i = 0; i < k ; i++)
        {
            int m;
            cin >> m;
            openings[m] = true;
        }
        cout<<possibleBrackets(0,0,n)<<endl;
    }
    return 0;
}
```

## 12-Ass : Distinct Subsequences

Send Feedback

Given a string, count the number of distinct subsequences of it ( including empty subsequence ). For the uninformed, A subsequence of a string is a new string which is formed from the original string by deleting some of the characters without disturbing the relative positions of the remaining characters.

For example, "AGH" is a subsequence of "ABCDEFGH" while "AHG" is not.

### Input Format:

First line of input contains an integer T which is equal to the number of test cases.
Each of next T lines contains a string s.

### Output Format:

Output consists of T lines. Ith line in the output corresponds to the number of distinct subsequences of ith input string. Since, this number could be very large, you need to output ans%(10^9 + 7) where ans is the number of distinct subsequences.

### Constraints:

T ≤ 100
1 <= length(S) ≤ 10^5
All input strings shall contain only uppercase letters.

```
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
#define mod 1000000007
ll all_subseq(string s){
    int len=s.length();

    ll *dp=new ll[len+1];

    ll *previndice=new ll[26];
    for(int i=0;i<26;i++){
```

```
        previndice[i]=-1;
    }

    dp[0]=1;
    for(ll i=1;i<=len;i++){
        dp[i]=(dp[i-1]*2)%mod;

        if(previndice[s[i-1]-65]!=-1){

            dp[i]=(dp[i]-dp[previndice[s[i-1]-65]-1]+mod)%mod;
        }

            previndice[s[i-1]-65]=i;
    }
    ll ans=dp[len]%mod;
    delete dp;
            delete previndice;
    return ans;
}

int main(){
    // write your code here
     int t;
    cin>>t;
    while(t--){
        string s;
        cin>>s;
        map<string,int> m;
        cout<<all_subseq(s)<<endl;

    }
    return 0;
}
```

## 13-Ass : Smallest Super-Sequence

Send Feedback

Given two strings S and T, find and return the length of their smallest super-sequence.

A shortest super sequence of two strings is defined as the shortest possible string containing both strings as subsequences.

**Note that if the two strings do not have any common characters, then return the sum of lengths of the two strings.**

**Input Format:**

First line will contain T(number of test case), each test consists of two lines.

Line 1 : A string
Line 2: Another string

**Output Format:**

Length of the smallest super-sequence of given two strings for each test case in new line.

**Constraints:**

1 <= T <= 50
1 <= |str1|, |str2| <= 500

**Sample Input:**

1
ab
ac

**Sample Output:**

3

**Sample Output Explanation:**

Their smallest super-sequence can be "abc" which has length=3.

```
#include<bits/stdc++.h>
using namespace std;

int smallestSuperSequenceHelper(string s1,string s2,int m,int n,int **dp){

    for(int i=m;i>=0;i--){
        for(int j=n;j>=0;j--){
            if(s1[i]==s2[j]){
                dp[i][j]=dp[i+1][j+1]+1;
            }
            else{
                dp[i][j]=min(dp[i+1][j],dp[i][j+1])+1;
            }
        }
    }

    return dp[0][0];
}

int smallestSuperSequence(string s1,string s2){
    int m=s1.length(),n=s2.length();
    int **dp=new int*[m+1];
    for(int i=0;i<=m;i++){
        dp[i]=new int[n+1];
    }
    int count=0;
    for(int i=n;i>=0;i--){
        dp[m][i]=count++;
```

```
    }
    count=0;
    for(int i=m;i>=0;i--){
        dp[i][n]=count++;
    }

    int ans=smallestSuperSequenceHelper(s1,s2,m-1,n-1,dp);
    return ans;
}

int main(){

    // write your code here
    int t;
    cin>>t;
    while(t--)
    {
        string s1,s2;
        cin>>s1>>s2;
        cout<<smallestSuperSequence(s1,s2)<<endl;
    }

    return 0;
}
```

## 14-Ass : Shortest Subsequence

Send Feedback

Gary has two string S and V. Now Gary wants to know the length shortest subsequence in S such that it is not a subsequence in V.

**Note: input data will be such so there will always be a solution.**

**Input Format :**

Line 1 : String S of length

Line 2 : String V of length

**Output Format :**

Length of shortest subsequence in S such that it is not a subsequence in V

**Constraints:**

1 <= |S|, |V| <= 1000

**Sample Input :**

babab

babba

**Sample Output :**

3

```
#include<bits/stdc++.h>
using namespace std;

int solve(string s1,string s2)
{
	int m =s1.size();
	int n =s2.size();

	vector<vector<int>> dp(m+1, vector<int>(n+1, 0));

	for (int i = 0; i < n+1; ++i)
	{
		dp[m][i] = n-i;
	}

	for (int i = 0; i < m+1; ++i)
	{
		dp[i][n] = m-i;
	}

	for (int i = m-1; i >= 0; --i)
	{
		for (int j = n-1; j >= 0; --j)
		{
			int k = j;

			while(k<s2.size())
			{
				if(s2[k]==s1[i])
				{
					break;
				}
				k++;
			}
			if (k==s2.size())
			{
				dp[i][j] = 1;
				//return 1;
			}

			int c1 = 1+dp[i+1][k+1];
			int c2 = dp[i+1][j];

			dp[i][j] = min(c1, c2);
```

```
                    }
                }

                return dp[0][0];


}

int main(){

    // write your code here
   // ios_base::sync_with_stdio(false) ;
                //cin.tie(NULL) ;

                string S,V;
                cin>>S>>V;
                cout<<solve(S,V)<<endl;

    return 0;
}
```

## 15-Ass : Balika Vadhu- Problem

Send Feedback

Anandi and Jagya were getting married again when they have achieved proper age. Dadi Sa invited Alok Nath to do the kanyadaan and give blessings. Alok Nath has 2 blessings. Each bessing is in the form of a string consisting of lowercase charaters(a-z) only. But he can give only one blessing of K length because some priest told him to do so. Thus he decides to generate a blessing using the other two blessings. While doing this he wants to ensure that happiness brought into their life by his blessing is maximum. The generated blessing is a common subsequence of length K of the two blessings he has. Happiness of the blessing he generates is calculated by the sum of ASCII values of characters in the blessing and he wants the happiness to be maximum. If he is not able to generate a common subsequence of length K then the happiness is 0 (zero). Alok Nath comes to you and asks you to find the maximum happiness that can be generated by the two blessings he has.

### Input Format:

First line consists of number of test cases T.
Each test case consists of two strings b1 (blessing 1),b2 (blessing 2) and an integer K, each of them in separate lines.

### Output Format:

Output consists of T lines each containing an integer denoting the maximum happiness value that can be generated by the two blessings.

### Constraint:

1 <= T <= 50

1 <= length(b1) , length(b2) <= 100
1 <= K <= 100
Sample Input

2
asdf
asdf
3
anandi
jagya
3
Sample Output

317
0
Explanation

Output1: Maximum happiness value that can be generated by the two blessings is 317 from the sum of ASCII values of characters in the common subsequence "sdf" of length 3.

Output2: There is no way to generate a common subsequence of length 3, hence the happiness is 0.

```
#include<bits/stdc++.h>
using namespace std;

int getCount(char* s1,char* s2,int m,int n,int k,int*** dp)
{
    if(m == 0 || n == 0){
        return 0;
    }

    if(k == 0){
        return 0;
    }

    if(k > m || k > n){
        return 0;
    }

    if(dp[m][n][k] > -1){
        return dp[m][n][k];
    }

    int ans = 0;
    if(s1[0] == s2[0]){
        int ascii = s1[0];
```

```
25.         int option1 = ascii + getCount(s1+1,s2+1,m-1,n-1,k-1,dp);//when included : get min
    sum for next k-1
26.         if(option1 - ascii == 0 && k > 1){
27.             option1 = 0;
28.         }
29.         int option2 = getCount(s1+1,s2,m-1,n,k,dp);//s1[0] = s2[0] but excluded to check
    wether we get min sum from remaining
30.         int option3 = getCount(s1,s2+1,m,n-1,k,dp);
31.         ans = max(option1, max(option2,option3));
32.     }else{
33.         int option1 = getCount(s1+1,s2,m-1,n,k,dp);//look for min sum in next characters
34.         int option2 = getCount(s1,s2+1,m,n-1,k,dp);
35.         ans = max(option1,option2);
36.     }
37.     dp[m][n][k] = ans;
38.     return ans;
39. }
40.
41.
42. int main(){
43.
44.     // write your code here
45.
46.      int t;
47.     cin >> t;
48.     while(t--){
49.         string s1,s2;
50.         cin >> s1;
51.         cin >> s2;
52.         int k;
53.         cin >> k;
54.         int m = s1.length();
55.         int n = s2.length();
56.         char* arr1 = new char[m];
57.         char* arr2 = new char[n];
58.         for(int i=0;i<m;i++){
59.             arr1[i] = s1[i];
60.         }
61.         for(int i=0;i<n;i++){
62.             arr2[i] = s2[i];
63.         }
64.
65.         int*** dp = new int**[m+1];
66.         for(int i=0;i<=m;i++){
```

```
            dp[i] = new int*[n+1];
            for(int j=0;j<=n;j++){
                dp[i][j] = new int[k+1];
                for(int a=0;a<=k;a++){
                    dp[i][j][a] = -1;
                }
            }
        }
        cout << getCount(arr1,arr2,m,n,k,dp) << endl;
    }

    return 0;
}
```

## 16-Ass : Adjacent Bit Counts

Send Feedback

For a string of n bits x1,x2,x3,...,Xn the adjacent bit count of the string (AdjBC(x)) is given by

X1*X2 + X2*X3 + X3*X4 + ... + Xn-1 * Xn

which counts the number of times a 1 bit is adjacent to another 1 bit. For example:

AdjBC(011101101) = 3

AdjBC(111101101) = 4

AdjBC(010101010) = 0

Write a program which takes as input integers n and k and returns the number of bit strings x of n bits (out of $2^n$) that satisfy AdjBC(x) = k. For example, for 5 bit strings, there are 6 ways of getting AdjBC(x) = 2:

11100, 01110, 00111, 10111, 11101, 11011

### Input Format:

First-line will contain T(number of the test case).
Each test case consists of a single line containing two space-separated integers N and K, a number of bits in the bit strings and desired adjacent bit count respectively.

### Output Format:

For each test case print the answer in a new line.
As answer can be very large print your answer modulo 10^9+7.

### Constraints:

1 <= T <= 10^5
1 <= N <= K <= 100

### Sample Input

10
5 2
20 8
30 17
40 24
50 37

60 52
70 59
80 73
90 84
100 90

**Sample Output**

6
63426
1861225
168212501
44874764
160916
22937308
99167
15476

23076518

```
#include<bits/stdc++.h>
using namespace std;
#define mod 1000000007

int main(){

    // write your code here
    int dp[101][101][2];
    memset(dp, 0, sizeof(dp));
    dp[1][0][0] = 1;
    dp[1][0][1] = 1;
    for(int i=2;i<=100;i++)
    {
        dp[i][i-1][0] = 0;
        dp[i][i-1][1] = 1;
        dp[i][0][0] = (dp[i-1][0][0] + dp[i-1][0][1]) % mod;
        dp[i][0][1] = dp[i-1][0][0];
    }
    for(int i=2;i<=100;i++)
    {
        for(int j=1;j<i;j++)
        {
            dp[i][j][0] = (dp[i-1][j][0] + dp[i-1][j][1]) % mod;
            dp[i][j][1] = (dp[i-1][j][0] + dp[i-1][j-1][1]) % mod;
        }
    }
    int t;
    cin>>t;
```

```
    while(t--)
    {
        int n,k;
        cin>>n>>k;
        if(n==0 && k==0)
            cout<<1<<endl;
        else
        {
            int ans = (dp[n][k][0] + dp[n][k][1]) % mod;
            cout<<ans<<endl;
        }
    }

    return 0;
}
```

# L13 : Bit Manipulation

## 1-Tut : Predict The Output

Send Feedback

```
#include <iostream>
using namespace std;

int main(){
    int x = 2;
    x = x << 1;
    cout << x;
}
```

### Answer

**Type here : 4**

**Correct Answer**

## 2-Tut : Predict The Output

Send Feedback

```
#include <iostream>
using namespace std;

int main(){
    int x = -2;
    x = x >> 1;
    cout << x;
}
```

### Answer

**Type here : -1**

**Correct Answer**

## 3-Tut : Predict The Output

Send Feedback

```
#include <iostream>
using namespace std;

int main(){
    if(~0 == 1) {
        cout << "yes";
    }
    else {
        cout << "no";
    }
}
```

## Options

This problem has only one correct answer

**yes**
**no**
**Compile time error**
**Undefined**

**Correct Answer : B (~0 online compiler shows it as -1)**

## 4-Tut : Predict The Output

Send Feedback

```
#include <iostream>
using namespace std;

int main(){
    int y = 0;
    if(1 | (y = 1)) {
        cout << "y is " << y;
    }
    else {
        cout << y;
    }
}
```

## Options

This problem has only one correct answer

**y is 0**
**y is 1**
**1**
**0**

**Correct Answer : B**

## 5-Tut : Predict The Output

Send Feedback

```
#include <iostream>
using namespace std;

int main(){
    int y = 1;
    if(y & (y = 2)) {
        cout << "true";
    }
    else {
        cout << "false";
    }
}
```

## Answer

**Type here : true**

Correct Answer

## 6-Tut : Turn Off The Bit

Send Feedback

Which bitwise operator is suitable for turning off a particular bit in a number?

## Options

This problem has only one correct answer

**&& operator**
**& operator**
**|| operator**
**| operator**

**Correct Answer : B**

## 7-Tut : Turn On The Bit

Send Feedback

Which bitwise operator is suitable for turning on a particular bit in a number?

## Options

This problem has only one correct answer

**&& operator**
**& operator**
**|| operator**
**| operator**

**Correct Answer : D**

## 8-Tut : Check ith bit

Send Feedback

Which bitwise operator is suitable for checking whether a particular bit is on or off?

**Note: Multiple options can be correct**

## Options

This problem may have one or more correct answers

**&& operator**
**& operator**
**|| operator**
**| operator**
**! operator**
**^ operator**

**The solution to this problem has been viewed**

## Solution Description

**If we want to find whether the ith bit is set or not for a given number N.**
**Then we can right shift given number(N) by (i - 1). Let's call this number b; b=(N>>(i - 1))**

**a) Using & operator: we take (b&1) if the result is 1, our ith bit was set else it was not set.**

**b) Using | operator:  we take (b|0) if the result is 1, our ith bit was set else it was not set.**

**c) Using ^ operator:  we take (b^0) if the result is 1, our ith bit was set else it was not set.**

## 9-Ass : Set ith Bit

Send Feedback

You are given two integers N and i. You need to make ith bit of binary representation of N to 1 and return the updated N.

Counting of bits start from 0 from right to left.

### Input Format:

First line of input will contain T(number of test cases), each test case follows as.
A single line containing two space-separated integers N and i.

### Output Format:

Updated N for each test case in new line.

### Constraints:

1 <= T <= 10^5
1 <= N <= 10^9
1 <= i <= 30

### Sample Input 1 :

1
4 1

### Sample Output 1 :

6

### Sample Input 2 :

1
4 4

### Sample Output 2 :

20

```
#include<bits/stdc++.h>
using namespace std;
int setibit(int N, int i){
    return (N | (1 << i));
}
int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int N,i; cin >> N >> i;
```

```
        cout << setibit(N,i) << endl;
    }
    return 0;
}
```

## 10-Ass : Unset ith Bit

Send Feedback

You are given two integers N and i. You need to make ith bit of binary representation of N to 0 and return the updated N.

Counting of bits start from 0 from right to left.

### Input Format:

First line of input contains T(number of test cases), each test case follows as.
Two integers N and i (separated by space)

### Output Format :

Updated N for each test case in new line.

### Constraints:

1 <= T <= 10^5
1 <= N <= 10^9
1 <= i < 30

### Sample Input 1 :

1
7 2

### Sample Output 1 :

3

### Sample Input 2 :

1
12 1

### Sample Output 2 :

12

```
#include<bits/stdc++.h>
using namespace std;
int unsetibit(int N, int i){

    return (N & ~(1 << i));

}
int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int N,i; cin >> N >> i;
```

```
        cout << unsetibit(N,i) << endl;
    }
    return 0;
}
```

## 11-Ass : Find First Set Bit

Send Feedback

You are given an integer N. You need to return an integer M, in which only one bit is set which at the position of a lowest set bit of N (from right to left).

**Input Format :**

The first line of input will contain T(number of the test case), each test case follows as.
The only line of each test case contains an integer N.

**Output Format:**

Integer M for each test case in a new line.

**Constraints:**

1 <= T <= 10^5
1 <= N <= 10^9

**Sample Input 1 :**

1
7

**Sample Output 1 :**1

**Sample Input 2 :**

1
12

**Sample Output 2 :** 4

```
#include<bits/stdc++.h>
using namespace std;
int firstsetbit(int N){
    int ans = 1;
    while( !(N&1) ){
        ans <<= 1;
        N >>= 1;
    }
    return ans;
}
int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int N; cin >> N;
        cout << firstsetbit(N) << endl;
    }
```

```
    return 0;
}
```

## 12-Ass : Turn Off First Set Bit

Send Feedback

You are given an integer Ni. You need to make rightmost set bit of binary representation of N to 0 and return the updated N.

Counting of bits start from 0 from right to left.

### Input Format :

The first line of input will contain T(number of test cases), each test case follows as.
A single integer N for each test case in a newline.

### Output Format :

Updated N for each test case in a newline.

### Constraints:

1 <= T <= 10^5
1 <= N <= 10^9

### Sample Input 1 :

1
4

### Sample Output 1 :

0

### Sample Input 2 :

1
12

### Sample Output 2 :

8

```
#include<bits/stdc++.h>
using namespace std;
int firstsetbitoff(int N){
    int N1 = N;
    int ans = 1;
    while( (N1&1) == 0 ){
        ans <<= 1;
        N1 >>= 1;
    }
    return N^ans;
}
int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
```

```
        int N; cin >> N;
        cout << firstsetbitoff(N) << endl;
    }
    return 0;
}
```

## 13-Ass : Clear All Bits From MSB

Send Feedback

You are given two integers N and i. You need to clear all bits from MSB to ith bit (start i from right to left) and return the updated N.

Counting of bits starts from 0 from right to left.

### Input Format :

First line of input will contain T(number of test cases), each test case follows as.
Line1: contain two space-separated integers N and i.

### Output Format :

Updated N for each test case in a newline.

### Constraints:

1 <= T <= 10^5
1 <= N <= 10^9
1 <= i <= 30

### Sample Input 1 :

1
15 2

### Sample Output 1 :

3

```
#include<bits/stdc++.h>
using namespace std;
int clearMSB(int N, int i){
    int mask = (1 << i)-1;
    return (N & mask);
}
int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int N,i; cin >> N >> i;
        cout << clearMSB(N,i) << endl;
    }
    return 0;
}
```

## 14-Ass : Odd Frequency

Send Feedback

You are given an array of size N with all elements with even frequency except one and you are supposed to find this element.

### Input Format:

The first line of input will contain T(number of test cases), each test case follows as.
Line 1: contain an integer N (number of elements in the array)
Line 2: contain N space-separated integers (elements of the array).

### Output Format:

For each test case print the element with the odd frequency in a new line.

### Constraints:

1 <= T <= 50
1 <= N <= 10^5
1 <= arr[i] <= 10^9

### Sample Input:

1
5
2 2 2 3 3

### Sample Output:

2

```
#include<bits/stdc++.h>
using namespace std;
int oddfreq(int a[],int N){
    int ans = 0;
    for(int i=0;i<N;i++){
        ans=ans^a[i];
    }
    return ans;
}
int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int N; cin >> N;
        int *arr = new int[N];
        for(int i = 0; i < N; i++){
            cin>>arr[i];
        }
        cout << oddfreq(arr,N) <<endl;
    }
    return 0; }
```

## 15-Ass : XOR of Natural Numbers

Send Feedback

You are given an integer N and asked to find the Xor of first N natural numbers.

### Input Format:

The first line of input will contain T(number of test cases), each test case follows as.
The only line of input contains an integer N.

### Output Format:

For each test case print the Xor of first N natural number in a new line.

### Constraints:

1 <= T <= 10^5
1 <= N <= 10^9

### Sample Input:

1
8

### Sample Output: 8

```
#include<bits/stdc++.h>
using namespace std;
int xorofFNNN(int N){
   int rem = N % 4;
    if(rem == 0){
        return N;
    }
    if(rem == 1){
        return 1;
    }
    if(rem == 2){
        return N+1;
    }
    if(rem == 3){
        return 0;
    }
}
int main(){

    // write your code here
    int T; cin >> T;
    while(T--){
        int N; cin >> N;
        cout << xorofFNNN(N) << endl;
    }
    return 0;
}
```

# L14 : DP and BitMasking

## 1-Tut : Candy

Send Feedback

Gary is a teacher at XYZ school. To reward his N students he bought a packet of N candies all with different flavours. But the problem is some students like certain flavour while some doesn't. Now Gary wants to know the number of ways he can distribute these N candies to his N students such that every student gets exactly one candy he likes.

### Input Format :

The first line of input will contain T(number of test cases), each test case follows as.
Line 1 : An integer N (1<= N <= 16) denoting number of students and candies.
Next N lines: N integers describing the preferences of one student. 1 at i'th (0 <= i < N) position denotes that this student likes i'th candy, 0 means he doesn't.
Assume input to be 0-indexed based.

### Output Format :

Return the number of ways Gary can distribute these N candies to his N students such that every student gets exactly one candy he likes for each test case in a new line.

### Sample Input:

```
1
3
1 1 1
1 1 1
1 1 1
```

### Sample Output: 6

### Explanation:

Since, all the students like all the candies, so, the candies can be distributed in the following 6 ways:

| | First Student | Second Student | Third Student |
|---|---|---|---|
| Way 1 | 1st Candy | 2nd Candy | 3rd Candy |
| Way 2 | 1st Candy | 3rd Candy | 2nd Candy |
| Way 3 | 2nd Candy | 1st Candy | 3rd Candy |
| Way 4 | 2nd Candy | 3rd Candy | 1st Candy |
| Way 5 | 3rd Candy | 1st Candy | 2nd Candy |
| Way 6 | 3rd Candy | 2nd Candy | 1st Candy |

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
int candies(int **like, int n, int person, int mask, int *dp)
{
    if (person >= n)
    {
        return 1;
    }

    if(dp[mask]!=-1)
    {
        return dp[mask];
    }
    int ans = 0;
    for (int i = 0; i < n; i++)
    {
        if (!(mask & (1 << i)) && like[person][i])
        {
            ans += candies(like, n, person + 1, mask | (1 << i), dp);
        }
    }
    dp[mask]=ans;
    return ans;
}


int solve(int **like,int n)
{
            int *dp = new int[1 << n];
    for (int i = 0; i < (1 << n); i++)
    {
        dp[i] = -1;
    }
    int ans= candies(like, n, 0, 0, dp);
    delete[]dp;
    return ans;
}

int32_t main(){

    // write your code here
    int t;
    cin>>t;
```

```
    while(t--){
        int n;
        cin>>n;
        int** like = new int*[n];

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


            like[i] = new int[n];
            }
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                cin>>like[i][j];
            }
        }

        cout<<solve(like,n)<<endl;
    }

    return 0;
}
```

## 2-Tut : Ghost Type

Send Feedback

Gengar has got an integer N. Now using his ghostly powers, he can create the permutation from 1 to N of this given number.

Since, he's a special kind of Poke'mon, so he thinks he deserves special permutations. He wants to find the total number of special permutations of length N, consisting of the integers from 1 to N.

A permutation is called special if it satisfies following condition:

If Ap & Aq == Ap, then p < q, where p and q are two distinct indices of permutation and A is the permutation itself. "&" denotes the bitwise and operation.
Help Gengar in finding the number of such permutations.

Input format:

The only line of input will consist of a single integer N denoting the length of the permutation.
Output format:

Output the total number of special permutations of length N.
Constraints:

**1 ≤ N ≤ 20**

SAMPLE INPUT **4**

SAMPLE OUTPUT **8**

## Explanation

All the special permutations of length 4 are:

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

4 2 1 3

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i, n) for (int i = 0; i < n; i++)
#define ll long long int
vector<int> submask[22];
bool vis[1 << 22];
ll dp[1 << 22];
ll rec(int mask, int n)
{
    if (mask == (1 << (n + 1)) - 2)
        return 1;
    if (vis[mask])
        return dp[mask];
    vis[mask] = 1;
    ll &ret = dp[mask];
    ret = 0;
    int x;
    for (int i = 1; i <= n; i++)
    {
        if (!(mask & (1 << i)))
        {
            bool ok = 1;
            rep(j, submask[i].size())
            {
                x = submask[i][j];
                if (!(mask & (1 << x)))
                    ok = 0;
            }
            if (ok)
            {
                ret += rec(mask | (1 << i), n);
```

```
            }
        }
    }
    return ret;
}
int main(){

    // write your code here
     int n;
    for (int i = 1; i <= 20; i++)
    {
        for (int j = i - 1; j >= 1; j--)
        {
            if ((i & j) == j)
                submask[i].push_back(j);
        }
    }
    cin >> n;
    cout << rec(0, n);
    return 0;
}
```

## 3-Tut : Dilemma

Send Feedback

Abhishek, a blind man recently bought N binary strings all of equal length .A binary string only contains '0's and '1's . The strings are numbered from 1 to N and all are distinct, but Abhishek can only differentiate between these strings by touching them. In one touch Abhishek can identify one character at a position of a string from the set. Find the minimum number of touches T Abhishek has to make so that he learn that all strings are different .

### Input Format:

First line of input will contain T(number of test cases), each test case follows as.
Line1: contain an integer N (number of strings)
Next N line contain binary strings.

### Output Format:

For each test case print the answer in newline.

### Constraints:

1 <= T <= 50
1 <= N <= 10
1 <= |S| <= 100

### Sample Input :

1
2
111010

100100

**Sample Output :**

2

**Explanation**

Abhishek touches 2nd bit from the start of both the binary string.
in the first touch (to string 1) he sees that the value is1.
in the second touch (to string 2)he sees that the value is 0.

so he concludes both strings are different in 2 touches.

```
#include<bits/stdc++.h>
using namespace std;
int dp[105][1 << 12];

int find_touches(int pos, int mask, vector<string> &v)
{
    if (!(mask & (mask - 1)) && mask)
    {
        return 0;
    }

    if (pos == -1 || mask == 0)
    {
        return 1000000;
    }

    if (dp[pos][mask])
    {
        return dp[pos][mask];
    }

    int newmask1 = 0, newmask2 = 0, touches = 0;

    for (int i = 0; i < v.size(); i++)
    {

        if ((mask >> i) & 1)
        {
            touches++;
            if (v[i][pos] == '0')
            {
                newmask1 |= (1 << i);
            }
            else
            {
```

```
36.                 newmask2 |= (1 << i);
37.             }
38.         }
39.     }
40.     return dp[pos][mask] = min(find_touches(pos - 1, newmask1, v) + find_touches(pos -
    1, newmask2, v) + touches, find_touches(pos - 1, mask, v));
41. }
42.
43. int solve(int n, vector<string> v)
44. {
45.     return find_touches(v[0].size() - 1, (1 << n) - 1, v);
46. }
47. int main(){
48.
49.     // write your code here
50.      int t;
51.     cin >> t;
52.     while (t--)
53.     {
54.         int n;
55.         cin >> n;
56.         vector<string> v;
57.         memset(dp, 0, sizeof(dp));
58.
59.         for (int i = 0; i < n; i++)
60.         {
61.             string s;
62.             cin >> s;
63.             v.push_back(s);
64.         }
65.         cout << solve(n, v) << endl;
66.     }
67.     return 0;
68. }
```

## 4-Ass : String Maker

Send Feedback

According to Ancient Ninjas , string making is an art form . There are various methods of string making , one of them uses previously generated strings to make the new one . Suppose you have two strings A and B , to generate a new string C , you can pick a subsequence of characters from A and a subsequence of characters from B and then merge these two subsequences to form the new string. Though while merging the two subsequences you can not change the relative order of individual subsequences. What this means is - suppose there two characters Ai and Aj in the subsequence chosen

from A , where i < j , then after merging if i acquires position k and j acquires p then k<p should be true and the same apply for subsequence from C.

Given string A , B , C can you count the number of ways to form string C from the two strings A and B by the method described above. Two ways are different if any of the chosen subsequence is different .

As the answer could be large so return it after modulo 10^9+7 .

### Input Format :

First line will contain T(number of test cases), each test case consists of three lines.
Line 1 : String A
Line 2 : String B
Line 3 : String C

### Output Format :

The number of ways to form string C for each test case in new line.

### Constraints :

1 <= T <= 500
1 <= |A| , |B|, |C| <=50

### Sample Input :

1
abc
abc
abc

### Sample Output :

8

```
#include<bits/stdc++.h>
using namespace std;
#define m 1000000007
typedef long long int ll;
int helper(string a, string b, string c, ll ***dp)
{
    if(c.length()==0)
    {
        return 1;
    }
    if(a.length()<=0&&b.length()<=0)
    {
        return 0;
    }

    if(dp[a.length()][b.length()][c.length()]>-1)
    {
        return dp[a.length()][b.length()][c.length()]%m;
    }
    ll ans=0;
```

```
    for(ll i=0; i<a.length(); i++)
    {
        if(a[i]==c[0])
        {
            ans+=helper(a.substr(i+1), b, c.substr(1), dp)%m;
        }
    }
    for(ll i=0; i<b.length(); i++)
    {
        if(b[i]==c[0])
        {
            ans+=helper(a, b.substr(i+1), c.substr(1), dp)%m;
        }
    }
    dp[a.length()][b.length()][c.length()]=ans%m;
    return ans%m;
}
int solve(string a, string b, string c)
{
    ll ***dp=new ll **[51];
    for(ll i=0; i<51; i++)
    {
        dp[i]=new ll *[51];
        for(ll j=0; j<51; j++)
        {
            dp[i][j]=new ll [51];
            for(ll k=0; k<51; k++)
            {
                dp[i][j][k]=-1;
            }
        }
    }
    ll ans= helper(a, b, c, dp)%m;
    for(ll i=0; i<51; i++)
    {
        for(ll j=0; j<51; j++)
        {
            delete[]dp[i][j];
        }
    }
    return ans;
}

int main(){
```

```
    // write your code here
    int n;
    cin>>n;
    while(n--){
        string a,b,c;
        cin>>a>>b>>c;
        cout<<solve(a,b,c)<<endl;
    }
    return 0;
}
```

## 5-Ass : Counting Strings

Send Feedback

Given a string 's' consisting of upper case alphabets, i.e. from 'A' to 'Z'. Your task is to find how many strings 't' with length equal to that of 's', also consisting of upper case alphabets are there satisfying the following conditions:

-> String 't' is lexicographical larger than string 's'.
-> When you write both 's' and 't' in the reverse order, 't' is still lexicographical larger than 's'.
Find out number of such strings 't'. As the answer could be very large, take modulo 10^9 + 7.

### Input Format:

First line will contain T(number of test cases).
Each test case consists of a single line containing the string s.

### Output Format:

For each test case output the number of strings (t) %(10^9 + 7) in new line.

### Constraints:

1 <= T <= 50
1 <= |S| <= 10^5

### Sample Input:

```
2
A
XKS
```

### Sample output:

```
25

523
```

```
#include<bits/stdc++.h>
using namespace std;
int mod = 1e9 + 7;

int countStrings(string &s){

                int n = s.size();
```

```
8.                      long long int req = 0, dp[n + 1], arr[n + 1], ans = 0;
9.
10.                     for (int i = 0; i < n; i++)
11.                     {
12.                             arr[i] = ('Z' - s[i]);
13.                     }
14.
15.                     dp[n - 1] = arr[n - 1];
16.
17.                     for (int i = n - 2; i >= 0; i--)
18.                     {
19.
20.                             req = (arr[i + 1] + (26ll * req) % mod) % mod;
21.
22.                             dp[i] = (arr[i] + (arr[i] * req) % mod) % mod;
23.                     }
24.
25.                     for (int i = 0; i < n; i++)
26.                     {
27.
28.                             ans = (ans + dp[i]) % mod;
29.                     }
30.
31.                     return ans;
32.             }
33.
34. int main(){
35.
36.     // write your code here
37.     int t;
38.                     cin >> t;
39.                     while (t--)
40.                     {
41.                             string s;
42.                             cin >> s;
43.                             cout << countStrings(s) << endl;
44.                     }
45.     return 0;
46. }
```

## 6-Ass : Number of APs

Send Feedback

Given an array of n positive integers. The task is to count the number of Arithmetic Progression subsequences in the array. As the answer could be very large, output it modulo 10^9 + 7.

Note: Empty sequence or single element sequence is Arithmetic Progression.

### Input Format:

First Line: N (the size of the array)
Second Line: Elements of the array separated by spaces.

### Output:

Print total number of subsequences

### Input Constraints:

1 <= arr[i] <= 1000
1 <= sizeof(arr) <= 1000

### Sample Input 1 :

3
1 2 3

### Sample output:

8

### Sample Output Explanation:

Total subsequence are: {}, { 1 }, { 2 }, { 3 }, { 1, 2 }, { 2, 3 }, { 1, 3 }, { 1, 2, 3 }

### Sample Input 2:

7
1 2 3 4 5 9 10

### Sample Output:

37

```
#include<bits/stdc++.h>
using namespace std;
int mod = 1e9 + 7;
int numofAP(int arr[], int n){

                int minElem = *min_element(arr, arr + n);
                int maxElem = *max_element(arr, arr + n);

                int totalAPs = n + 1;
                int sum[1001];
        //CODE BY KAMAL CHAUHAN
                for (int d = (minElem - maxElem); d <= (maxElem - minElem); d++)
                {
                        memset(sum, 0, sizeof(sum));

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

```
                                        int a = 1;
                                        if (arr[i] - d >= 1 && arr[i] - d <= 1000)

                                                a = (a + sum[arr[i] - d]) % mod;

                                        totalAPs = ((totalAPs + a - 1) % mod + mod) % mod;
                                        sum[arr[i]] = (sum[arr[i]] + a) % mod;
                                }
                        }
                        return totalAPs;
}
int main(){

    // write your code here
    int n;
                        cin >> n;
                        int arr[n];
                        for (int i = 0; i < n; i++)
                        {
                                cin >> arr[i];
                        }
                        cout << numofAP(arr, n);
    return 0;
}
```

# L15 : Number Theory 1

## 1-Tut : Find Prime Numbers From 1 to N

Send Feedback

Given a number N, find number of primes in the range [1,N].

**Input Format:**

The only line of input consists of a number N

**Output Format:**

Print the number of primes in the range [1,N].

**Constraints:**

1 ≤ N ≤ 10^6

**Sample Input :**

3

**Sample Output -**

2

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1000005;
bool sieve[N];

int main()
{
    for(int i = 0 ; i < N ; ++i) {
        sieve[i] = true;
    }
    sieve[0] = sieve[1] = false;

    for(int i = 2 ; i*i <= N ; ++i) {
        if(sieve[i]) {
            for(int j = i*i ; j < N ; j += i) {
                sieve[j] = false;
            }
        }
    }

    int n; cin >> n;

    int count = 0;
    for(int i = 0 ; i <= n ; ++i) {
        if(sieve[i]) ++count;
    }
```

```
27.
28.     cout << count << '\n';
29.
30.     return 0;
31. }
```

## 2-Tut : GCD

Send Feedback

Calculate and return GCD of two given numbers x and y. Numbers are within range of Integer.

**Input format :**

First line of Input will contain T(number of test cases), each test case follows as.
x and y (separated by space)

**Output format :**

Print GCD of x and y for each test case in newline

**Constraints:**

1 <= T <= 10^5
1 <= x, y <= 10^9

**Sample Input 1:**

1
20 5

**Sample Output 1:**

5

**Sample Input 2:**

1
96 14

**Sample Output 2:**

2

```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.  int gcd(int x,int y){
4.      if((x == 0) || y == 0 ){
5.          return x^y;
6.      }
7.      if(x > y){
8.          return gcd(y,x%y);
9.      }
10.
11.     return gcd(x,y%x);
12. }
13. int main(){
14.
15.     // write your code here
```

```
    int t;cin>>t;
    while(t--){
        int x,y; cin>>x>>y;
        cout<<gcd(x,y)<<endl;
    }
    return 0;
}
```

## 3-Ass : Super Prime

Send Feedback

A number is called super-prime if it has exactly two distinct prime divisors

Example 10 , 6

You are supposed to find the count of super-prime numbers between 1 and N (inclusive).

**Input Format:**

Contain an integer N

**Output Format:**

Print the number of super prime between [1, N]

**Constraints:**

1 <= N <= 10^6

**Sample Input 1:**

10

**Sample Output 1:**

2

**Sample Input 2:**

25

**Sample Output 2:**

10

**Explanation:**

The super-primes are: 6, 10, 12, 14, 15, 18, 20, 21, 22, 24.

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1000005;
long long sieve[N];

int main()
{
    for(int i = 0 ; i < N ; ++i) {
        sieve[i] = 0;
    }

    for(int i = 2 ; i < N ; ++i) {
        if(!sieve[i]) {
```

```
            for(int j = 2*i ; j < N ; j += i) {
                ++sieve[j];
            }
        }
    }

    int n; cin >> n;

    int count = 0;
    for(int i = 0 ; i <= n ; ++i) {
        if(sieve[i] == 2) ++count;
    }

    cout << count << '\n';

    return 0;
}
```

## 4-Ass : Ninja and Flowers

Send Feedback

Ninja wants to get N flowers and he will pay i + 1 amount of money for the Ith flower, example (if n=3 he will pay {2,3,4})

Now he wants to pack these N flowers in boxes of different colours. With one condition if the cost of a flower is a prime divisor of another flower they needed to be of a different colour.

As we know that ninja is a little money minded he wants to minimize the number of different colours of boxes that he needs.

### Input Format:

The only line of input will contain an integer N (number of flowers).

### Output Format:

In first-line print K, the minimum number of different colour boxes that are needed to pack the flowers. Next line contains K space-separated integers in sorted order denoting the counts of the different coloured boxes.

### Constraints:

1 <= N <= 2*10^5

### Sample Input:

4

### Sample Output:

2

1 3

```
#include<bits/stdc++.h>
using namespace std;
```

```cpp
#define int long long
#define double long double

const int N = (int) 1e6+5;
vector<bool> sieve;

int32_t main()
{
	ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

	sieve = vector<bool>(N, true);
	for(int i = 2 ; i*i <= N ; ++i) {
		if(sieve[i]) {
			for(int j = i*i ; j < N ; j += i) {
				sieve[j] = false;
			}
		}
	}

	int n; cin >> n;

	if(n <= 1) {
		cout << 1 << '\n';
	}
	else {
		cout << 2 << '\n';
	}

	int cp = 0, cnp = 0;
	for(int i = 2 ; i <= n+1 ; ++i) {
		if(sieve[i]) ++cp;
		else ++cnp;
	}

	cout << min(cp, cnp) << ' ' << max(cp, cnp) << '\n';

	return 0;
}
```

## 5-Ass : Special Prime

Send Feedback

Special Prime is a prime number that can be written as the sum of two neighbouring primes and 1.

You are given an integer N and you are supposed to find the number special prime in the range: [1, N].

Example of special prime 19 = 7 + 11 + 1

Neighbouring primes are prime number such that there is no other prime number between them.

**Input Format:**

An integer N.

**Output Format:**

Print the number of special primes

**Constraints:**

1 <= N <= 2*10^5

**Sample Input:**

27

**Sample Output:**

2

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

const int N = (int) 1e6+5;
vector<bool> sieve;

int32_t main()
{
	ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

	int n; cin >> n;

	sieve = vector<bool>(N, true);
	for(int i = 2 ; i*i <= N ; ++i) {
		if(sieve[i]) {
			for(int j = i*i ; j < N ; j += i) {
				sieve[j] = false;
			}
		}
	}

	vector<int> primes;
	for(int i = 2 ; i < N ; ++i) {
		if(sieve[i]) primes.emplace_back(i);
	}

	unordered_set<int> set;
	for(int i = 1 ; i < primes.size() ; ++i) {
		set.insert(primes[i-1] + primes[i]);
```

```
32.             }
33.
34.             int count = 0;
35.             for(int i = 2 ; i <= n ; ++i) {
36.                     if(sieve[i] && set.count(i-1)) ++count;
37.             }
38.
39.             cout << count << '\n';
40.
41.             return 0;
42. }
```

# L16 : Number Theory 2

## 1-Tut : Sachin And Varun Problem

Send Feedback

Varun and you are talking about a movie that you have recently watched while Sachin is busy teaching Number Theory. Sadly, Sachin caught Varun talking to you and asked him to answer his question in order to save himself from punishment. The question says:

Given two weights of a and b units, in how many different ways you can achieve a weight of d units using only the given weights? Any of the given weights can be used any number of times (including 0 number of times).

Since Varun is not able to answer the question, he asked you to help him otherwise he will get punished.

Note: Two ways are considered different if either the number of times a is used or a number of times b is used is different in the two ways.

### Input Format:

The first line of input consists of an integer
T denoting the number of test cases.
Each test case consists of only one line containing three space-separated integers a, b and d.

### Output Format:

For each test case, print the answer in a separate line.

### Constraints:

1 ≤ T ≤ 10 ^ 5

1 ≤ a < b ≤ 10 ^ 9

0 ≤ d ≤ 10 ^ 9

### Sample Input 1

```
4
2 3 7
4 10 6
6 14 0
2 3 6
```

### Sample Output 1

```
1
0
1
2
```

### Explanation

Test case 1: 7 can only be achieved by using 2 two times and 3 one time.

Test case 2: 6 can't be achieved by using 4 and 10. So, 0 ways are there.

```
1. #include<bits/stdc++.h>
2. using namespace std;
```

```
3.  typedef long long int ll;
4.  class triplet
5.  {
6.  public:
7.      ll x, y, gcd;
8.  };
9.  triplet extended_euclid(ll a, ll b)
10. {
11.     if (b == 0)
12.     {
13.         triplet ans;
14.         ans.x = 1;
15.         ans.y = b;
16.         ans.gcd = a;
17.         return ans;
18.     }
19.     triplet small_ans = extended_euclid(b, a % b);
20.     triplet ans;
21.     ans.gcd = small_ans.gcd;
22.     ans.x = small_ans.y;
23.     ans.y = small_ans.x - small_ans.y * (a / b);
24.     return ans;
25. }
26. ll Multiplicative_Modulo_Inverse(ll a, ll m)
27. {
28.     ll ans=extended_euclid(a, m).x;
29.     return (ans%m+m)%m;
30. }
31. ll count_ways(ll a, int b, ll d)
32. {
33.     ll y1=((d%a)*Multiplicative_Modulo_Inverse(b, a))%a;
34.     ll first_value=d/b;
35.     if((d/b)<y1)
36.     {
37.         return 0;
38.     }
39.     ll n=(first_value-y1)/a;
40.     return n+1;
41. }
42. int main()
43. {
44.     ll t;
45.     cin>>t;
46.     while(t--)
```

```
    {
        ll a, b, d;
        cin>>a>>b>>d;
        ll g=__gcd(a, b);
        if(d%g!=0)
        {
            cout<<0<<endl;
            continue;
        }
        a/=g;
        b/=g;
        d/=g;
        cout<<count_ways(a, b, d)<<endl;
    }
}
```

## 2-Tut : Advanced GCD

Send Feedback

Varun explained its friend Sanchit the algorithm of Euclides to calculate the GCD of two numbers. Then Sanchit implements the algorithm

```
int gcd(int a, int b)
{
   if (b==0)
   return a;
   else
   return gcd(b,a%b);
}
```

and challenges to Varun to calculate gcd of two integers, one is a little integer and another integer can have 10^4 digits.

Your task is to help Varun an efficient code for the challenge of Sanchit.

### Input Format

The first line of input will contain T(number of the test case), each test case follows as.
Each test case consists of two number A and B.

### Output Format:

Print for each pair (A,B) in the input one integer representing the GCD of A and B.

### Constraints:

1 <= T <= 100
1 <= A <= 4*10^5
1 <= |B| <= 10^4
where |B| is the length of the integer B

### Sample Input:

2
2 6
10 11

**Sample Output:**

```
2

1
```

```
#include<bits/stdc++.h>
using namespace std;

int gcdmain(int a ,int b){
    if(b==0){
        return a;
    }
    return gcdmain(b,a%b);
}

int mod(string num, int a)
{

    int res = 0;

    for (int i = 0; i < num.length(); i++)
        res = (res * 10 + (int)num[i] - '0') % a;

    return res;
}
int gcd(int a,string b)
{

 int d=mod(b,a);
 int maxi=max(a,d);
 int mini=min(a,d);
 return gcdmain(maxi,mini);
}
int main(){

  int t;
    cin>>t;
    while(t--){
        int a;
        cin>>a;
        string b;
        cin>>b;
        cout<<gcd(a,b)<<endl;
```

```
    }
    return 0;
}
```

## 3-Tut : Divisors Of Factorial

Send Feedback

Given a number, find the total number of divisors of the factorial of the number.

Since the answer can be very large, print answer modulo 10^9+7.

### Input Format:

The first line contains T, number of test cases.
T lines follow each containing the number N.

### Output Format:

Print T lines of output each containing the answer.

### Constraints

1 <= T <= 500
0 <= N <= 50000

### Sample Input:

```
3
2
3
4
```

### Sample Output:

```
2
4

8
```

```
#include<bits/stdc++.h>
using namespace std;
#define m 1000000007
void fact_divisor(int n){
    bool *prime=new bool[n+1];
    for(long long i=0;i<n+1;i++){
        prime[i]=true;
    }

    prime[0]=false;
    prime[1]=false;
    for(long long i=2;i<=sqrt(n);i++){
        if(prime[i]){
            for(long long j=i;j*i<=n;j++){
                prime[j*i]=false;
            }
        }
```

```
    }


    long long divisor=1;
    for(long long i=0;i<=n;i++){
        if(prime[i]){
            long long sum=0;
            for(long long j=1;pow(i,j)<=n;j++){
                sum+=n/pow(i,j);
            }
            divisor=(divisor%m*(sum+1)%m)%m;
        }
    }
    cout<<divisor<<endl;

}
int main(){

    // write your code here
     long long t;
    cin>>t;
    while(t--){
        long long n;
        cin>>n;
        fact_divisor(n);
    }
    return 0;
}
```

## 4-Ass : Find The Cube Free Number

Send Feedback

A cube free number is a number who's none of the divisor is a cube number (A cube number is a cube of a integer like 8 (2 * 2 * 2) , 27 (3 * 3 * 3) ). So cube free numbers are 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, 18 etc (we will consider 1 as cube free). 8, 16, 24, 27, 32 etc are not cube free number. So the position of 1 among the cube free numbers is 1, position of 2 is 2, 3 is 3 and position of 10 is 9. Given a positive number you have to say if its a cube free number and if yes then tell its position among cube free numbers.

Note: we will consider 1 as the 1st cube free number

### Input Format:

First line of the test case will be the number of test case T
Each test case contain an integer N

**Output Format:**

For each test case print the position of N in cube free numbers and if its not a cube free number print "Not Cube Free" in a newline.

**Constraints:**

1 <= T <= 10^5

1 <= N <= 10^6

**Sample Input:**

10
1
2
3
4
5
6
7
8
9
10

**Sample Output:**

1
2
3
4
5
6
7
Not Cube Free
8

9

```
#include<bits/stdc++.h>
using namespace std;
void cubic(int arr[]){
    for(int i=0;i<1000001;i++){
        arr[i]=0;
    }
    arr[1]=1;
    for(int i=2;i<=100;i++){
        int cube=i*i*i;
        for(int j=1;cube*j<1000001;j++){
            arr[cube*j]=-1;

        }
    }


```

```
    int k=1;
    for(int i=1;i<1000001;i++){
        if(arr[i]!=-1){
            arr[i]=k;
            k++;
        }
    }
}
int main(){

    // write your code here
    int n;
    cin>>n;
    int arr[1000001];
    cubic(arr);
    for(int i=1;i<=n;i++){
        int val;
        cin>>val;
        if(arr[val]!=-1){
            cout<<arr[val]<<endl;
        }
        else{
            cout<<"Not Cube Free"<<endl;
        }
    }

    return 0;
}
```

## 5-Ass : Number Of Factors

Send Feedback

A number is called n-factorful if it has exactly n distinct prime factors. Given positive integers a, b, and n, your task is to find the number of integers between a and b, inclusive, that are n-factorful. We consider 1 to be 0-factorful.

### Input Format:

First line of input will contain T(number of test cases), each test case follows as.
Each test cases consists of a single line containing integers a, b, and n as described above.

### Output Format:

Output for each test case one line containing the number of n-factorful integers in [a, b].

### Constraints:

1 <= T <= 10000
1 ≤ a ≤ b ≤ 10^6
0 <= b - a <= 100

$0 \le n \le 10$

**Sample Input**

```
4
1 3 1
1 10 2
1 10 3
1 100 3
```

**Sample Output**

```
2
2
0

8
```

```
#include<bits/stdc++.h>
using namespace std;
void makeSieve(int *isprime, int n)
{
    for (int i = 0; i < n + 1; i++)
    {
        isprime[i] = 0;
    }
    for (int i = 2; i <= n; i++)
    {
        if (isprime[i] == 0)
            for (int j = 1; j * i <= n; j++)
            {
                isprime[j * i] += 1;
            }
    }
}
int main(){

    // write your code here
     int t;
    cin >> t;
    int *arr = new int[1000001];
    int **table=new int* [11];
    for(int i=0; i<11; i++)
    {
        table[i]=new int[10000001];
    }
    makeSieve(arr, 1000001);
    for(int i=0; i<11; i++)
    {
```

```
32.         table[i][0]=0;
33.         table[i][1]=0;
34.         for(int j=2; j<1000001; j++)
35.         {
36.             if(i==arr[j])
37.             {
38.                 table[i][j]=table[i][j-1]+1;
39.             }
40.             else
41.             {
42.                 table[i][j]=table[i][j-1];
43.             }
44.         }
45.     }
46.     for (int test_case = 1; test_case <= t; test_case++)
47.     {
48.         int a, b, n;
49.         cin>>a>>b>>n;
50.         if(a==1&&n==0)
51.         {
52.             cout<<1<<endl;
53.             continue;
54.         }
55.         cout<<table[n][b]-table[n][a-1]<<endl;
56.     }
57.     return 0;
58. }
```

## 6-Ass : Find the good sets!

Send Feedback

You are given array a consisting of n distinct integers. A set s of numbers is called good if you can rearrange the elements in such a way that each element divides the next element in the order, i.e. 'si' divides 'si + 1', such that i < |s|, where |s| denotes size of set |s|.

Find out number of distinct good sets that you can create using the values of the array. You can use one value only once in a particular set; ie. a set cannot have duplicate values. Two sets are said to be different from each other if there exists an element in one set, which does not exist in the other.

As the answer could be large, print your answer modulo 10^9 + 7.

### Input Format:

First line of the input contains an integer T denoting the number of test cases. T test cases follow.
First line of each test case contains an integer n denoting number of elements in array a.
Next line contains n space separated integers denoting the elements of array a.

### Output Format:

For each test case, output a single line containing the corresponding answer.

**Constraints**

1 ≤ T ≤ 10
1 ≤ n ≤ 10^5
1 ≤ ai ≤ 10^5
All the elements of array a are distinct.

**Input**

2
2
1 2
3
6 2 3

**Output:**

3
5

**Explanation**

Test case 1. There are three sets which are good {1}, {2}, {1, 2}.

Test case 2. There are five sets which are good {6}, {2}, {3}, {6 2}, {6, 3}.

```
#include<bits/stdc++.h>
using namespace std;
#define m 1000000007
#define ll long long int
ll size_of_sieve = 750001;
void Seive(ll *arr, ll n)
{
    ll *sieve = new ll[size_of_sieve];
    for (ll i = 0; i < size_of_sieve; i++)
    {
        sieve[i] = 0;
    }
    for (ll i = 0; i < n; i++)
    {
        sieve[arr[i]] = 1;
    }
    for (ll i = 1; i < size_of_sieve; i++)
    {
        ll current_element = i;
        if (sieve[current_element] != 0)
            for (ll j = 2; j * current_element < size_of_sieve; j++)
            {
                if (sieve[j * current_element] != 0)
                {
                    sieve[j * current_element] = (sieve[j * current_element] % m + sieve[i] % m)
% m;
```

```
            }
        }
    }
    ll total_sum = 0;
    for (ll i = 0; i < size_of_sieve; i++)
    {
        total_sum = (total_sum % m + sieve[i] % m) % m;
    }
    cout << total_sum << endl;
    delete[] sieve;
}
int main(){

    // write your code here
    ll t;
    cin >> t;
    ll *arr = new ll[750001];
    while (t--)
    {
        ll n;
        cin >> n;
        for (ll i = 0; i < n; i++)
        {
            cin >> arr[i];
        }
        Seive(arr, n);
    }
    delete[] arr;
    return 0;
}
```

## 7-Ass : Card Game

Send Feedback

Vova again tries to play some computer card game.

The rules of deck creation in this game are simple. Vova is given an existing deck of n cards and a magic number k. The order of the cards in the deck is fixed. Each card has a number written on it; number ai is written on the i-th card in the deck.

After receiving the deck and the magic number, Vova removes x (possibly x = 0) cards from the top of the deck, y (possibly y = 0) cards from the bottom of the deck, and the rest of the deck is his new deck (Vova has to leave at least one card in the deck after removing cards). So Vova's new deck actually contains cards x + 1, x + 2, ... n - y - 1, n - y from the original deck.

Vova's new deck is considered valid iff the product of all numbers written on the cards in his new deck is divisible by k. So Vova received a deck (possibly not a valid one) and a number k, and now he wonders,

how many ways are there to choose x and y so the deck he will get after removing x cards from the top and y cards from the bottom is valid?

### Input

The first line contains two integers n and k (1 ≤ n ≤ 100 000, 1 ≤ k ≤ 10^9).
The second line contains n integers a1, a2, ..., an (1 ≤ ai ≤ 10^9) — the numbers written on the cards.

### Output

Print the number of ways to choose x and y so the resulting deck is valid.

### Sample Input 1

3 4
6 2 8

### Sample Output 1

4

### Sample Input 2

3 6
9 1 14

### Sample Output 2

1

```
#include<iostream>
using namespace std;
#include<utility>
#include<vector>
#define ll long long int
#define PII pair<int, int>
#define f first
#define s second
#define mk make_pair
#define pb push_back
int main(){
        // Write your code here
 int n, i, j;
ll k, temp_for_k, ans = 0;
cin >> n >> k;
temp_for_k = k;
vector<ll> a(n);
for (i = 0; i < n; i++)
{
    cin >> a[i];
}
vector<PII> vp; //store prime factors of k with respective powers
for (i = 2; i * i <= temp_for_k; i++)
{
    if (temp_for_k % i == 0)
```

```
        {
            int count = 0;
            while (temp_for_k % i == 0)
            {
                temp_for_k /= i;
                count++;
            }
            vp.push_back(make_pair(i, count));
        }
    }
    if (temp_for_k != 1)
    {
        vp.push_back(make_pair(temp_for_k, 1));
    }
    //vq contains all prime factors of k with their respective powers 0
    vector<PII> vq = vp;
    for (i = 0; i < vq.size(); i++)
    {
        vq[i].s = 0;
    }
    //j is back pointer and i is front pointer
    //We are checking if j..i is divisible by k by adding prime factors of a[i] to vq
    //If it is divisible, then j...z for all z > i is also divisible by k
    //So we add n-i to answer
    //Then we remove j's contribution by subtracting it's prime factors from vq
    j = 0;
    for (i = 0; i < n; i++)
    {
        //add factors of a[i]
        for (int z = 0; z < vp.size(); z++)
        {
            if (a[i] % vp[z].f == 0)
            {
                temp_for_k = a[i];
                int cn = 0;
                while (temp_for_k % vp[z].f == 0)
                {
                    temp_for_k /= vp[z].f;
                    cn++;
                }
                vq[z].s += cn;
            }
        }
        while (j <= i)
```

```
70.         {
71.             int z;
72.             //check if divisible by k from j to i
73.             for (z = 0; z < vp.size(); z++)
74.             {
75.                 if (vp[z].s > vq[z].s)
76.                     break;
77.             }
78.             if (z != vp.size())
79.                 break;
80.             //if divisible update ans
81.             ans += n - i;
82.             //remove factors of a[j] from vq
83.             for (int z = 0; z < vp.size(); z++)
84.             {
85.                 if (a[j] % vp[z].f == 0)
86.                 {
87.                     temp_for_k = a[j];
88.                     int cn = 0;
89.                     while (temp_for_k % vp[z].f == 0)
90.                     {
91.                         temp_for_k /= vp[z].f;
92.                         cn++;
93.                     }
94.                     vq[z].s -= cn;
95.                 }
96.             }
97.             j++;
98.         }
99.     }
100.        cout << ans;
101.         return 0;
102.     }
```

## 8-Ass : Strange order

Send Feedback

Given an integer n . Initially you have permutation p of size n : p[i] = i . To build new array a from p following steps are done while permutation p is not empty:

Choose maximum existing element in p and define it as x ; Choose all such y in p that gcd ( x , y ) ≠ 1 ; Add all chosen numbers into array a in decreasing order and remove them from permutation. Your task is to find a after p became empty.

Note: gcd ( a , b ) is the greatest common divisor of integers a and b .

**Input format**

Input contains single integer n — the size of permutation p p.

**Output format**

Print n integers — array a .

**Constraints:**

1 <= N <= 10^5

**Sample Input:**

5

**Sample Output:**

5 4 2 3 1

**Explanation**

It's needed 4 iterations to create array a:

Add 5

Add 4 and 2

Add 3

Add 1

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
#define MaxSize 2000001
int main()
{
    int *sieve = new int[MaxSize];
    for (int i = 0; i <= MaxSize; i++)
    {
        sieve[i] = i;
    }
    for (int i = 2; i * i <= MaxSize; i++)
    {
        for (int j = i * i; j <= MaxSize; j += i)
        {
            if (sieve[j] > i)
            {
                sieve[j] = i;
            }
        }
    }
    int n, k = 0;
    cin >> n;
    int *finalans = new int[n];
    bool *marked = new bool[n + 1];
    for (int i = 0; i <= n; i++)
```

```
    {
        marked[i] = false;
    }
    for (int i = n; i >= 1; i--)
    {
        if (!marked[i])
        {
            int lpd = sieve[i];
            int x = i;
            vector<int> v;
            marked[i] = true;
            v.push_back(i);
            while (x != 1)
            {
                for (int j = i - lpd; j >= 1; j = j - lpd)
                {
                    if (!marked[j])
                    {
                        marked[j] = true;
                        v.push_back(j);
                    }
                }
                while (x % lpd == 0)
                {
                    x = x / lpd;
                }
                lpd = sieve[x];
            }
            sort(v.begin(), v.end(), greater<int>());
            for (int i = 0; i < v.size(); i++)
            {
                finalans[k] = v[i];
                k++;
            }
        }
    }
    finalans[n - 1] = 1;
    for (int i = 0; i < n; i++)
    {
        cout << finalans[i] << " ";
    }
    cout << endl;
}
```

# L17 : Number Theory 3

## 1-Tut : Totient Function

Send Feedback

You are given an integer N and are supposed to find the value of Euler toient function for N phi(N)

**Input Format:**

First line of input will contain T(number of test case), each test case follows as.
An integer N in new line.

**Output Format:**

For each test case print the answer in new line

**Constraints:**

1 <= T <= 100
1 <= N <= 10^9

**Sample Input:**

2
20
21

**Sample Output:**

8

12

```
#include<bits/stdc++.h>
using namespace std;

int main(){

    int t;
    cin>>t;
    while(t--){
        int n;
        cin>>n;
        int temp=n;
        int ans=n;
        for( int i=2 ; i*i<=n ; i++)
        {
            if((temp%i)==0)
            {
                while((temp%i)==0)
                temp/=i;

                ans-=ans/i;
             }
```

```
        }
        if(temp>1)
        {
            ans-=ans/temp;
        }
        cout<<ans<<endl;

    }
}
```

## 2-Tut : Sum of LCM

Send Feedback

Given n, calculate and print the sum :

LCM(1,n) + LCM(2,n) + .. + LCM(n,n)
where LCM(i,n) denotes the Least Common Multiple of the integers i and n.

**Input Format :**

First line of input will contain T(number of test case), each test case follows as.
An Integer N

**Output Format :**

Required sum for each test case in newline.

**Constraints :**

1 <= T <= 10^4
1 <= n <= 10^5

**Sample Input 1 :**

1
5

**Sample Output 1 :**

55

**Sample Input 2 :**

1
2

**Sample Output 2 :**

4

```
#include<bits/stdc++.h>
using namespace std;
unsigned long long int phi(unsigned long long int n)
{
        unsigned long long int result = n;
        for (unsigned long long int i = 2; i * i <= n; i++)
        {
                if (n % i == 0)
```

```
                {
                while (n % i == 0)
                        n /= i;

                result = (result*(i-1))/i;
                }
        }
        if (n > 1)
                result = (result*(n-1))/n;
        return result;
}

int main(){


    int t;
    cin>>t;
    while(t--)
    {
        long long n;
        cin>>n;
        vector<long long> divisors;
         for (long long i=2; i<=sqrt(n); i++)
           {
           if (n%i == 0)
           {

                   if (n/i == i)
              {
                 divisors.push_back(i);
              }
                                    else
              {
               divisors.push_back(i);
                 divisors.push_back(n/i);
              }
        }
    }
        unsigned long long int sum=0;
        for(int i=0;i<divisors.size();i++)
        {
            sum=(sum+(phi(n/divisors[i])*((n*n)/divisors[i])));
        }
        sum=(sum+(phi(n)*(n*n)));
```

```
        sum=(sum+2*n);
        sum=sum/2;
        cout<<sum<<endl;

    }
    return 0;
}
```

## 3-Tut : Segmented Sieve Problem

Send Feedback

In this problem you have to print all primes from given interval.

### Input Format:

First line of input will contain T(number of test cases), each test case follows as.
On each line are written two integers L and U separated by a blank. L - lower bound of interval, U - upper bound of interval.

### Output Format:

For each test case output must contain all primes from interval [L; U] in increasing order.

### Constraints:

1 <= T <= 100
1 <= L <= U <= 10^9
0 <= U - L <= 10^5

### Sample Input:

```
2
2 10
3 7
```

### Sample Output:

```
2 3 5 7

3 5 7
```

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    long long n=1000000000;
    long long size=sqrt(n);
    bool arr[size];
    //cout<<size<<endl;
    arr[0]=false;
    arr[1]=false;
    for(long long i=2;i<=size;i++)
    {
        arr[i]=true;
    }
    for(long long i=2;i<=size;i++)
```

```
15.  {
16.      for(long long j=i*i;j<=size;j+=i)
17.      {
18.          arr[j]=false;
19.      }
20.  }
21.  int t;
22.  cin>>t;
23.  while(t--)
24.  {
25.      long long l,u;
26.      cin>>l>>u;
27.      bool isPrime[u-l+1];
28.      for(int i=0;i<=u-l;i++)
29.      {
30.          isPrime[i]=true;
31.      }
32.      if(u>size)
33.      {
34.          for(long long i=2;i<=size;i++)
35.          {
36.              if(i*i>u)
37.              {
38.                  break;
39.              }
40.              if(arr[i]==true)
41.              {
42.                  long long base=(l/i)*i;
43.                  if(base<l)
44.                  {
45.                      base=base+i;
46.                  }
47.                   for(long long j=base;j<=u;j+=i)
48.                     {
49.                  isPrime[j-l]=false;
50.                     }
51.                  if(base==i)
52.                  {
53.                      isPrime[base-l]=true;
54.                  }
55.              }
56.          }
57.          for(long long i=0;i<=(u-l);i++)
58.          {
```

```
            if(isPrime[i]==true)
            {
                cout<<i+l<<" ";
            }
        }
    }
    else
    {
        for(long long i=l;i<=u;i++)
        {
            if(arr[i]==true)
            {
                cout<<i<<" ";
            }
        }
    }
    cout<<endl;
  }
  return 0;
}
```

## 4-Ass : Ninja Factor

Send Feedback

Ninja is given an array of integers of size N and Q queries, each query will be consists of two integers l, r and ninja is supposed to calculate the number of i such that the ninja factor of array[i] is a prime number where l <= i <= r

Ninja factor of a Number A is defined as the number of integers B such that

1 <= B <= A
LCM (A, B) = A * B

### Input Format:

Line1: contain two space-separated integers N and Q denoting the number of elements in array and number of queries.
Line2: contain N space-separated integers denoting the elements of the array
Next, Q lines contain two space-separated integers l, r describing the query.

### Output Format:

For each query print the answer in a newline.

### Constraints:

1 <= N , Q <= 10^5
1 <= arr[i] <= 10^9
1 <= l, r <= N

### Sample Input:

10 6
8 8 6 8 6 7 10 7 9 9
2 10

7 7
5 7
7 8
6 10
3 4

**Sample Output:**

2
0
1
0
0
1

**Explanation:**

6 is the only number whose ninja factor is prime here(i.e. 2) {[LCM(1,6]=1*6; LCM(5, 6)=5*6}

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;
int main()
{
   ll n,q;
    cin>>n>>q;
    vector<ll> a(n+1), dp(n+1, 0);

    for(ll i=1;i<=n;++i)
    {
        cin>>a[i];
        if(a[i] == 3 || a[i]==4 || a[i]==6)
            dp[i] = 1;

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

    while(q--)
    {
        ll l,r;
        cin>>l>>r;
        cout<<dp[r] - dp[l-1]<<'\n';
    }

    return 0;
}
```

# L18 : Number Theory 4

## 1-Tut : Fibonacci Sum

Send Feedback

The fibonacci sequence is defined by the following relation:

F(0) = 0
F(1) = 1
F(N) = F(N - 1) + F(N - 2), N >= 2

Your task is very simple. Given two non-negative integers N and M (N <= M), you have to calculate and return the sum (F(N) + F(N + 1) + ... + F(M)) mod 1000000007.

### Input Format :

First line of input will contain T(number of test cases), each test case follows as.
Two non-negative space-separated integers N and M. (N <= M)

### Output Format :

A new line containing the answer for the each test case.

### Constraints:

1 <= T <= 10^3
1 <= N <= M <= 10^18

### Sample Input :

1
10 19

### Sample Output :

10857

```
#include <bits/stdc++.h>
using namespace std;
#define MOD 1000000007
typedef unsigned long long ll;

void multiply(ll A[2][2], ll M[2][2])
{

    ll firstValue = A[0][0] * M[0][0] + A[0][1] * M[1][0];
        ll secondValue = A[0][0] * M[0][1] + A[0][1] * M[1][1];
        ll thirdValue = A[1][0] * M[0][0] + A[1][1] * M[1][0];
        ll fourthValue = A[1][0] * M[0][1] + A[1][1] * M[1][1];

    A[0][0] = firstValue % MOD;
    A[0][1] = secondValue % MOD;
    A[1][0] = thirdValue % MOD;
    A[1][1] = fourthValue % MOD;
```

```
}
void power(ll A[2][2], ll n)
{
    if (n == 1)
    {
        return;
    }
    power(A, n / 2);
    multiply(A, A);
    if (n % 2 != 0)
    {
        ll F[2][2] = {{1, 1}, {1, 0}};
        multiply(A, F);
    }
}
ll getFibonacci(ll n)
{
    if (n == 0 || n == 1)
    {
        return n;
    }
    ll A[2][2] = {{1, 1}, {1, 0}};
    power(A, n - 1);
    return A[0][0] % MOD;
}

int main()
{

    int t;
    cin >> t;

    while (t--)
    {
        ll m,n;
        cin >> m >> n;

        cout << (getFibonacci(n + 2)%MOD - getFibonacci(m + 1)%MOD+MOD)%MOD <<
    endl;
    }
}
```

## 2-Tut : Boring Factorials

Send Feedback

Sameer and Arpit want to overcome their fear of Maths and so they have been recently practising Maths problems a lot. Aman, their friend has been helping them out. But as it goes, Sameer and Arpit have got bored with problems involving factorials. Reason being, the factorials are too easy to calculate in problems as they only require the residue modulo some prime and that is easy to calculate in linear time. So to make things interesting for them, Aman - The Mathemagician, gives them an interesting task. He gives them a prime number P and an integer N close to P, and asks them to find N! modulo P. He asks T such queries.

### Input Format:

First line contains an integer T, the number of queries asked.
Next T lines contains T queries of the form “N P”. (quotes for clarity)

### Output Format:

Output exactly T lines, containing N! modulo P.

### Constraints:

1 <= T <= 1000
1 < P <= 2*10^9
1 <= N <= 2*10^9
Abs(N-P) <= 1000

### Sample Input:

```
3
2 5
5 11
21 71
```

### Sample Output:

```
2
10

6
```

```
#include<bits/stdc++.h>
#define ll long long int

using namespace std;

ll pow1(ll a, ll b, ll c)
{
    ll ans = 1LL;
    while (b > 0)
    {
        if (b & 1)
            ans = (ans * a) % c;
        a = (a * a) % c;
        b = b >> 1;
```

```
    }
    return ans;
}
int main()
{
    ll n, p, i, ans, fact;
    int t;
    cin>>t;
    while (t--)
    {
        fact = 1;
        cin>>n>>p;
        if (n >= p)
        {
            cout<<0<<endl;
            continue;
        }
        for (i = n + 1; i <= p - 1; i++)
        {
            fact = (fact * i) % p;
            if (fact == 0)
                break;
        }
        ans = pow1(fact, p - 2, p);
        cout<<p-ans<<endl;
    }
    return 0;
}
```

## 3-Tut : Income On Nth Day

Send Feedback

Daulat Ram is an affluent business man. After demonetization, IT raid was held at his accommodation in which all his money was seized. He is very eager to gain his money back, he started investing in certain ventures and earned out of them. On the first day, his income was Rs. X, followed by Rs. Y on the second day. Daulat Ram observed his growth as a function and wanted to calculate his income on the Nth day. The function he found out was FN = FN-1 + FN-2 + FN-1×FN-2

Given his income on day 0 and day 1, calculate his income on the Nth day (yeah Its that simple).

### Input Format:

The first line of input consists of a single integer T denoting the number of test cases.
Each of the next T lines consists of three integers F0, F1 and N respectively.

### Output Format:

For each test case, print a single integer FN, as the output can be large, calculate the answer modulo 10^9+7.

**Constraints:**

1 ≤ T ≤ 10^5

0 ≤ F0, F1, N ≤ 10^9

**Sample Input :**

2

0 1 2

1 2 4

**Sample Output:**

1

107

**Explanation**

In the second test case his income on day 0 is 1 and the income on day 1 is 2. We need to calculate his income on day 4.

F0=1

F1=2

F2=1 + 2 + 1×2 = 5

F3=2 + 5 + 2×5 = 17

F4=5 + 17 + 5×17 = 107

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define mod 1000000007

ll fib(ll n)
{
    if (n == 0 || n == 1 || n == 5)
        return n;
    if (n == 2)
        return 1;
    n--;

    ll a[2][2] = {1, 1,
                  1, 0};
    ll ans[2][2] = {
        1, 0,
        0, 1};
    ll temp[2][2];
    ll m = mod - 1, i, j, k;
    while (n)
    {
        if (n & 1)
        {
            //ans=ans*a
            for (i = 0; i < 2; i++)
```

```
            for (j = 0; j < 2; j++)
            {
                temp[i][j] = 0;
                for (k = 0; k < 2; k++)
                {
                    temp[i][j] += a[i][k] * ans[k][j];
                    temp[i][j] %= m;
                }
            }
        for (i = 0; i < 2; i++)
            for (j = 0; j < 2; j++)
                ans[i][j] = temp[i][j];
    }
    //a=a*a
    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
        {
            temp[i][j] = 0;
            for (k = 0; k < 2; k++)
            {
                temp[i][j] += a[i][k] * a[k][j];
                temp[i][j] %= m;
            }
        }
    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            a[i][j] = temp[i][j];
    n >>= 1;
  }
  return ans[0][0];
}
ll mpow(ll a, ll b)
{
  ll ans = 1;
  while (b)
  {
    if (b & 1)
      ans = (ans * a) % mod;
    a = (a * a) % mod;
    b >>= 1;
  }
  return ans;
}
int main()
```

```
{
    int t;
    ll a, b, c, n, i, x, y;
    cin >> t;
    while (t--)
    {
        cin >> a >> b >> n;
        if (n == 0)
            cout << a << endl;
        else if (n == 1)
            cout << b << endl;
        else
        {
            x = fib(n - 1);
            y = fib(n);
            //cout<<x<<" "<<y<<endl;
            c = mpow(a + 1, x) * mpow(b + 1, y);
            c--;
            c = c % mod;
            if (c < 0)
                c += mod;
            cout << c << endl;
        }
    }
    return 0;
}
```

## 4-Ass : Cubic Square

Send Feedback

Varun is learning method of successive squaring so that he can calculate a^b mod m quickly. To give himself practice he wrote many tuples of a, b and m and went to school thinking that he will do it after school.

After school he found that tuples he wrote are modified by his little sister. His sister converted each b into base 3. Varun wrote everything in base 10.

Help Varun to do his exercise.

### Input Format:

First line of input contains a number T(number of test case). Each test case contains an integer a(base 10) followed by a string b (base 3) followed by integer m (base 10). All are space-separated.

### Output Format;

Output a number for each test case a^b mod m in base 10 in new line.

### Constraints:

1 <= T <= 1000

1 <= a, m <= 10^9

Number of digits in b will be less than 250.

**Sample Input:**

```
2
2 10 10
3 21101 19
```

**Sample Output:**

```
8

3
```

```
#include<bits/stdc++.h>
using namespace std;
int power(long long int a, string b, int m)
{
	int len = b.length();
	long long int final_ans = 1;
	for(int i=len-1;i>=0;i--)
	{
		if(b[i] == '0')
		{
			a %= m;
			a = (((a*a)%m)*a)%m;
		}
		if(b[i] == '1')
		{
			final_ans = final_ans * a;
			final_ans %= m;
			a = (((a*a)%m)*a)%m;
		}
		if(b[i] == '2')
		{
			final_ans = ((final_ans * a)%m) * a;
			final_ans %= m;
			a = (((a*a)%m)*a)%m;
		}
	}
	return final_ans;
}
int main()
{
	long long int t;
	cin >> t;
	//write code here
	while(t--)
	{
```

```
                long long int a; //in base 10
                string b; //in base 3
                int m; //in base 10
                cin >> a >> b >> m;

                cout << power(a,b,m) << endl;
        }

        return 0 ;

}
```

## 5-Ass : GCD Extreme

Send Feedback

Given the value of N, you will have to find the value of G. The meaning of G is given in the following code

```
G=0;
for(i = 1 ; i < N ; i++)
  for(j = i+1 ; j <= N ; j++)
     G+=gcd(i,j);
```

Here gcd() is a function that finds the greatest common divisor of the two input numbers.

### Input Format:

The first line of input will contain T(number of the test case). Each test case contains an integer N.

### Output Format:

For each test case print the answer in a new line.

### Constraints:

1 <= T <= 10^5
1 <= N <= 10^5

### Sample Input:

```
3
10
100
3
```

### Sample Output:

```
67
13015
```

3

```
#include <bits/stdc++.h>

using namespace std;

#define endl '\n'
```

```
#define ll long long int
#define MAX 1000001

ll phi[MAX];
unsigned ll S[MAX], G[MAX];

int main()
{
        unsigned ll i, j, n;
    phi[1] = 1;
    for (i = 2; i < MAX; i += 2)
    {
        phi[i] = i / 2;
    }
    for (i = 3; i < MAX; i += 2)
    {
        if (!phi[i])
        {
            phi[i] = i - 1;
            for (j = i << 1; j < MAX; j += i)
            {
                if (!phi[j])
                {
                    phi[j] = j;
                }

                phi[j] = phi[j] / i * (i - 1);
            }
        }
    }
    for (i = 0; i < MAX; i++)
        S[i] = phi[i];
    for (i = 2; i < MAX; i++)
    {
        for (j = 2; j * i < MAX; j++)
        {
            S[i * j] += j * phi[i];
        }
    }
    G[1] = 0;
    for (i = 2; i < MAX; i++)
        G[i] = G[i - 1] + S[i];

    ll t;
```

```
    cin>>t;
    while (t--)
    {
        cin >> n;
        if (n == 0)
            break;
        cout << G[n] << endl;
    }

    return 0;
}
```

## 6-Ass : Sanchit And Nuclear Reactor

Send Feedback

We all know Sanchit Lee Cooper who is a Caltech theoretical physicist. He has eccentric and arrogant behavior. Due to his belief that he's intellectually superior, he's not ashamed to insult his own friends, like Howard, who is an engineer and not a real scientist. But nobody messes with an engineer. So Howard accepted an challenge from Sanchit. Sanchit was involved in numerous experiments as a wunderkind, such as his plan for building his own nuclear reactor - a plan stopped by government. So Sanchit presented Howard with a problem about his own nuclear reactor. It contains a large tank and at each second an atom is introduced in the tank which reacts with already existing atoms and produces some energy. Also he defined a special threshold number for his reactor called Cooper number m which is always a prime number. Energy output is defined as previous energy output of the tank multiplied by number of atoms present in it. But due to some special condition of the tank, all atoms attains stable state when number of atoms are multiple of Cooper number and no new reaction occurs. Energy output in this case is same as previous case. Also initial energy of the reactor is 1 and initially there is no atom in the tank. Now Sanchit ask Howard to tell the energy output after time T. But sadly Howard is not able to solve it and ask for your help.

### Input Format

The first line of input contains T(number of the test case), each test case follows as.
contian two space-separated integers N and M where M is a prime number.

### Output Format

You have to determine the energy output after time T. As the number can be quite large so output it modulo Cooper number m.

### Constraints:

1 <= T <= 100
1 <= N <= 10^18
1 <= M <= 10^4

### Sample Input

2
1 5

2 5

## Sample Output

1
2

## Explanation

After 1 seconds, there is only 1 atom in the tank. Hence energy output is 1. After 2 seconds, there are 2 atoms which reacts to give energy output of 2.

```
#include<bits/stdc++.h>
using namespace std;

#define endl '\n'
#define ll long long int

long long fact(ll a, ll m){
        long long res=1;
        for(ll i=2;i<=a;i++)
                res=(res%m * i%m)%m;
        return res;
}
int main(){

        ll n;
        cin>>n;
        while(n--){
                ll t;
                ll m;
                cin>>t>>m;
                if(t<m){
                        if(m-t==1){
                                cout<<1<<endl;
                                continue;
                        }
                        else{
                                cout<<fact(t, m)%m<<endl;
                                continue;
                        }
                }
                else{
                        ll last = t%m;
                        ll facto=fact(last, m)%m;

                        if((t/m)%2==0)
                                cout<<facto<<endl;
```

```
                else
                    cout<<((m-1)%m*facto%m)%m<<endl;
            }
        }
        return 0;
}
```

## 7-Ass : Innocent Swaps and His Emotions

Send Feedback

There are only three phases in Swaps life: Sleep, Play and Study. Also, there are two types of emotions Swaps experiences: Happy and Sad. Each phase of his life brings either kind of emotions.

The sleep and the play phase makes Swaps happy whereas the study phase makes him sad. Quite obvious, isn't it? But we know that life isn't that great, one cannot be happy all the time.

Swaps, being a very sensitive guy, doesn't like to mix his emotions on a particular day. So each day, he is in exactly one of the three phases.

Given N which denotes the number of days and K which denotes the exact number of days Swaps needs to be happy out of these N days, can you tell him in how many ways can he achieve this? Since the output value can be very large, take modulo with 1000000007(10^9+7)

### Input Format:

The first line of the input contains T, denoting the number of test cases.
The next T lines contain two space-separated integers N and K.

### Constraints:

1 <=T <= 10^5
1<= K <= N <= 10^6

### Output Format:

For each test-case, output a single integer, the number of ways modulo 1000000007(10^9+7).

### Sample Input 1:

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

### Sample Output 2:

```
2
4
12
```

### Explanation

In the first test case, he needs to feel joyful on Day 1. Hence, answer is 2 (He can either play video games or sleep).

In the second test case, he can be joyful either on Day 1 or Day 2. So number of ways = 4.

```
1. #include <bits/stdc++.h>
2. using namespace std;
```

```
#define ll long long
#define MAX 1000001
#define mod 1000000007
ll fact[MAX];
ll modexpo(ll a, ll b)
{
    ll ans = 1;
    while (b)
    {
        if (b & 1)
            ans = (ans * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return ans;
}
int main()
{
        ll n,
        i, t, k, ans;
    fact[0] = 1;
    for (i = 1; i < MAX; i++)
    {
        fact[i] = i * fact[i - 1];
        if (fact[i] >= mod)
            fact[i] %= mod;
    }
    cin >> t;
    while (t--)
    {
        cin >> n >> k;
        if (k > n)
        {
            cout << endl;
            continue;
        }
        ans = modexpo(2, k);
        ans = (ans * fact[n]) % mod;
        ans = (ans * modexpo(fact[k], mod - 2)) % mod;
        ans = (ans * modexpo(fact[n - k], mod - 2)) % mod;
        cout << ans << endl;
    }
    return 0;
}
```

## 8-Ass : Sehwag and ETF

Send Feedback

Sehwag has been solving a lot of mathematical problems recently. He was learning ETF (Euler Totient Function) and found the topic quite interesting. So, he tried solving a question on ETF. He will be given two numbers L and R. He has to find the probability that the ETF of a number in the range [L, R] is divisible by a number K.

### Input Format:

The first line contains an integer T, representing the number of test cases.
The next T lines will contain three integers L, R and K.

### Constraints:

1 <= T <= 10
1 <= L <= R <= 10^12
0 <= R - L <= 10^5
1 <= K <= 10^6

### Output Format:

Print the answer in a new line after rounding off the first 6 digits after the decimal place.

### Sample Input 1:

```
3
1 4 2
2 5 2
3 10 4
```

### Sample Output 1:

```
0.500000
0.750000

0.375000
```

```
1. #include<bits/stdc++.h>
2. using namespace std;
3.
4. #define ll long long
5. #define MAX 1100001
6. #define ss(n) scanf("%lld", &n)
7.
8. bool primes[MAX];
9. ll phi[MAX];
10. ll p[300001];
11. ll num[MAX];
12. int main()
13. {
14.     ll t, a, b, k, i, j, x, base;
15.     cin >> t;
16.     for (i = 3; i < MAX; i += 2)
```

```
    primes[i] = 1;
primes[2] = 1;
for (i = 3; i * i < MAX; i += 2)
{
    if (primes[i])
    {
        for (j = i * i; j < MAX; j += 2 * i)
            primes[j] = 0;
    }
}
p[0] = 2;
ll c = 1;
for (i = 3; i < MAX; i += 2)
{
    if (primes[i])
        p[c++] = i;
}
while (t--)
{
    cin >> a >> b >> k;
    if (k == 1)
    {
        std::cout << std::fixed;
        std::cout << std::setprecision(6) << 1 << endl;
        continue;
    }
    for (i = a; i <= b; i++)
    {
        phi[i - a] = i;
        num[i - a] = i;
    }
    for (i = 0; p[i] * p[i] <= b; i++)
    {
        base = a / p[i] * p[i];
        while (base < a)
            base += p[i];
        while (base < p[i])
            base += p[i];
        if (base == p[i])
        {
            // cout<<phi[base-a]<<endl;
            base += p[i];
        }
        for (j = base; j <= b; j += p[i])
```

```
61.             {
62.                 while (num[j - a] % p[i] == 0)
63.                     num[j - a] /= p[i];
64.                 phi[j - a] -= phi[j - a] / p[i];
65.                 //cout<<j<<" "<<phi[j-a]<<endl;
66.             }
67.         }
68.         for (i = a; i <= b; i++)
69.         {
70.             if (num[i - a] > 1)
71.                 phi[i - a] -= phi[i - a] / num[i - a];
72.             num[i - a] = 1;
73.         }
74.         ll c = 0;
75.         for (i = a; i <= b; i++)
76.         {
77.             //cout<<i<<" "<<phi[i-a]<<endl;
78.             if (phi[i - a] % k == 0)
79.                 c++;
80.         }
81.         double ans = c;
82.         ans /= (b - a + 1);
83.         std::cout << std::fixed;
84.         std::cout << std::setprecision(6) << ans << endl;
85.     }
86.     return 0;
87. }
```

## Understanding Purpose : Nth fib number optimized

```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.
4.
5.  void multiply(int A[2][2],int M[2][2]){
6.
7.          int firstValue = A[0][0] * M[0][0] + A[0][1] * M[1][0];
8.          int secondValue = A[0][0] * M[0][1] + A[0][1] * M[1][1];
9.          int thirdValue = A[1][0] * M[0][0] + A[1][1] * M[1][0];
10.         int fourthValue = A[1][0] * M[0][1] + A[1][1] * M[1][1];
11.
12.         A[0][0] =firstValue;
13.         A[0][1] = secondValue;
14.         A[1][0] = thirdValue;
15.         A[1][1] = fourthValue;
16.
17. }
18. void power(int A[2][2],int n){
19.         if(n==1){
20.                 return;
21.         }
22.         power(A,n/2);
23.         multiply(A,A);
24.         if(n%2 !=0){
25.                 int F[2][2] = {{1,1},{1,0}};
26.                 multiply(A,F);
27.         }
28. }
29. int getFibonacci(int n){
30.         if(n==0 || n==1){
31.                 return n;
32.         }
33.         int A[2][2] = {{1,1},{1,0}};
34.         power(A,n-1);
35.         return A[0][0];
36. }
37. int main(){
38.         int n;
39.         cin >> n;
40.         cout << getFibonacci(n)<<endl;
41.         return 0;
42. }
```

# L19 : GreedyProblems

## 1-Tut : Min. Absolute Difference In Array

Send Feedback

Given an integer array A of size N, find and return the minimum absolute difference between any two elements in the array.

We define the absolute difference between two elements $a_i$ and $a_j$ (where i != j ) as $|a_i - a_j|$.

### Input format :

The first line of input contains an integer that denotes the number of test cases. Let us denote it by the symbol T.

Each of the test cases contain two lines. The first line of each test case contains an integer, that denotes the value of N. The following line contains N space separated integers, that denote the array elements.

### Constraints :

1 <= T <= 10

2 <= N <= 10^5

0 <= arr[i] <= 10^8

### Output Format :

You have to print minimum difference for each test case in new line.

### Sample Input 1:

```
1
5
2 9 0 4 5
```

### Sample Output 1:

```
1
```

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long int

int mod(int a)
{
        if(a<0)
        {
                return -a;
        }
        return a;
}
ll minAbsoluteDiff(ll * arr, int n) {
        sort(arr, arr+n);
        ll i=0;
        ll j=1;
        ll minimum=INT_MAX;
```

```
        while(j<n)
        {
                if(mod(arr[i]-arr[j])<minimum)
                {
                        minimum=mod(arr[i]-arr[j]);
                }
                i++;
                j++;
        }
        return minimum;
}

int main(){

    // write your code here
    int t;
    cin>>t;
    while(t--){
        int n;
        cin>>n;
        ll * arr = new ll[n];
        for(int i=0;i<n;i++){
            cin >> arr[i];
        }
        ll ans = minAbsoluteDiff(arr,n);
        cout<<ans<<endl;
    }
    return 0;
}
```

## 2-Tut : Nikunj and Donuts

Send Feedback

Nikunj loves donuts, but he also likes to stay fit. He eats n donuts in one sitting, and each donut has a calorie count, ci. After eating a donut with k calories, he must walk at least 2^j * k(where j is the number donuts he has already eaten) miles to maintain his weight.

Given the individual calorie counts for each of the n donuts, find and print a long integer denoting the minimum number of miles Nikunj must walk to maintain his weight. Note that he can eat the donuts in any order.

### Input Format:

First line of input will contain T(number of test cases), each test case follows as.
The first line contains an integer, n, denoting the number of donuts.
The second line contains n space-separated integers describing the respective calorie counts of each donut, i.e ci.

**Constraints**

1 <= T <= 10^5

1 ≤ n ≤ 40

1 ≤ ci ≤ 1000

**Output Format:**

Print a long integer denoting the minimum number of miles Nikunj must walk to maintain his weight for each test case in new line

**Sample Input 1**

1

3

1 3 2

**Sample Output 1**

11

**Explanation:**

The minimum miles that Nikunj has to walk are 11 miles. He will first eat donut with calorie count 3, then the one with calorie count 2 and then the donut with calorie count as 1. The miles travelled are calculated as: 2^0*3 + 2^1*2 + 2^2*1.

```
#include<bits/stdc++.h>
using namespace std;

#define ll unsigned long long int

ll minimum_miles(ll *arr, ll n)
{
        ll total_miles=0;
        for(ll i=0; i<n; i++)
        {
                total_miles+=pow(2, i)*arr[n-1-i];
        }
        return total_miles;
}
int main()
{
    int t;
    cin>>t;

    while(t--){
       ll n;
                cin>>n;
                ll *arr=new ll [n];
                for(ll i=0; i<n; i++)
                {
                        cin>>arr[i];
```

```
                }
                sort(arr, arr+n);
                cout<<minimum_miles(arr, n)<<endl;
    }

}
```

## 3-Tut : Activity Selection

Send Feedback

You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

### Input Format

The first line of input contains one integer denoting N.
Next N lines contains two space separated integers denoting the start time and finish time for the ith activity.

### Constraints

1 ≤ N ≤ 10^6
1 ≤ ai, di ≤ 10^9

### Output Format

Output one integer, the maximum number of activities that can be performed

### Sample Input 1

6
1 2
3 4
0 6
5 7
8 9
5 9

### Sample Output 1

4

### Explanation:

The four activities will be done at following time intervals:
1. 1 2
2. 3 4
3. 5 7
4. 8 9

```
#include<bits/stdc++.h>
using namespace std;

bool compare(pair<int, int> i1, pair<int, int> i2)
{
        return i1.second<i2.second;
}
```

```
int activities(pair<int, int>*arr, int n)
{
        int current_ending_time=arr[0].second;
        int count=1;
        for(int i=1; i<n; i++)
        {
                if(current_ending_time<=arr[i].first)
                {
                        count+=1;
                        current_ending_time=arr[i].second;
                }
        }
        return count;
}
int main()
{
        int n;
        cin>>n;
        pair<int, int> *arr=new pair<int, int>[n];
        for(int i=0; i<n; i++)
        {
                cin>>arr[i].first>>arr[i].second;
        }
        sort(arr, arr+n, compare);
        cout<<activities(arr, n)<<endl;
}
```

## 4-Tut : Fractional Knapsack

Send Feedback

You are given weights and values of N items. You have to select and put these selected items in a knapsack of capacity W. Select the items in such a way that selected items give the maximum total value possible with given capacity of the knapsack.

**Note:** You are allowed to break the items in parts.

### Input Format:

The first line of input contains two space separated integers, that denote the value of N and W.
Each of the following N lines contains two space separated integers, that denote value and weight, respectively, of the N items.

### Constraints:

1 <= N = 2*10^5
1 <= W <= 10^9
1 <= value, weight <= 10^5
Time Limit: 1 sec

### Output Format:

Print the maximum total value upto six decimal places.

**Sample Input 1:**

```
4 22
6 4
6 4
4 4
4 4
```

**Sample Output 1:**

```
20.000000
```

**Explanation:**

The total weight of all the items is 16 units, which is less than the total capacity of knapsack, i.e 22 units. Hence, we will add all the items in the knapsack and total value will be 20 units.

```
#include <bits/stdc++.h>
using namespace std;

struct Item {
        int value;
    int weight;
};

bool cmp(struct Item a, struct Item b)
{
        double r1 = (double)a.value / a.weight;
        double r2 = (double)b.value / b.weight;
        return r1 > r2;
}

double fractionalKnapsack(Item * arr, int N, int size)
{
        sort(arr, arr + size, cmp);

        int curWeight = 0;

        double finalvalue = 0.0;

        for (int i = 0; i < size; i++) {

                if (curWeight + arr[i].weight <= N) {
                        curWeight += arr[i].weight;
                        finalvalue += arr[i].value;
                } else {
                        int remain = N - curWeight;
                        finalvalue += arr[i].value * ((double)remain / arr[i].weight);
                        break;
```

```
                }
            }

            return finalvalue;
}

int main()
{
            int n;
    long long int w;
    cin >> n >> w;

    Item * arr = new Item [n];

    for(int i=0;i<n;i++){
        cin>>arr[i].value>>arr[i].weight;
    }

            double ans = fractionalKnapsack(arr, w, n);
    cout.precision(6);
            cout << fixed << ans << endl;
            return 0;
}
```

## 5-Tut : Weighted Job Scheduling

Send Feedback

You are given N jobs where every job is represented as:

1. Start Time

2. Finish Time

3. Profit Associated

Find the maximum profit subset of jobs such that no two jobs in the subset overlap.

### Input Format:

The first line of input contains an integer denoting N.
Next N lines contains three space separated integers denoting the start time, finish time and the profit associated with the ith job.

### Constraints:

1 ≤ N ≤ 10^6
1 ≤ ai, di, p ≤ 10^6

### Output Format:

Output one integer, the maximum profit that can be achieved.

### Sample Input 1

4
3 10 20

```
1 2 50
6 19 100
2 100 200
```

**Sample Output 1**

```
250
```

```
#include<bits/stdc++.h>
using namespace std;

using namespace std;
// A job has start time, finish time and profit.
struct Job
{
    int start, finish, profit;
};
// A utility function that is used for sorting events
// according to finish time
bool myfunction(Job s1, Job s2)
{
    return (s1.finish < s2.finish);
}
// A Binary Search based function to find the latest job
// (before current job) that doesn't conflict with current
// job. "index" is index of the current job. This function
// returns -1 if all jobs before index conflict with it.
// The array jobs[] is sorted in increasing order of finish
// time.
int binarySearch(Job jobs[], int index)
{
// Initialize 'lo' and 'hi' for Binary Search
    int lo = 0, hi = index - 1;
// Perform binary Search iteratively
    while (lo <= hi)
    {
        int mid = (lo + hi) / 2;
        if (jobs[mid].finish <= jobs[index].start)
        {
            if (jobs[mid + 1].finish <= jobs[index].start)
                lo = mid + 1;
            else
                return mid;
        }
        else
            hi = mid - 1;
```

```
    }
    return -1;
}
// The main function that returns the maximum possible
// profit from given array of jobs
int findMaxProfit(Job arr[], int n)
{
    // Sort jobs according to finish time
    sort(arr, arr+n, myfunction);
    // Create an array to store solutions of subproblems. table[i]
    // stores the profit for jobs till arr[i] (including arr[i])
    int *table = new int[n];
    table[0] = arr[0].profit;
    // Fill entries in table[] using recursive property
    for (int i=1; i<n; i++)
    {
        // Find profit including the current job
        int inclProf = arr[i].profit;
        int l = binarySearch(arr, i);
        if (l != -1)
            inclProf += table[l];
        // Store maximum of including and excluding
        table[i] = max(inclProf, table[i-1]);
    }
    // Store result and free dynamic memory allocated for table[]
    int result = table[n-1];
    delete[] table;
    return result;
}
// Driver program
int main()
{
    int n;
    cin>>n;
    Job arr[n];
    for(int i=0;i<n;i++)
    {
        cin>>arr[i].start>>arr[i].finish>>arr[i].profit;
    }
    cout << findMaxProfit(arr, n);
    return 0;
}
```

**Live Problem 1 : https://www.codechef.com/problems/SVENGY**
**Live Problem 2 : https://dmoj.ca/problem/coci13c1p4**

## 6-Ass : Winning Lottery

Send Feedback

Harshit knows through his resources that this time the winning lottery number is the smallest number whose sum of the digits is S and the number of digits is D. You have to help Harshit and print the winning lottery number.

### Input Format:

First line of input contians an integer T, representing the number of test cases.
Next T lines contains two space-separated integers: S,D

### Constraints:

1 <= T <= 1000
1 <= D <= 1000
1 <= S <= 9*D
Time Limit: 1 second

### Output Format

The output contains a single integer denoting the winning lottery number

### Sample Input 1:

1
9 2

### Sample Output 1:

18

### Explanation

There are many other possible numbers like 45, 54, 90, etc with the sum of digits as 9 and number of digits as 2. The smallest of them is 18.

```
#include<bits/stdc++.h>
using namespace std;
void findSmallest(int m, int s)
{

    if (s == 0)
    {
        (m == 1)? cout << 0
                : cout << "Not possible";
        return ;
    }

    if (s > 9*m)
    {
        cout << "Not possible";
        return ;
```

```
    }

    int res[m];

    s -= 1;

    for (int i=m-1; i>0; i--)
    {

        if (s > 9)
        {
            res[i] = 9;
            s -= 9;
        }
        else
        {
            res[i] = s;
            s = 0;
        }
    }

    res[0] = s + 1;


    for (int i=0; i<m; i++)
        cout << res[i];
}


int main()
{
    int t;
    cin>>t;
    while(t--){

        int s, d;
                    cin >> s >> d;
                    findSmallest(d, s);
        cout<<endl;

    }

}
```

## 7-Ass : King Arthur and Excalibur

Send Feedback

King Arthur's land is in danger. He wants to fend off the Saxon invaders. In order to do so, he needs to find the fabled sword, Excalibur. He has a vague idea about its location, and has dispatched his best Knights and their Squires to acquire it. The Knights go to the location to find The Dark Forest. They set up tents and decided to split into groups to find Excalibur faster. The famous knight, Sir Lancelot was trying to form search parties, but ran into some trouble.

Most of the knights were untrained for quests, and sending them alone into the dangers of the quest would be a grave mistake. Each of the knights have been given a positive integer parameter by King Arthur, ti - their training score. Sir Lancelot decided that a knight with training t can join the group of t or more knights.

Some of the knights are very egotistical and can't work well with others. Hence Sir Lancelot decided it is not necessary to include all the knights, some can stay back and defend the tents from danger. Now Sir Lancelot needs to figure out how many search parties he can organize. Sir Lancelot is one of the most responsible knights and doesn't want to go back to King Arthur empty handed. He would rather die than fail. You need to help him in his quest.

### Input Format:

First line of input will contain T(number of test cases), each test case follows as.
Line1 : will contain an integer N denoting the total number of explorer
Line2: Will contain N space-separated integers denoting the inexperience level

### Ouput Format:

For each test case output the answer in a new line.

### Constraints:

1 <= T <= 100
1 <= N <= 10^4
1 <= levels <= N

### Sample Input:

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

### Sample Output:

```
3

2
```

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4.
5. int main()
```

```
{

        int t;
    cin >> t;
        while(t--)
        {
                int n;
      cin >> n;
                vector<int> a(n+1);
                for(int i = 1 ; i <= n ; ++i) {
                        cin >> a[i];
                }
                sort(a.begin()+1, a.end());

                vector<int> dp(n+1, 0);

                int ans = 0;
                for(int i = 1 ; i <= n ; ++i) {
                        if(i - a[i] >= 0) {
                                dp[i] = max(dp[i-1], dp[i-a[i]] + 1);
                        }
                        else {
                                dp[i] = dp[i-1];
                        }
                }

                cout << dp[n] << endl;
        }
        return 0;
}
```

## 8-Ass : Call of War

Send Feedback

DecenTile Games have launched a new First Person Shooter soldier game, called the Call of War, that young gamers can play on their website.

Our protagonist, Ninja (played by you) is a soldier whose mission is to kill all the enemies plotting against you. Your enemies are standing in a circle, numbered clockwise from 1 to n. Initially, the i-th enemy has ai health.

You have only one pistol, and infinite bullet magazines. You have to shoot the enemies in order to kill them. Each bullet fired at the enemy decreases their health by 1 (deals 1 damage to it). When the health of some enemy i becomes 0 or less than 0, he dies and his grenade drops down and explodes, dealing bi damage to the next enemy (enemy i+1, if i<n, or enemy 1, if i=n). If the next enemy is already dead, then

nothing happens. If the frag from the grenade kills the next enemy, even he drops a grenade, damaging the enemy after him and possibly triggering another explosion, and so on.
You have to calculate the minimum number of bullets you need in order to kill all the enemies and win the game.

### Input Format:
First line of input will contain T(number of test cases), each test case follows as.
Line 1: contain an integer N
Next N line contains two space separated integers a and b
### Output Format:
For each test case print the output in a new line.
### Constraints:
1 <= T <= 100
1 <= N <= 10^4
1 <= a, b <= 10^12
### Sample Input:
1
3
7 15
2 14
5 3
### Sample Output:

6

```
#include<bits/stdc++.h>
using namespace std;

#define ll long long int

int main()
{


        int t;
    cin >> t;
        while(t--)
        {
                int n;
      cin >> n;
      ll * a = new ll[n];
      ll * b = new ll[n];
      ll * d = new ll[n];

      ll * ans = new ll[n];
```

```
21.
22.         for(int i=0;i<n;i++){
23.             cin >> a[i]>> b[i];
24.         }
25.
26.         d[0] = a[0] - b[n-1];
27.         if(d[0]<0){
28.             d[0]=0;
29.         }
30.
31.         ll sumD = d[0];
32.
33.         for(int i=1;i<n;i++){
34.
35.             d[i] = a[i]-b[i-1];
36.             if(d[i]<0){
37.                 d[i]=0;
38.             }
39.             sumD+=d[i];
40.         }
41.
42.         ll minCost = LLONG_MAX;
43.         for(ll i=0 ; i<n ; i++){
44.
45.             ans[i] = a[i] + sumD - d[i];
46.             if(ans[i]<minCost){
47.                 minCost = ans[i];
48.             }
49.
50.         }
51.         cout<<minCost<<endl;
52.
53.             }
54.             return 0;
55. }
```

## 9-Ass : Minimum Number of Platforms

Send Feedback

You have been given two arrays, 'AT' and 'DT', representing the arrival and departure times of all trains that reach a railway station.

Your task is to find the minimum number of platforms required for the railway station so that no train needs to wait.

**Note :**

1. Every train will depart on the same day and the departure time will always be greater than the arrival time. For example, A train with arrival time 2240 and departure time 1930 is not possible.

2. Time will be given in 24H format and colons will be omitted for convenience. For example, 9:05AM will be given as "905", or 9:10PM will be given as "2110".

3. Also, there will be no leading zeroes in the given times. For example, 12:10AM will be given as “10” and not as “0010”.

### Input Format

The first line of input contains an integer 'T' representing the number of test cases.
The first line of each test case contains an integer 'N', representing the total number of trains.

The second line of each test case contains 'N' single-spaced separated elements of the array 'AT', representing the arrival times of all the trains.

The third line of each test case contains 'N' single-spaced separated elements of the array 'DT', representing the departure times of all the trains.

### Output Format:

For each test case, return the minimum number of platforms required for the railway station so that no train needs to wait.

### Note :

You don't need to print the output, it has already been taken care of. You just need to implement the given function.

### Follow up :

Try to solve the problem in O(N) time and O(1) space.

### Constraints:

1 <= T <= 10
1 <= N <= 50000
0 <= AT[i] <= DT[i] <= 2359
Where 'AT[i]' and 'DT[i]' are the elements of the arrival and the departure arrays respectively.
Time limit: 1 sec

### Sample Input 1:

2
6
900 940 950 1100 1500 1800
910 1200 1120 1130 1900 2000
4
100 200 300 400
200 300 400 500

### Sample Output 1:

3
2

### Explanation of the Sample Output 1:

In test case 1, For the given input, the following will be the schedule of the trains:
Train 1 arrived at 900 on platform 1.
Train 1 departed at 910 from platform 1.
Train 2 arrived at 940 on platform 1.

Train 3 arrived at 950 on platform 2 (since platform 1 was already occupied by train 1).
Train 4 arrived at 1100 on platform 3 (since both platforms 1 and 2 were occupied by trains 2 and 3 respectively).
Train 3 departed at 1120 from platform 2 (platform 2 becomes vacant).
Train 4 departed at 1130 from platform 3 (platform 3 also becomes vacant).
Train 2 departed at 1200 from platform 1 (platform 1 also becomes vacant).
Train 5 arrived at 1500 on platform 1.
Train 6 arrived at 1800 on platform 2.
Train 5 departed at 1900 from platform 1.
Train 6 departed at 2000 from platform 2.

Thus, minimum 3 platforms are needed for the given input.

In test case 2, For the given input, the following will be the schedule of the trains:

Train 1 arrived at 100 on platform 1.
Train 2 arrived at 200 from platform 2 (as platform 1 is occupied by train 1).
Train 1 departed at 200 from platform 1.
Train 3 arrived at 300 on platform 1.
Train 2 departed at 300 from platform 2.
Train 4 arrived at 400 on platform 2.
Train 3 departed at 400 from platform 1.
Train 4 departed at 500 from platform 2.

Thus, 2 platforms are needed for the given input.

### Sample Input 2:

```
2
2
900 1000
999 1100
3
1200 1300 1450
1310 1440 1600
```

### Sample Output 2:

```
1
2
```

### Explanation of the Sample Output 2:

In test case 1, For the given input, the following will be the schedule of the trains:

Train 1 arrived at 900 on platform 1.
Train 1 departed at 999 from platform 1.
Train 2 arrived at 1000 on platform 1.
Train 2 arrived at 1100 on platform 1.

Thus, only 1 platform is needed for the given input.

In test case 2, For the given input, the following will be the schedule of the trains:

Train 1 arrived at 1200 on platform 1.
Train 2 arrived at 1300 on platform 2. (since platform 1 was already occupied by train 1).
Train 1 departed at 1310 from platform 1.
Train 2 departed at 1440 from platform 2.
Train 3 arrived at 1450 on platform 1.
Train 3 departed at 1600 on platform 1.

Thus, minimum 2 platforms are needed for the given input.

```
#include<bits/stdc++.h>
using namespace std;

int calculateMinPatforms(int at[], int dt[], int n) {
    // Write your code here.

    sort(at,at+n);
    sort(dt,dt+n);

    int plat_count=1;
    int result = 1;

    for(int i=1, j=0; i<n && j<n ;){
        if(at[i]<=dt[j]){
            plat_count++;
            i++;
        } else if(dt[j]<at[i]){
            plat_count--;
            j++;
        }


        if(plat_count>result){
            result = plat_count;
        }
    }

    return result;
}
```

## 10-Ass : Gas Tank

Send Feedback

You have a car with a gas tank of infinite capacity. There are ‘N’ gas stations along a circular route. Gas stations are numbered from 0 to N - 1. You begin the journey with an empty tank at one of the gas

stations. You want to travel around the circular route once in the clockwise direction. I.e if you start to travel from station ‘i’, then you will go to i + 1, i + 2, …, n - 1, 0, 1, 2, i - 1 and then return back to ‘i’.
You are given two integer arrays ‘gas’ and ‘cost’. ‘gas[i]’ gives the amount of gas available at station ‘i’ and cost[i] gives the amount of gas required to travel from station ‘i’ to next station i.e station ‘i’+1 (or 0 if the station is N - 1).
Return the starting gas station's index if it is possible to complete cycle of given circular route once in the clockwise direction. If there are multiple possible starting gas station’s indexes, then return the minimum of those gas station indexes. If there is no such gas station then return -1.

### Input Format :

The first line of input contains a single integer T, representing the number of test cases or queries to be run, then the T test cases follow.

The first line of each test case contains a positive integer 'N' which represents the number of gas stations.

The second line of each test case contains 'N' space-separated integers representing the integer array 'gas'.

The third line of each test case contains 'N' space-separated integers representing the integer array 'cost'.

### Output Format :

For each test case, print a single integer denoting the minimum index of the starting gas station if you are able to travel around the cycle once, otherwise print -1.

### Note:

You do not need to print anything, it has already been taken care of. Just implement the given function.

### Constraint :

1 <= T <=50
1 <= N <= 10 ^ 4
0 <= GAS[i] <= 10 ^ 5
0 <= COST[i] <= 10 ^ 5
Where GAS[i] represents the ith element of 'GAS' array,
COST[i] represents the ith element of 'COST' array.
Time Limit: 1 sec

### Sample Input 1 :

```
1
2
1 2
2 1
```

### Sample Output 1:

```
1
```

### Explanation of Sample Input 1 :

Test Case 1:
If you start from index 0, you can fill in 1 unit of gas. Now your tank has 1 unit of gas. But you need 2 units of gas to travel to station 1. Thus you can not start at station 0.
If you start from index 1, you can fill in 2 units of gas. Now your tank has 2 units of gas. You need 1 unit of gas to get to station 0. So, you travel to station 0 and still have 1 unit of gas left. You fill in 1 unit of

additional gas, making your current gas = 2 unit. It costs you 2 amounts of gas to get to station 1, which you do and complete the cycle. Thus you can start at index 1.

**Sample Input 2 :**

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

**Sample Output 2:**

3

**Explanation of Sample Input 2 :**

Test Case 1:
If you start from index 3 and fill all available gas at each station, then you can reach station 4 with 3 units of gas, station 0 with 6 units of gas, station 1 with 4 units of gas, station 2 with 2 units of gas, and back to station 3 after consuming all the gas.

We can show that index 3 is the minimum index of the gas station from where the complete circular trip is possible

```
#include <iostream>
#include <vector>

using namespace std;


int minimumStartingIndex(vector<int> &gas, vector<int> &cost, int n)
{
    // Write your code here.


        int start = 0;
    int total = 0;
    int tank = 0;
    for(int i = 0; i < n ; i++) {
        tank = tank + gas[i] - cost[i];
        if(tank < 0) {
            start = i+1;
            total= total + tank;
            tank=0;
        }
    }
    if(total + tank < 0) {
        return -1;
    } else {
        return start;
    }
}
```

# L20 : Range Query 1

Github : jitendrabhadoria

## 1-Tut : Minimum In SubArray

Send Feedback

You are given a sequence A[1], A[2], ..., A[N] of N numbers ( 0 ≤ A[i] ≤ 10^8 , 2 ≤ N ≤ 10^5 ). There are Q queries, and each query is defined in either of the following two ways:

**Query on range**:

You are given two numbers i and j; the answer to each query is the minimum number between the range (i, j) (including both i and j). Note that in this query, i and j lies in the range: [1, N].

**Update query**:

You are given two numbers i and B; update A[i] to B. Note that in this query, i lies in the range: [1, N].

### Input Format:

The first line of input contains two integers: N and Q, representing the length of the sequence and the number of queries.
The second line of input contains N space-separated integers, A[i].
Next, Q lines contain the queries. Each line contains one character, followed by two space-separated integers. For the query on range, the character will be q and for the update query, the character will be u.

### Constraints:

1 ≤ i ≤ N
0 ≤ B ≤ 10^8
1 ≤ i <= j ≤ N

### Output Format:

For each query on range, print the minimum number between the range, in a new line.

### Sample Input 1:

```
4 3
5 2 4 3
q 1 3
u 1 1
q 3 4
```

### Sample Output 1:

```
2

3
```

```
1. #include<bits/stdc++.h>
2. using namespace std;
3.
4. void buildtree(int* arr, int* tree, int start, int end, int treenode)
5. {
6.         if (start == end)
7.         {
8.                 tree[treenode] = arr[start];
```

```
9.                      return;
10.         }
11.         int mid = (start + end) / 2;
12.         buildtree(arr, tree, start, mid, 2 * treenode);
13.         buildtree(arr, tree, mid + 1, end, 2 * treenode + 1);
14.         tree[treenode] = min(tree[2 * treenode] , tree[2 * treenode + 1]);
15. }
16. void updatetree(int* arr, int* tree, int start, int end, int treenode, int index, int value)
17. {
18.         if (start == end)
19.         {
20.                 arr[index] = value;
21.                 tree[treenode] = value;
22.                 return;
23.         }
24.         int mid = (start + end) / 2;
25.         if (index > mid)//right
26.         {
27.                 updatetree(arr, tree, mid + 1, end, 2 * treenode + 1, index, value);
28.         }
29.         else//left
30.         {
31.                 updatetree(arr, tree, start, mid, 2 * treenode, index, value);
32.         }
33.         tree[treenode] = min(tree[2 * treenode] , tree[2 * treenode + 1]);
34. }
35. int query(int* tree, int start, int end, int treenode, int left, int right)
36. {
37.         if(start>right || end <left)
38.         {
39.                 return INT_MAX;
40.
41.         }
42.
43.    if(start>=left && end<=right)
44.    {
45.        return tree[treenode];
46.    }
47.         int mid=(start+end)/2;
48.         int ans1=query(tree, start, mid, 2*treenode, left, right);
49.         int ans2=query(tree, mid+1, end, 2*treenode+1, left, right);
50.         return min(ans1, ans2);
51. }
52. int main()
```

```
{
        int n, q;
        cin>>n>>q;
        int *arr=new int [n];
        for(int i=0; i<n; i++)
        {
                cin>>arr[i];
        }
        int *tree=new int [4*n];
        buildtree(arr, tree, 0, n - 1, 1);
        //at this point of time i have my tree with me.
        while(q--)
        {
                char query_type;
    int l,r,x,y;

                cin>>query_type>>l>>r;

                if(query_type=='q')
                {
                        cout<<query(tree, 0, n-1, 1, l-1, r-1)<<endl;
                }
                else
                {
                        x=l;
        y=r;
                        //we need to update arr[x]=y;
                        updatetree(arr, tree, 0, n - 1, 1, x-1, y);
                }
        }
}
```

## 2-Tut : Maximum Pair Sum

Send Feedback

You are given a sequence A[1], A[2], ..., A[N], ( 0 ≤ A[i] ≤ 10^8 , 2 ≤ N ≤ 10^5 ). There are two types of operations and they are defined as follows:

**Update**:

This will be indicated in the input of a 'U' followed by space and then two integers i and x.

```
U i x
```

This operation sets the value of A[i] to x.

**Query**:

This will be indicated in the input of a 'Q' followed by a single space and then two integers x and y.

Q x y
You must find two integers i and j such that x ≤ i, j ≤ y and i != j, such that the sum A[i]+A[j] is maximized.

Print the sum A[i]+A[j].

### Input Format:

The first line of input contains an integer N, representing the length of the sequence.
The second line of input contains of N space separated integers, A[i].
The third line of input contains an integer Q, Q ≤ 10^5, representing the number of operations.
Next Q lines contain the operations.

### Constraints:

1 ≤ i ≤ N
0 ≤ x ≤ 10^8
1 ≤ x < y ≤ N

### Output Format:

For each query, print the maximum sum mentioned above, in a new line.

### Sample Input 1:

```
5
1 2 3 4 5
6
Q 2 4
Q 2 5
U 1 6
Q 1 5
U 1 7
Q 1 5
```

### Sample Output 1:

```
7
9
11

12
```

```
#include<bits/stdc++.h>
using namespace std;

pair<int,int> query(pair<int,int>* tree, int start, int end, int treeNode, int left, int right){

        //Completely out
        if (left>end || right<start)
        {
                pair<int,int> temp = make_pair(INT_MIN, INT_MIN);
                return temp;
        }

        //Completely inside
        if (start>=left && end<=right)
```

```
15.         {
16.                 return tree[treeNode];
17.         }
18.
19.         //Partially inside
20.         int mid = (start+end)/2;
21.
22.         pair<int,int> l = query(tree, start, mid, 2*treeNode+1, left, right);
23.         pair<int,int> r = query(tree, mid+1, end, 2*treeNode+2, left, right);
24.
25.
26.         pair<int,int> result;
27.         result.first = max(l.first,r.first);
28.    result.second = min(max(l.first,r.second),max(l.second,r.first));
29.         return result;
30.
31. }
32.
33.
34. void update(int* arr, pair<int,int>* tree, int start, int end, int treeNode, int idx, int value){
35.         int mid = (start+end)/2;
36.
37.         if (start == end)
38.         {
39.                 arr[idx] = value;
40.                 tree[treeNode] = make_pair(value, INT_MIN);
41.                 return;
42.         }
43.
44.         if (idx<=mid)
45.         {
46.                 update(arr, tree, start, mid, 2*treeNode+1, idx, value);
47.         }else{
48.
49.                 update(arr, tree, mid+1, end, 2*treeNode+2, idx, value);
50.         }
51.
52.         pair<int,int> left = tree[2*treeNode+1];
53.         pair<int,int> right = tree[2*treeNode+2];
54.
55.         tree[treeNode].first = max(left.first,right.first);
56.         tree[treeNode].second =
    min(max(left.first,right.second),max(left.second,right.first));
57.         return;
```

```
58. }
59.
60.
61. void create(int* arr, pair<int,int>* tree, int start, int end, int treeNode){
62.         if (end == start)
63.         {
64.                 tree[treeNode] = make_pair(arr[start], INT_MIN);
65.                 return;
66.         }
67.
68.         int mid = (start+end)/2;
69.
70.         create(arr, tree, start, mid, 2*treeNode+1);
71.         create(arr, tree, mid+1, end, 2*treeNode+2);
72.
73.         pair<int, int> left = tree[2*treeNode+1];
74.         pair<int, int> right = tree[2*treeNode+2];
75.
76.         tree[treeNode].first = max(left.first,right.first);
77.         tree[treeNode].second =
    min(max(left.first,right.second),max(left.second,right.first));
78.
79.         return;
80.
81.
82. }
83.
84. int main()
85. {
86.         int n, q;
87.         cin>>n;
88.
89.         int* arr = new int[n];
90.         for (int i = 0; i < n; ++i)
91.         {
92.                 cin>>arr[i];
93.         }
94.
95.         cin>>q;
96.         pair<int,int>* tree = new pair<int,int>[4*n];
97.         create(arr, tree, 0, n-1, 0);
98.         while(q--){
99.                 char a;
100.                int b, c;
```

```
            cin>>a>>b>>c;

            if (a == 'Q')
            {
                pair<int,int> result = query(tree, 0, n-1, 0, b-1, c-1);
                cout << result.first+result.second << endl;
            }else{
                update(arr, tree, 0, n-1, 0, b-1, c);
            }

        }
        return 0 ;
    }
```

## 3-Tut : Maximum Sum In Subarray

Send Feedback

You are given a sequence A[1], A[2], ..., A[N].

A **query** is defined as follows:

Query(x,y) = Max { a[i]+a[i+1]+...+a[j] ; x ≤ i ≤ j ≤ y }.
Given M queries, write a program that outputs the results of these queries.

### Input Format:

The first line of input contains an integer N.
In the second line contains N space separated integers.
The third line contains the integer M.
Next M lines contains 2 integers x and y.

### Output Format:

For each query, print the answer in a new line.

### Constraints:

1 <= N <= 10^5
1 < = Q <= 10^5
-10^4 <= arr[i] <= 10^4

### Sample Input 1:

3
-1 2 3
1
1 2

### Sample Output 1:

2

```
1. #include<bits/stdc++.h>
2. using namespace std;
3. struct attrs {
```

```
4.          int max_sum;
5.          int sum;
6.          int best_prefix_sum;
7.          int best_suffix_sum;
8.  };
9.  void build_tree(int* arr, attrs* tree, int start, int end, int treenode)
10. {
11.         if (start == end)
12.         {
13.                 tree[treenode].sum = arr[start];
14.                 tree[treenode].max_sum = arr[start];
15.                 tree[treenode].best_suffix_sum = arr[start];
16.                 tree[treenode].best_prefix_sum = arr[start];
17.                 return;
18.         }
19.         int mid = (start + end) / 2;
20.         build_tree(arr, tree, start, mid, 2 * treenode);
21.         build_tree(arr, tree, mid + 1, end, 2 * treenode + 1);
22.         tree[treenode].sum = tree[2 * treenode].sum + tree[2 * treenode + 1].sum;
23.         tree[treenode].best_prefix_sum = max(tree[2 * treenode].best_prefix_sum, tree[2
    * treenode].sum + tree[2 * treenode + 1].best_prefix_sum);
24.         tree[treenode].best_suffix_sum = max(tree[2 * treenode + 1].best_suffix_sum,
    tree[2 * treenode + 1].sum + tree[2 * treenode].best_suffix_sum);
25.         tree[treenode].max_sum = max
26.         (
27.                 tree[2 * treenode].max_sum,
28.                 max
29.                 (
30.                         tree[2 * treenode + 1].max_sum,
31.                         max
32.                         (
33.                                 tree[2 * treenode].sum + tree[2 * treenode +
    1].best_prefix_sum,
34.                                 max
35.                                 (
36.                                         tree[2 * treenode + 1].sum + tree[2 *
    treenode].best_suffix_sum,
37.                                         tree[2 * treenode].best_suffix_sum + tree[2 *
    treenode + 1].best_prefix_sum
38.                                 )
39.                         )
40.                 )
41.         );
42. }
```

```
attrs query(attrs* tree, int start, int end, int treenode, int left, int right)
{
	//completely outside
	if (start > right || end < left)
	{
		return {-100000, -100000, -100000, -100000};
	//not used int min here because during recursion it will become +ve if any number is subtracted from it.
	}
	//completely inside
	if (start >= left && end <= right)
	{
		return tree[treenode];
	}
	//partially outside and partially inside
	int mid = (start + end) / 2;
	attrs q1 = query(tree, start, mid, 2 * treenode, left, right);
	attrs q2 = query(tree, mid + 1, end, 2 * treenode + 1, left, right);
	attrs ans;
	ans.sum = q1.sum + q2.sum;
	ans.best_prefix_sum = max(q1.best_prefix_sum, q1.sum + q2.best_prefix_sum);
	ans.best_suffix_sum = max(q1.best_suffix_sum + q2.sum, q2.best_suffix_sum);
	ans.max_sum = max
	(
		q1.max_sum,
		max
		(
			q2.max_sum,
			max
			(
				q1.sum + q2.best_prefix_sum,
				max
				(
					q2.sum + q1.best_suffix_sum,
					q1.best_suffix_sum + q2.best_prefix_sum
				)
			)
		)
	);
	return ans;
}
int main()
{
	int n;
```

```
        cin >> n;
        int* arr = new int[n];
        attrs* tree = new attrs[4 * n];
        for (int i = 0; i < n; i++)
        {
                cin >> arr[i];
        }
        build_tree(arr, tree, 0, n - 1, 1);
        int m;
        cin >> m;
        while (m--)
        {
                int xi, yi;
                cin >> xi >> yi;
                cout << query(tree, 0, n - 1, 1, xi-1, yi-1).max_sum << endl;

        }
    }
```

## 4-Tut : Sum Of Squares

Send Feedback

Segment trees are extremely useful. In particular "Lazy Propagation" (i.e. see here, for example) allows one to compute sums over a range in O(lg(n)), and update ranges in O(lg(n)) as well. In this problem you will compute something much harder:

The sum of squares over a range with range updates of 2 types:

1) increment in a range

2) set all numbers the same in a range.

There will be T test cases in the input file. First line of the input contains two positive integers, N and Q. The next line contains N integers, each at most 1000. Each of the next Q lines starts with a number, which indicates the type of operation:

2 st nd -- return the sum of the squares of the numbers with indices in [st, nd] {i.e., from st to nd inclusive} (1 <= st <= nd <= N).

1 st nd x -- add "x" to all numbers with indices in [st, nd] (1 <= st <= nd <= N, and 1 <= x <= 1,000).

0 st nd x -- set all numbers with indices in [st, nd] to "x" (1 <= st <= nd <= N, and 1 <= x <= 1,000).

### Input Format:

First line of input will contain T(number of test case), each test case follows as.
Line 1: contain two space-separated integers denoting the value of N and Q respectively
Line 2: contain N space-separated integers denoting th value of array elements
Next Q line contain the space separated value for queries as defined as above.

**Constraints:**

1 <= T <= 10
1 <= N <= 10000
1 <= arr[i] <= 1000
1 <= Q <= 10000

**Output Format:**

For each test case, print the answer in new line for query of type 2

**Sample Input 1:**

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

**Sample Output 1:**

```
30
7
13

1
```

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

int t;
int n;

ll tree[400000];
ll arr[100000];

void build(int node, int start,  int end){
    if(start > end)
        return;
    if(start==end){
        //Leaf node.
        tree[node] = arr[start]*arr[start];
        return;
    }
    //Recurse on the left child.
```

```
    int mid = (start+end)>>1;
    int left = node<<1, right = left+1;
    build(left, start, mid);
    //Recurse on the right child.
    build(right, mid+1, end);
    //Internal node will have the sum of both its children.
    tree[node] = tree[left] + tree[right];
}


//Time Complexity O(log(n))
ll query(int node, int start, int end, int l, int r){
    if(start > end || start > r || end < l)
        return 0;
        //complete overlap
    if(l <= start && end <= r)
        return tree[node];
    //partial overlap
    int mid = (start+end)/2;
    int left = node<<1, right = left+1;
        return query(left, start, mid, l, r) + query(right, mid+1, end, l, r);
}

//Worst case complexity is O(n)
void update1(int node, int start, int end, int l, int r, int value){
    //increase values from l to r by value
    //no overlap of the l to r segment on the current segment
    if(start > end || start > r || end < l)
        return;
    if(start==end){
        //Leaf node
        arr[start]+=value;
        tree[node] = arr[start]*arr[start];
        return;
    }
    int mid = (start+end)/2;
    int left = node<<1, right = left+1;
    update1(left, start, mid, l, r, value);
    update1(right, mid+1, end, l, r, value);
    tree[node] = tree[left] + tree[right];
}

void update2(int node, int start, int end, int l, int r, int value){
    //increase values from l to r by value
```

```
    //no overlap of the l to r segment on the current segment
    if(start > end || start > r || end < l)
        return;
    if(start==end){
        //Leaf node
        arr[start] = value;
        tree[node] = arr[start]*arr[start];
        return;
    }
    int mid = (start+end)/2;
    int left = node<<1, right = left+1;
    update2(left, start, mid, l, r, value);
    update2(right, mid+1, end, l, r, value);
    tree[node] = tree[left] + tree[right];
}

int main(){

    cin >> t;
    while(t--){
        memset(arr, 0, sizeof(arr));
        memset(tree, 0, sizeof(tree));
        cin >> n;
        int q;
        cin >> q;
        for(int i = 0;i < n; i++){
            cin >> arr[i];
        }
        build(1, 0, n-1);
        int flag = 0;
        while(q--){
            int type;
            cin >> type;
            if(type == 0){
                int st, end, x;
                    cin >> st >> end >> x;
                    update2(1, 0, n-1, st-1, end-1, x);
                } else if(type == 1){
                    int st, end, x;
                    cin >> st >> end >> x;
                    update1(1, 0, n-1, st-1, end-1, x);
                } else {
                    int st, end;
                    cin >> st >> end;
```

```
                if(flag == 0){
                    flag = 1;
                }
                cout << query(1, 0, n-1, st-1, end-1) << endl;
            }
        }
    }
    return 0;
}
```

## 5-Ass : Maximum Query

Send Feedback

You are given an array of integet of size N and Q queries in form of (l, r). You are supposed to find the maximum value of array between index l and r (both inclusive)

### Input Format:

First line of input contain an integer N (number of elements in the array)
Second line contain N space-separated integers denoting the elements of the array
Third line contain an integer Q (number of queries to be processed)
Next Q line contain two space-separated integers denoting l and r.

### Output Format:

For each test case print the output in newline.

### Constraints:

1 <= N <= 10^4
1 <= Q <= 10^6
1 <= arr[i] <= 10^9
0 <= l <= r < N

### Sample Input:

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

### Sample Output:

```
2

5
```

```
#include<bits/stdc++.h>
using namespace std;

int query(int* tree, int start, int end, int treeNode, int left, int right){

        //Completely out
        if (left>end || right<start)
        {
```

```
9.                      int temp = INT_MIN;
10.                     return temp;
11.             }
12.
13.             //Completely inside
14.             if (start>=left && end<=right)
15.             {
16.                     return tree[treeNode];
17.             }
18.
19.             //Partially inside
20.             int mid = (start+end)/2;
21.
22.             int l = query(tree, start, mid, 2*treeNode+1, left, right);
23.             int r = query(tree, mid+1, end, 2*treeNode+2, left, right);
24.
25.
26.             int result;
27.             result = max(l,r);
28.
29.             return result;
30.
31. }
32.
33. void create(int* arr, int* tree, int start, int end, int treeNode){
34.
35.             if (end == start)
36.             {
37.                     tree[treeNode] = arr[start];
38.                     return;
39.             }
40.
41.             int mid = (start+end)/2;
42.
43.             create(arr, tree, start, mid, 2*treeNode+1);
44.             create(arr, tree, mid+1, end, 2*treeNode+2);
45.
46.             int left = tree[2*treeNode+1];
47.             int right = tree[2*treeNode+2];
48.
49.             tree[treeNode] = max(left,right);
50.
51.             return;
52.
```

```
}

int main()
{
        int n, q;
        cin>>n;

        int* arr = new int[n];
        for (int i = 0; i < n; ++i)
        {
                cin>>arr[i];
        }

        cin>>q;
        int* tree = new int[4*n];
        create(arr, tree, 0, n-1, 0);

        while(q--){

                int b, c;
                cin>>b>>c;
                int result = query(tree, 0, n-1, 0, b, c);

    cout << result << endl;

        }

        return 0 ;

}
```

## 6-Ass : Counting Even/Odd

Send Feedback

Tanmay and Rohit are best buddies. One day Tanmay gives Rohit a problem to test his intelligence and skills. He gives him an array of N natural numbers and asks him to solve the following queries:-

**Query 0**:

`0 x y`

This operation modifies the element present at index i to x.

**Query 1**:

`1 x y`

This operation counts the number of even numbers in range l to r inclusive.

**Query 2**:

2 x y
This operation counts the number of odd numbers in range l to r inclusive.

## Input Format:

First line of the input contains the number N.
Next line contains N natural numbers.
Next line contains an integer Q followed by Q queries.
0 x y - modify the number at index x to y.
1 x y - count the number of even numbers in range l to r inclusive.
2 x y - count the number of odd numbers in range l to r inclusive.

## Constraints:

1<=N,Q<=10^5
1<=l<=r<=N
0<=Ai<=10^9
1<=x<=N
0<=y<=10^9

## Output Format:

For each query, print the answer in a new line.

## Note: Indexing starts from 1

## Sample Input 1:

```
6
1 2 3 4 5 6
4
1 2 5
2 1 4
0 5 4
1 1 6
```

## Sample Output 1:

```
2
2

4
```

```
#include<bits/stdc++.h>
using namespace std;
struct tree_attrs
{
        int even;
        int odd;
};
void build_tree(int* arr, tree_attrs* tree, int start, int end, int treenode)
{
        if (start == end)
        {
```

```
12.                     if (arr[start] % 2 == 0)
13.                     {
14.                             tree[treenode].even = 1;
15.                             tree[treenode].odd = 0;
16.                     }
17.                     else
18.                     {
19.                             tree[treenode].odd = 1;
20.                             tree[treenode].even = 0;
21.                     }
22.                     return;
23.             }
24.             int mid = (start + end) / 2;
25.             build_tree(arr, tree, start, mid, 2 * treenode);
26.             build_tree(arr, tree, mid+1, end, 2 * treenode+1);
27.             tree[treenode].odd = tree[2 * treenode].odd + tree[2 * treenode + 1].odd;
28.             tree[treenode].even = tree[2 * treenode].even + tree[2 * treenode + 1].even;
29. }
30. void update_modify(int *arr, tree_attrs* tree, int start, int end, int treenode, int index, int
    value)
31. {
32.             if (start == end)
33.             {
34.                     arr[index] = value;
35.                     if (value % 2 == 0)
36.                     {
37.                             tree[treenode].even = 1;
38.                             tree[treenode].odd = 0;
39.                     }
40.                     else
41.                     {
42.                             tree[treenode].even = 0;
43.                             tree[treenode].odd = 1;
44.                     }
45.                     return;
46.             }
47.             int mid = (start + end) / 2;
48.             if (index > mid)//right
49.             {
50.                     update_modify(arr, tree, mid + 1, end, 2 * treenode + 1, index, value);
51.             }
52.             else//left
53.             {
54.                     update_modify(arr, tree, start, mid, 2 * treenode, index, value);
```

```
55.         }
56.         tree[treenode].even = tree[2 * treenode].even + tree[2 * treenode + 1].even;
57.         tree[treenode].odd = tree[2 * treenode].odd + tree[2 * treenode + 1].odd;
58. }
59. tree_attrs query(tree_attrs* tree, int start, int end, int treenode, int left, int right)
60. {
61.         //completely outside
62.         if (start > right || end < left)
63.         {
64.                 tree_attrs ans;
65.                 ans.even = 0;
66.                 ans.odd = 0;
67.                 return ans;
68.         }
69.         //complete overlap
70.         if (start >= left && end <= right)
71.         {
72.                 return tree[treenode];
73.         }
74.         //partial overlap
75.         int mid = (start + end) / 2;
76.         tree_attrs left_child = query(tree, start, mid, 2 * treenode, left, right);
77.         tree_attrs right_child = query(tree, mid + 1, end, 2 * treenode + 1, left, right);
78.         tree_attrs ans;
79.         ans.even = left_child.even + right_child.even;
80.         ans.odd = left_child.odd + right_child.odd;
81.         return ans;
82. }
83. int main()
84. {
85.         int n;
86.         cin >> n;
87.         int* arr = new int[n];
88.         for (int i = 0; i < n; i++)
89.         {
90.                 cin >> arr[i];
91.         }
92.         tree_attrs* tree = new tree_attrs[4 * n]();
93.         build_tree(arr, tree, 0, n - 1, 1);
94.         int q;
95.         cin >> q;
96.         while (q--)
97.         {
98.                 int query_type;
```

```
99.                 cin >> query_type;
100.                if (query_type == 0)
101.                {
102.                        int index, value;
103.                        cin >> index >> value;
104.                        update_modify(arr, tree, 0, n - 1, 1, index-1, value);
105.                }
106.                else if (query_type == 1)
107.                {
108.                        int left, right;
109.                        cin >> left >> right;
110.                        cout << query(tree, 0, n - 1, 1, left - 1, right - 1).even<<endl;
111.                }
112.                else
113.                {
114.                        int left, right;
115.                        cin >> left >> right;
116.                        cout << query(tree, 0, n - 1, 1, left - 1, right - 1).odd<<endl;
117.                }
118.        }
119.    }
```

## 7-Ass : This is Sparta!

Send Feedback

King Leonidas of Sparta is preparing his men and country for a war against the Persian King Xerxes. He has N soldiers with him and he has arranged them in a line at The Hot Gates. Let us number them from 1 to N. Leonidas will fight Xerxes' army for Q days, and each day he can send only one of his men to fight. For each warrior, we know 2 traits: Strength and Cowardice. These are given to us in a form of integer. Each day, Leonidas can choose his warrior from a range Li to Ri, and he will choose the warrior with maximum Strength value. If there is more than one warrior having the same maximum Strength value, he will choose the warrior with minimum Cowardice value. If there is still more than 1 warrior with the same maximum Strength value and same minimum Cowardice value, he chooses the one with lower index in line.

King Leonidas is ready to lay his life for Sparta. You, his right hand man, have to help him save Sparta by helping him choose a warrior for each day.

### Input Format:

First line contains a single integer N, denoting the number of warriors Leonidas has.
Second line contains N space separated integers, representing Strength of ith warrior.
Third line contains N space separated integers, representing Cowardice of ith warrior
Next line contains a single integer Q, denoting the number of days Queen Vasya chooses a warrior.
Each of the next Q lines contains 2 integers Li and Ri.

**Constraints:**

1 ≤ N,Q ≤ 10^5
1 ≤ Ai,Bi ≤ 10^9
1 ≤ Li ≤ Ri

**Output Format:**

For each Li and Ri, print the index of the warrior that King Leonidas should choose.

**Sample Input 1:**

```
5
1 8 4 6 8
4 8 6 3 7
4
1 4
2 4
3 4
1 5
```

**Sample Output 1:**

```
2
2
4

5
```

```
#include<bits/stdc++.h>
using namespace std;
struct warriors
{
        int a;
        int b;
};
struct tree_attrs
{
        int a;
        int b;
        int index;
};
void build_tree(warriors* arr, tree_attrs* tree, int start, int end, int treenode)
{
        if (start == end)
        {
                tree[treenode].a = arr[start].a;
                tree[treenode].b = arr[start].b;
                tree[treenode].index = start;
                return;
        }
        int mid = (start + end) / 2;
        build_tree(arr, tree, start, mid, 2 * treenode);
```

```
25.         build_tree(arr, tree, mid + 1, end, 2 * treenode + 1);
26.         tree_attrs left_child = tree[2 * treenode];
27.         tree_attrs right_child = tree[2 * treenode + 1];
28.         if (left_child.a > right_child.a)
29.         {
30.                 tree[treenode].a = left_child.a;
31.                 tree[treenode].b = left_child.b;
32.                 tree[treenode].index = left_child.index;
33.         }
34.         else if(left_child.a < right_child.a)
35.         {
36.                 tree[treenode].a = right_child.a;
37.                 tree[treenode].b = right_child.b;
38.                 tree[treenode].index = right_child.index;
39.         }
40.         else
41.         {
42.                 if (left_child.b < right_child.b)
43.                 {
44.                         tree[treenode].a = left_child.a;
45.                         tree[treenode].b = left_child.b;
46.                         tree[treenode].index = left_child.index;
47.                 }
48.                 else if (left_child.b > right_child.b)
49.                 {
50.                         tree[treenode].a = right_child.a;
51.                         tree[treenode].b = right_child.b;
52.                         tree[treenode].index = right_child.index;
53.                 }
54.                 else
55.                 {
56.                         if (left_child.index < right_child.index)
57.                         {
58.                                 tree[treenode].a = left_child.a;
59.                                 tree[treenode].b = left_child.b;
60.                                 tree[treenode].index = left_child.index;
61.                         }
62.                         else
63.                         {
64.                                 tree[treenode].a = right_child.a;
65.                                 tree[treenode].b = right_child.b;
66.                                 tree[treenode].index = right_child.index;
67.                         }
68.                 }
```

```
69.         }
70. }
71. tree_attrs query(tree_attrs* tree, int start, int end, int treenode, int left, int right)
72. {
73.         //completely outside
74.         if (start > right || end < left)
75.         {
76.                 tree_attrs ans;
77.                 ans.a = INT_MIN;
78.                 ans.b = INT_MAX;
79.                 ans.index = INT_MAX;
80.                 return ans;
81.         }
82.         //completely inside
83.         if (start >= left && end <= right)
84.         {
85.                 return tree[treenode];
86.         }
87.         //partial overlap
88.         int mid = (start + end) / 2;
89.         tree_attrs left_child = query(tree, start, mid, 2 * treenode, left, right);
90.         tree_attrs right_child = query(tree, mid + 1, end, 2 * treenode + 1, left, right);
91.         if (left_child.a > right_child.a)
92.         {
93.                 return left_child;
94.         }
95.         else if (left_child.a < right_child.a)
96.         {
97.                 return right_child;
98.         }
99.         else
100.        {
101.                if (left_child.b < right_child.b)
102.                {
103.                        return left_child;
104.                }
105.                else if (left_child.b > right_child.b)
106.                {
107.                        return right_child;
108.                }
109.                else
110.                {
111.                        if (left_child.index < right_child.index)
112.                        {
```

```
                return left_child;
            }
            else
            {
                return right_child;
            }
        }
    }
}
int main()
{
    int n;
    cin >> n;
    warriors* arr = new warriors[n];
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i].a;
    }
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i].b;
    }
    tree_attrs* tree = new tree_attrs[4 * n]();
    build_tree(arr, tree, 0, n - 1, 1);
    int q;
    cin >> q;
    while (q--)
    {
        int left, right;
        cin >> left >> right;
        cout << query(tree, 0, n - 1, 1, left - 1, right - 1).index+1 << endl;
      // I have added 1 because indexing starts from 1
    }
}
```

## 8-Ass : 2 vs 3

Send Feedback

The fight for the best number in the globe is going to finally come to an end.The top two contenders for the best number are number 2 and number 3.It's the final the entire world was waiting for. Expectorates from all across the globe came to witness the breath taking finals.

The finals began in an astonishing way.A common problem was set for both of them which included both these number.The problem goes like this.

Given a binary string (that is a string consisting of only 0 and 1). They were supposed to perform two types of query on the string.

Type 0: Given two indices l and r.Print the value of the binary string from l to r modulo 3.

Type 1: Given an index l flip the value of that index if and only if the value at that index is 0.

The problem proved to be a really tough one for both of them.Hours passed by but neither of them could solve the problem.So both of them wants you to solve this problem and then you get the right to choose the best number in the globe.

### Input format:

The first line contains N denoting the length of the binary string .
The second line contains the N length binary string.Third line contains the integer Q indicating the number of queries to perform.
This is followed up by Q lines where each line contains a query.

### Output format:

For each query of Type 0 print the value modulo 3.

### Constraints:

1<= N <=10^5
1<= Q <= 10^5
0 <= l <= r < N

### Sample Input

```
5
10010
6
0 2 4
0 2 3
1 1
0 0 4
1 1
0 0 3
```

### Sample Output

2
1
2

1

```
#include<bits/stdc++.h>
using namespace std;
int power[100001];

void buildPower(){
    power[0] = 1;
    for(int i = 1; i < 100001; i++) power[i] = (power[i-1]*2)%3;
}
void build_tree(int* arr, int* tree, int start, int end, int treenode)
{
        if (start == end)
        {
                tree[treenode] = arr[start];
                return;
        }
        int mid = (start + end) / 2;
        build_tree(arr, tree, start, mid, 2 * treenode);
        build_tree(arr, tree, mid + 1, end, 2 * treenode + 1);
        int right_child = tree[2 * treenode + 1];
        int left_child = tree[2 * treenode];
        tree[treenode] = (power[end-mid]*left_child + right_child)%3;
}

void update_flip(int *arr, int* tree, int start, int end, int treenode, int index)
{
        if (start == end)
        {
                arr[index]=1;
      tree[treenode]=1;
      return;
   }
        int mid = (start + end) / 2;
        if (index > mid)
        {
                update_flip(arr, tree, mid + 1, end, 2 * treenode + 1, index);
        }
        else
        {
                update_flip(arr, tree, start, mid, 2 * treenode, index);
```

```
        }

        int right_child = tree[2 * treenode + 1];
        int left_child = tree[2 * treenode];
        tree[treenode] = (power[end-mid]*left_child + right_child)%3;
}

int query_value(int* tree, int start, int end, int treenode, int left, int right)
{
        //completely outside
        if (start > right || end < left)
        {
                return 0;
        }
        //completely inside
        if (start >= left && end <= right)
        {
                return (tree[treenode]*power[right-end])%3;
        }
        //partial overlap
        int mid = (start + end) / 2;
        int answer_from_left = query_value(tree, start, mid, 2 * treenode, left, right);
        int answer_from_right = query_value(tree, mid + 1, end, 2 * treenode + 1, left,
right);
        return (answer_from_left+answer_from_right)%3;
}
int main()
{
    buildPower();
        int n;
        cin >> n;
        string s;
        cin >> s;
        int* arr = new int[n];
        for (int i = 0; i < n; i++)
        {
                arr[i] = s[i]-'0';
        }
        int* tree = new int[4 * n]();
        build_tree(arr, tree, 0, n - 1, 1);
        int q;
        cin >> q;
        while (q--)
        {
```

```
                int query_type;
                cin >> query_type;
                if (query_type == 0)
                {
                        int left, right;
                        cin >> left >> right;
                        cout << query_value(tree, 0, n - 1, 1, left, right) << endl;;
                }
                else
                {
                        int index;
                        cin >> index;
        if(arr[index]==0)
                            update_flip(arr, tree, 0, n - 1, 1, index);
                }
        }
}
```

## 9-Ass : Legion of Doom

Send Feedback

Lex Luthor's Legion of Doom is a tough organization to get into, even for greatest supervillains. Recently, a spot has opened up because The Mad Hatter has retired. Harley Quinn doesn't want to waste this opportunity, and jumps at the chance of the interview. But she has a PhD in psychology, not in Computer Science. She has kidnapped you and will let you go only if you are able to solve the evil questions of Lex Luthor.

You are given an array of N elements, which are initially all 0. After that you will be given C commands. They are -

0 p q v - you have to add v to all numbers in the range of p to q (inclusive), where p and q are two indexes of the array.

1 p q - output a line containing a single integer which is the sum of all the array elements between p and q (inclusive)

### Input Format:

In the first line you'll be given T, number of test cases.

Each test case will start with N and C. After that you'll be given C commands in the format as mentioned above

### Constraints:

1 <= T <= 10

1 <= N, C <= 10000

1 <= val <= 10^8
1 <= p <= q <= N

**Output Format:**
Print the answers of the queries in new line for each test case.

**Sample Input 1:**
```
1
8 6
0 2 4 26
0 4 8 80
0 4 5 20
1 8 8
0 5 7 14
1 4 8
```

**Sample Output 1:**
```
80

508
```

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;
void update_add(ll* tree, ll* lazy, ll start, ll end, ll treenode, ll left, ll right, ll value)
{
        if (start > end)
        {
                return;
        }
        if (lazy[treenode] != 0)
        {
                tree[treenode] += (end - start + 1) * lazy[treenode];
                if (start != end)
                {
                        lazy[2 * treenode] += lazy[treenode];
                        lazy[2 * treenode + 1] += lazy[treenode];
                }
                lazy[treenode] = 0;
        }
        //completely outside
```

```
21.         if (start > right || end < left)
22.         {
23.                 return;
24.         }
25.         //completely inside
26.         if (start >= left && end <= right)
27.         {
28.                 tree[treenode] += (end - start + 1) * value;
29.                 if (start != end)
30.                 {
31.                         lazy[2 * treenode] += value;
32.                         lazy[2 * treenode + 1] += value;
33.                 }
34.                 return;
35.         }
36.         //partial overlap
37.         ll mid = (start + end) / 2;
38.         update_add(tree, lazy, start, mid, 2 * treenode, left, right, value);
39.         update_add(tree, lazy, mid + 1, end, 2 * treenode + 1, left, right, value);
40.         tree[treenode] = tree[2 * treenode] + tree[2 * treenode + 1];
41.         return;
42. }
43. ll query_sum(ll* tree, ll* lazy, ll start, ll end, ll treenode, ll left, ll right)
44. {
45.         if (start > end)
46.         {
47.                 return 0;
48.         }
49.         if (lazy[treenode] != 0)
50.         {
51.                 tree[treenode] += (end - start + 1) * lazy[treenode];
52.                 if (start != end)
53.                 {
54.                         lazy[2 * treenode] += lazy[treenode];
55.                         lazy[2 * treenode + 1] += lazy[treenode];
56.                 }
57.                 lazy[treenode] = 0;
58.         }
59.         //completely outside
60.         if (start > right || end < left)
61.         {
62.                 return 0;
63.         }
64.         //completely inside
```

```
65.         if (start >= left && end <= right)
66.         {
67.                 return tree[treenode];
68.         }
69.         //partial overlap
70.         ll mid = (start + end) / 2;
71.         ll left_child = query_sum(tree, lazy, start, mid, 2 * treenode, left, right);
72.         ll right_child = query_sum(tree, lazy, mid + 1, end, 2 * treenode + 1, left, right);
73.         return left_child + right_child;
74. }
75. int main()
76. {
77.         ll t;
78.         cin >> t;
79.         while (t--)
80.         {
81.                 ll n, c;
82.                 cin >> n >> c;
83.
84.                 ll* tree = new ll[4 * n]();
85.                 ll* lazy = new ll[4 * n]();
86.                 while (c--)
87.                 {
88.                         ll command_type;
89.                         cin >> command_type;
90.                         if (command_type == 0)
91.                         {
92.                                 ll left, right, value;
93.                                 cin >> left >> right >> value;
94.                                 update_add(tree, lazy, 0, n - 1, 1, left - 1, right - 1, value);
95.                         }
96.                         else
97.                         {
98.                                 ll left, right;
99.                                 cin >> left >> right;
100.                                cout << query_sum(tree, lazy, 0, n - 1, 1, left - 1, right - 1)
    << endl;
101.                        }
102.                }
103.        }
104.    }
```

## 10-Ass : The GCD Dillema

Send Feedback

Dwight is always bragging about how amazing he is at solving complicated problems with much ease. Jim got tired of this and gave him an interesting problem to solve.

Jim gave Dwight a sequence of integers $a_1, a_2, ..., a_n$ and q queries $x_1, x_2, ..., x_q$ on it. For each query $x_i$ Dwight has to count the number of pairs (l, r) such that $1 \le l \le r \le n$ and $GCD(a_l, a_{l+1}, ..., a_r) = x_i$. Dwight is feeling out of his depth here and asked you to be his Secret Assistant to the Regional Manager. Your first task is to help him solve the problem. Are you up to it?

## Input Format:

First line of input contains an integer N, representing the number of elements in the sequence.
Second line contains N space-separated integers denoting the elements of the sequence.
Third line of input contains an integer Q, representing the number of queries.
Next Q line contains an integer X.

## Constraints:

1 < = N <= 10^4
1 <= arr[i] <= 10^9
1 <= Q <= 10^4
1 <= X <= 10^9

## Output Format:

For each query, print the answer in a new line.

## Sample Input:

```
2
8 12
3
8
12
4
```

## Sample Output:

```
1
1

1
```

```
1. #include<bits/stdc++.h>
2. using namespace std;
3.
4. const int LIM = 100003;
5. int A[LIM], N;
6. map<int, long long> countg[2], finalc; // final[i] stores the countg of subarrays with gcd = i
7. map<int, long long>::iterator it;
8.
9. void precompute() {
10.  // at any state i, countg[prev] stores the gcd's of subarrays ending at A[i - 1] with their
     countg,
```

```
11.  // for making countg[curr], we take the gcd of A[i] with all elements in countg[prev] and add the countg
12.  // original solution by Rachit http://codeforces.com/contest/475/submission/8093857
13.  countg[0][A[0]] = 1;
14.  finalc[A[0]] = 1LL;
15.  int g, curr, prev;
16.  for(int i = 1; i < N; i++) {
17.    curr = i & 1, prev = curr ^ 1;
18.    countg[curr].clear();
19.    countg[curr][A[i]] = 1LL;    // 1 for the subarray containing only A[i], rest will come from taking gcd with those till A[i - 1] in countg[prev]
20.    for(it = countg[prev].begin(); it != countg[prev].end(); it++) {
21.      g = __gcd(it->first, A[i]);
22.      countg[curr][g] += it->second;   // every subarray giving it->first till A[i - 1] will give g with A[i]
23.    }
24.    for(it = countg[curr].begin(); it != countg[curr].end(); it++) finalc[it->first] += it->second;
25.  }
26. }
27.
28. int main() {
29.
30.  cin >> N;
31.  for(int i = 0; i < N; i++) cin >> A[i];
32.  precompute();
33.  int Q, x;
34.  long long ans;
35.  cin >> Q;
36.  while(Q--) {
37.    cin >> x;
38.    ans = finalc[x];
39.    cout << ans << "\n";
40.  }
41.
42.  return 0;
43. }
```

## 11-Ass : Sheldon and Trains

Send Feedback

Sheldon always tells people, “When you have only one day to visit Los Angeles, make it a Train Day”. He loves spending time while travelling in trains and considers it a fun activity. Sheldon’s mom has come to visit him and he decides to take her out on a train tour of the city of Pasadena, along with his friend Howard. There are n train stations in the city. Howard knows how irritating Sheldon can be during a train ride. So, to keep him busy, Howard gives Sheldon a problem so interesting that he just cannot do

anything else other than devote his entire mind to solving it. The problem goes like this. At the i-th station it's possible to buy only tickets to stations from i + 1 to ai (inclusive). No tickets are sold at the last station. Let $\rho_{i,j}$ be the minimum number of tickets one needs to buy in order to get from stations i to station j. Sheldon's task is to compute the sum of all values $\rho_{i,j}$ among all pairs $1 \le i < j \le n$. As brilliant as he may be, he asked for your help.

**Input Format:**

First line of input will contain N number of trains
Second line will contain N-1 space-separated integers denoting the values of ai

**Output Format:**

Print the answer as mentioned above

**Constraints:**

2 < = N <= 10^5
i + 1 <=arr[i] <= N

**Sample Input 1:**

7
2 7 5 7 6 7

**Sample Output 1:**

29

```
#include<bits/stdc++.h>
using namespace std;
#define MAXN 111111

struct Node{
    int mmm,idx;
    Node():mmm(0){}
}tree[MAXN<<2];
int x,y,N;
void update(int i,int j,int k){
    if(i==j){
        tree[k].mmm=y;
        tree[k].idx=i;
        return;
    }
    int mid=i+j>>1;
    if(x<=mid) update(i,mid,k<<1);
    else update(mid+1,j,k<<1|1);
    if(tree[k<<1].mmm>tree[k<<1|1].mmm){
        tree[k]=tree[k<<1];
    }else{
        tree[k]=tree[k<<1|1];
    }
```

```
}
Node query(int i,int j,int k){
    if(x<=i && j<=y){
        return tree[k];
    }
    int mid=i+j>>1;
    Node ret;
    if(x<=mid){
        Node tmp=query(i,mid,k<<1);
        if(ret.mmm<tmp.mmm) ret=tmp;
    }
    if(y>mid){
        Node tmp=query(mid+1,j,k<<1|1);
        if(ret.mmm<tmp.mmm) ret=tmp;
    }
    return ret;
}

int a[MAXN];
long long d[MAXN];
int main(){
    int n;
    scanf("%d",&n);
    for(N=1; N<n; N<<=1);
    for(int i=1; i<n; ++i){
        scanf("%d",a+i);
        x=i; y=a[i];
        update(1,N,1);
    }
    x=n; y=n;
    update(1,N,1);
    for(int i=n-1; i>=1; --i){
        x=i+1; y=a[i];
        Node tmp=query(1,N,1);
        d[i]=d[tmp.idx]+n-i-(a[i]-tmp.idx);
    }
    long long res=0;
    for(int i=1; i<=n; ++i){
        res+=d[i];
    }
    printf("%lld",res);
    return 0;
}
```

Bonus problem :

**12-Lazy Propogation :**

```
#include<bits/stdc++.h>
using namespace std;

void buildTree(int* arr,int* tree,int start,int end,int treeNode){

	if(start == end){
		tree[treeNode] = arr[start];
		return;
	}
	int mid = (start+end)/2;

	buildTree(arr,tree,start,mid,2*treeNode);
	buildTree(arr,tree,mid+1,end,2*treeNode+1);
	tree[treeNode] = min(tree[2*treeNode],tree[2*treeNode+1]);
}

void updateSegmentTreeLazy(int* tree,int* lazy,int low,int high,int startR,int endR,int
currPos,int inc){

	if(low > high){
		return;
	}

	if(lazy[currPos] !=0){
		tree[currPos] += lazy[currPos];

		if(low!=high){
			lazy[2*currPos] += lazy[currPos];
			lazy[2*currPos+1] += lazy[currPos];
		}
		lazy[currPos] = 0;
	}

	// No overlap
	if(startR > high || endR < low){
		return;
	}

	// Complete Overlap

	if(startR<= low && high <= endR){
		tree[currPos] += inc;
```

```
42.                     if(low!=high){
43.                             lazy[2*currPos] += inc;
44.                             lazy[2*currPos+1] += inc;
45.                     }
46.                     return;
47.             }
48.
49.             // Partial Overlap
50.
51.             int mid = (low+high)/2;
52.             updateSegmentTreeLazy(tree,lazy,low,mid,startR,endR,2*currPos,inc);
53.             updateSegmentTreeLazy(tree,lazy,mid+1,high,startR,endR,2*currPos+1,inc);
54.             tree[currPos] = min(tree[2*currPos],tree[2*currPos+1]);
55. }
56.
57. int main(){
58.
59.             int arr[] = {1,3,-2,4};
60.             int* tree = new int[12]();
61.             buildTree(arr,tree,0,3,1);
62.             int* lazy = new int[12]();
63.             updateSegmentTreeLazy(tree,lazy,0,3,0,3,1,3);
64.             updateSegmentTreeLazy(tree,lazy,0,3,0,1,1,2);
65.
66.             cout<< "Segment Tree" <<endl;
67.             for(int i=1;i<12;i++){
68.                     cout<<tree[i]<< endl;
69.             }
70.
71.             cout<< "Lazy Tree" <<endl;
72.             for(int i=1;i<12;i++){
73.                     cout<<lazy[i]<< endl;
74.             }
75.
76.             return 0;
77. }
```

# L21 Range Query 2

## 1-Tut : Coder's Rating

Send Feedback

Some of the more elite (and not-so-elite) coders around take part in a certain unnamed programming contest. In said contest, there are multiple types of competitions. Here, we consider the Open and High School competition types. For each type, each competitor receives a rating, an integer between 1 and 100000, inclusive. A coder's rating is based upon his or her level of performance in matches and is calculated using a complicated formula which, thankfully, you will not be asked to implement.

Although the Open and High School ratings for a coder who has participated in both competition types lately are usually close, this is not always the case. In particular, High School matches are more about speed, since many coders are able to solve all the problems, whereas Open matches require more thinking and there is a steeper curve in terms of problem difficulty.

### Problem Statement

You are given N coders (1 ≤ N ≤ 300000), conveniently numbered from 1 to N. Each of these coders participates in both High School and Open matches. For each coder, you are also given an Open rating Ai and a High School rating Hi. Coder i is said to be better than coder j if and only if both of coder i's ratings are greater than or equal to coder j's corresponding ratings, with at least one being greater. For each coder i, determine how many coders coder i is better than.

### Input Format

On the first line of input is a single integer N, as described above. N lines then follow. Line i+1 contains two space-separated integers, Ai and Hi.

### Output Format

Line i should contain the number of coders that coder i is better than.

Sample Input 1:

```
8
1798 1832
862 700
1075 1089
1568 1557
2575 1984
1033 950
1656 1649
1014 1473
```

Sample Output 1:

```
6
0
2
```

4
7
1
5
1
Explanation

1st code is better than 2nd, 3rd, 4th, 5th, 6th and 7th coder.

Hence he is better than 6 coders.

```
#include <bits/stdc++.h>
using namespace std;

class coder
{
public:
    int x, y, index;
};
bool sorter(coder c1, coder c2)
{
    if (c1.x == c2.x)
    {
        return c1.y < c2.y;
    }
    return c1.x < c2.x;
}
void update(int y, int *bit)
{
    for (; y <= 100000; y += y & (-y))
    {
        bit[y]++;
    }
}
int query(int y, int *bit)
{
    int count = 0;
    for (; y > 0; y -= y & (-y))
    {
        count += bit[y];
    }
    return count;
}
int main()
{
```

```
    int n;
    cin >> n;
    coder *arr = new coder[n];
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i].x >> arr[i].y;
        arr[i].index = i;
    }
    sort(arr, arr + n, sorter);
    int *bit = new int[100001];
    int *ans = new int[n];
    for (int i = 0; i < n;)
    {
        int endindex = i;
        while (endindex < n && arr[i].x == arr[endindex].x && arr[i].y == arr[endindex].y)
        {
            endindex++;
        }
        for (int j = i; j < endindex; j++)
        {
            ans[arr[j].index] = query(arr[j].y, bit);
        }
        for (int j = i; j < endindex; j++)
        {
            update(arr[j].y, bit);
        }
        i = endindex;
    }
    for (int i = 0; i < n; i++)
    {
        cout << ans[i] << endl;
    }
    return 0;
}
```

## 2-Tut : Distinct Query Problem

Send Feedback

Given a sequence of n numbers a1, a2, ..., an and a number of d-queries. A d-query is a pair (i, j) (1 ≤ i ≤ j ≤ n). For each d-query (i, j), you have to return the number of distinct elements in the subsequence ai, ai+1, ..., aj.

### Input Format:

Line 1: n (1 ≤ n ≤ 30000).
Line 2: n numbers a1, a2, ..., an (1 ≤ ai ≤ 10^6).

Line 3: q (1 ≤ q ≤ 200000), the number of d-queries.
In the next q lines, each line contains 2 numbers i, j representing a d-query (1 ≤ i ≤ j ≤ n).

**Output Format:**

For each d-query (i, j), print the number of distinct elements in the subsequence ai, ai+1, ..., aj in a single line.

**Sample Input 1:**

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

**Sample Output 1:**

```
3
2

3
```

```
#include <bits/stdc++.h>
using namespace std;

class event
{
public:
    int first, second, index;
};

bool compare(event a, event b)
{

    return a.second < b.second;
}

void update(int index, int value, int n, int *bit)
{
    for (; index <= n; index += (index & (-index)))
    {
        bit[index] += value;
    }
}

int value(int index, int *bit)
{
    int count = 0;
    for (; index > 0; index -= (index & (-index)))
    {
```

```
        count += bit[index];
    }
    return count;
}

int main()
{

    int n;
    cin >> n;

    int *arr = new int[n + 1];
    for (int i = 1; i <= n; i++)
    {
        cin >> arr[i];
    }

    int q;
    cin >> q;

    event *query = new event[q];
    for (int i = 0; i < q; i++)
    {
        cin >> query[i].first >> query[i].second;
        query[i].index = i;
    }

    sort(query, query + q, compare);

    int *bit = new int[n + 1];
    int *ans = new int[q];

    int total = 0;
    int k = 0;
    int *last = new int[1000001];
    for (int i = 0; i <= 1000000; i++)
    {
        last[i] = -1;
    }

    for (int i = 1; i <= n; i++)
    {
        if (last[arr[i]] != -1)
        {
```

```
            update(last[arr[i]], -1, n, bit);
        }
        else
        {
            total++;
        }

        update(i, 1, n, bit);
        last[arr[i]] = i;

        while (k < q && query[k].second == i)
        {
            ans[query[k].index] = total - value(query[k].first - 1, bit);
            k++;
        }
    }

    for (int i = 0; i < q; i++)
    {
        cout << ans[i] << endl;
    }
    return 0;
}
```

## 3-Tut : OrderSet - Problem

Send Feedback

In this problem, you have to maintain a dynamic set of numbers which support the two fundamental operations

INSERT(S,x): if x is not in S, insert x into S
DELETE(S,x): if x is in S, delete x from S
and the two type of queries

K-TH(S) : return the k-th smallest element of S
COUNT(S,x): return the number of elements of S smaller than x

### Input Format:

Line 1: Q , the number of operations
In the next Q lines, the first token of each line is a character I, D, K or C meaning that the corresponding operation is INSERT, DELETE, K-TH or COUNT, respectively, following by a whitespace and an integer which is the parameter for that operation.

### Constraints:

1 <= Q <= 2*10^5
-10^9 <= x <= 10^9

**Output Format:**

For each query, print the corresponding result in a single line. In particular, for the queries K-TH, if k is larger than the number of elements in S, print the word 'invalid'.

**Sample Input 1:**

```
8
I -1
I -1
I 2
C 0
K 2
D -1
K 1
K 2
```

**Sample Output 1:**

```
1
2
2

invalid
```

```
#include <bits/stdc++.h>

using namespace std;

#define MAX 200005
#define ll long long int

pair<int, int> M[MAX], T[MAX];
int BIT[MAX], A[MAX], n;
char C[MAX];
bool b;

int get_count(int idx)
{
    if (idx == 0)
        return 0;
    ll sum = 0;
    while (idx > 0)
    {
        sum += BIT[idx];
        idx -= idx & (-idx);
    }
    return sum;
}
void updateBIT(int idx, int val)
{
```

```
27.     while (idx < MAX)
28.     {
29.
30.         BIT[idx] += val;
31.         idx += idx & (-idx);
32.     }
33. }
34. int b_search(int x)
35. {
36.     int lo = 0, hi = n - 1, mid, ans = -1;
37.     while (lo <= hi)
38.     {
39.         mid = (lo + hi) >> 1;
40.
41.         if (M[mid].first == x)
42.         {
43.             b = 1;
44.             return mid;
45.         }
46.         else if (M[mid].first > x)
47.             hi = mid - 1;
48.         else
49.         {
50.             ans = mid;
51.             lo = mid + 1;
52.         }
53.     }
54.     b = 0;
55.     return ans;
56. }
57. int main()
58. {
59.     int i, Q, x, k = 0, lo, hi, mid;
60.     cin >> Q;
61.     for (i = 0; i < Q; i++)
62.     {
63.         C[i] = getchar();
64.         while (C[i] < 65 || C[i] >= 91)
65.             C[i] = getchar();
66.         cin >> A[i];
67.         if (C[i] == 'I')
68.         {
69.             T[k].first = A[i];
70.             T[k++].second = 0;
```

```
71.         }
72.     }
73.     sort(T, T + k);
74.     M[0] = T[0];
75.     n = 1;
76.     for (i = 1; i < k; i++)
77.     {
78.         if (T[i].first != T[i - 1].first)
79.         {
80.             M[n++] = T[i];
81.         }
82.     }
83.     for (i = 0; i < Q; i++)
84.     {
85.         if (C[i] == 'I')
86.         {
87.             x = b_search(A[i]);
88.
89.             if (M[x].second == 0)
90.             {
91.                 M[x].second = 1;
92.                 updateBIT(x + 1, 1);
93.             }
94.         }
95.         else if (C[i] == 'D')
96.         {
97.             x = b_search(A[i]);
98.             if (x != -1 && M[x].second == 1 && b)
99.             {
100.                 updateBIT(x + 1, -1);
101.                 M[x].second = 0;
102.             }
103.          }
104.          else if (C[i] == 'C')
105.          {
106.              x = b_search(A[i]);
107.
108.              if (b)
109.                  cout << get_count(x + 1 - 1) << endl;
110.              else if (x != -1)
111.                  cout << get_count(x + 1) << endl;
112.              else
113.                  cout << 0 << endl;
114.          }
```

```
        else if (C[i] == 'K')
        {
            k = A[i];
            x = -1;
            bool mno = 0;
            lo = 1;
            hi = MAX - 1;
            while (lo <= hi)
            {
                mid = (lo + hi) >> 1;
                if (get_count(mid) == k && get_count(mid - 1) != k)
                {
                    mno = 1;
                    x = mid;
                    break;
                }
                else if (get_count(mid) < k)
                {
                    x = mid;
                    lo = mid + 1;
                }
                else
                    hi = mid - 1;
            }

            if (!mno)
                cout << "invalid" << endl;
            else
                cout << M[x - 1].first << endl;
        }
    }
    return 0;
}
```

## 4-Ass : KQUERY

Send Feedback

Given a sequence of n numbers a1, a2, ..., an and a number of k- queries. A k-query is a triple (i, j, k) (1 ≤ i ≤ j ≤ n). For each k-query (i, j, k), you have to return the number of elements greater than k in the subsequence ai, ai+1, ..., aj.

### Input Format

Line 1: Contains an integer N denoting the number of elements in the array
Line 2: N space-separated integers denoting the elements of the array.
Line 3: Number of queries Q

Next Q line contain two space-separated integers i, j, k descrbing the current query

**Constraints:**

1 <= N <= 10^5
1 <= Q <= 10^5
1 <= arr[i] <= 10^9
1 <= i <= j <= N
1 <= k <= 10^9

**Output Format:**

For each k-query (i, j, k), print the number of elements greater than k in the subsequence ai, ai+1, ..., aj in a single line.

**Sample Input 1:**

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

**Sample Output 1:**

```
2
0

3
```

```
#include <bits/stdc++.h>
using namespace std;

vector<pair<int, int>> a;

vector<pair<pair<int, int>, pair<int, int>>> queries;

int ans[100002];
int bit[100002];
int query(int r, int n)
{
    int ans = 0;
    while (r > 0)
    {
        ans += bit[r];
        r -= (r & -r);
    }
    return ans;
}
void update(int i, int n)
{
    while (i <= n)
        bit[i]++, i += (i & -i);
```

```
24. }
25. int main()
26. {
27.     int n;
28.     cin >> n;
29.     int i;
30.     for (i = 1; i <= n; i++)
31.     {
32.         int x;
33.         cin >> x;
34.         a.push_back({x, i});
35.     }
36.     sort(a.begin(), a.end());
37.     int q;
38.     cin >> q;
39.     for (i = 1; i <= q; i++)
40.     {
41.         int l, r, k;
42.         cin >> l >> r >> k;
43.         queries.push_back({{k, i}, {l, r}});
44.     }
45.     sort(queries.begin(), queries.end(), greater<pair<pair<int, int>, pair<int, int>>>());
46.     for (pair<pair<int, int>, pair<int, int>> p : queries)
47.     {
48.         int k = p.first.first;
49.         i = p.first.second;
50.         int l = p.second.first;
51.         int r = p.second.second;
52.         int j = a.size() - 1;
53.         while (j >= 0)
54.         {
55.             if (a[j].first > k)
56.             {
57.                 update(a[j].second, n);
58.                 a.pop_back();
59.             }
60.             else
61.                 break;
62.             j--;
63.         }
64.         int temp = query(r, n) - query(l - 1, n);
65.         ans[i] = temp;
66.     }
67.     for (i = 1; i <= queries.size(); i++)
```

```
68.         cout << ans[i] << "\n";
69.     return 0;
70. }
71.
```

## 5-Ass : Shil and Wave Sequence

Send Feedback

Given a sequence A1 , A2 , A3 .. AN of length N. Find total number of wave subsequences having length greater than 1.

Wave subsequence of sequence A1 , A2 , A3 .. AN is defined as a set of integers i1 , i2 .. ik such that Ai1 < Ai2 > Ai3 < Ai4 .... or Ai1 > Ai2 < Ai3 > Ai4 .... and i1 < i2 < ...< ik.Two subsequences i1 , i2 .. ik and j1 , j2 .. jm are considered different if k != m or there exists some index l such that il ! = jl

### Input Format:

First line of input consists of integer N denoting total length of sequence.Next line consists of N integers A1 , A2 , A3 .. AN .

### Constraints:

1 ≤ N ≤ 10^5
1 ≤ Ai ≤ 10^5

### Output Format:

Output total number of wave subsequences of given sequence. Since answer can be large, output its modulo with 10^9+7

### Sample Input 1:

5
1 3 5 4 2

### Sample Output 1:

17

### Explanation:

All the possible sequences are: [ 1 3 ] , [1 5 ] , [ 1 4 ] , [1 2 ] , [1 3 2] , [1 4 2 ] , [1 5 2] , [1 5 4] , [3 5] , [3 4] , [3 2] , [3 5 2] , [3 4 2] , [3 5 4] , [5 4] , [5 2 ] , [4 2] . Note that value in the bracket are the values from the original sequence whose positions are maintained.

```
1.  #include <bits/stdc++.h>
2.  #define MOD 1000000007
3.  using namespace std;
4.
5.  int main()
6.  {
7.      long long n, x, t, sum1, sum2;
8.      cin >> n;
9.      long long dpl[100001] = {0}, dph[100001] = {0}, a[100001] = {0};
10.     for (long long i = 1; i < n + 1; i++)
11.     {
12.         cin >> x;
```

```
        t = 100000;
        sum1 = 0, sum2 = 0;
        while (t)
        {
            sum1 = (sum1 + dph[t] + a[t]) % MOD;
            t -= (t & (-t));
        }
        t = x;
        while (t)
        {
            sum1 = (sum1 - dph[t] - a[t] + MOD) % MOD;
            t -= (t & (-t));
        }
        t = x - 1;
        while (t)
        {
            sum2 = (sum2 + dpl[t] + a[t]) % MOD;
            t -= (t & (-t));
        }
        t = x;
        while (t < 100001)
        {
            dpl[t] = (dpl[t] + sum1) % MOD;
            dph[t] = (sum2 + dph[t]) % MOD;
            a[t] += 1;
            t += (t & (-t));
        }
    }
    long long ans = 0;
    sum1 = 0;
    sum2 = 0;
    t = 100000;
    while (t)
    {
        ans = (ans + dph[t]) % MOD;
        ans = (ans + dpl[t]) % MOD;
        t -= (t & (-t));
    }
    cout << ans;
}
```

## 6-Ass : INCSEQ

Send Feedback

Given a sequence of N integers S1, ..., SN, compute the number of increasing subsequences of S with length K and that is, the number of K-tuples i1, ..., iK such that 1 ≤ i1 < ... < iK ≤ N and Si1 < ... < SiK.

**Input Format:**

The first line contains the two integers N and K.
Next line contains N space-separated integers denoting the elements of the array.

**Constraints:**

1 <= N <= 10^4
1 <= K <= 50
1 <= arr[i] <= 10^5

**Output Format:**

Print a single integer representing the number of increasing subsequences of S of length K, modulo 5,000,000.

**Sample Input 1:**

4 3
1 2 2 10

**Sample Output 1:**

2

```
#include <bits/stdc++.h>
using namespace std;

#define int long long
int M = 5000000;

int n, k;
int tree[52][300001];

void update(int ind, int val, int i)
{
    while (ind <= n)
    {
        tree[i][ind] = (tree[i][ind] + val) % M;
        ind += ind & (-ind);
    }
}

int query(int ind, int i)
{
    int ans = 0;
    while (ind > 0)
    {
        ans = (tree[i][ind] + ans) % M;
        ind -= ind & (-ind);
```

```
    }
    return ans;
}

void compress(vector<int> &a)
{
    vector<int> b = a;
    sort(b.begin(), b.end());
    unordered_map<int, int> m;
    for (int i = 1, c = 1; i <= n; i++)
    {
        if (m.find(b[i]) == m.end())
            m[b[i]] = c++;
    }
    for (int i = 1; i <= n; i++)
    {
        a[i] = m[a[i]];
    }
}

signed main()
{
    cin >> n >> k;
    if (k == 1)
    {
        cout << n;
        return 0;
    }

    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
    }
    compress(a);

    int ans = 0;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= k; j++)
        {
            int p = (j == 1 ? 1 : query(a[i] - 1, j - 1));
            update(a[i], p, j);
            if (j == k)
```

```
70.                 ans = (ans + p) % M;
71.         }
72.     }
73.     cout << ans;
74.
75.     return 0;
76. }
```

# L22 : Graph1

## 1-Tut : Code : BFS Traversal

Send Feedback

Given an undirected graph G(V, E), print its BFS traversal.

Here you need to consider that you need to print BFS path starting from vertex 0 only.

V is the number of vertices present in graph G and vertices are numbered from 0 to V-1.

E is the number of edges present in graph G.

### Note :

1. Take graph input in the adjacency matrix.
2. Handle for Disconnected Graphs as well

### Input Format :

Line 1: Two Integers V and E (separated by space)

Next 'E' lines, each have two space-separated integers, 'a' and 'b', denoting that there exists an edge between Vertex 'a' and Vertex 'b'.

### Output Format :

BFS Traversal (separated by space)

### Constraints :

2 <= V <= 1000

1 <= E <= 1000

### Sample Input 1:

```
4 4
0 1
0 3
1 2
2 3
```

### Sample Output 1:

```
0 1 3 2
```

```
#include <bits/stdc++.h>
using namespace std;

void printBFS(int **edges, int n, int sv, bool *visited)
{

    queue<int> pendingVertices;
    pendingVertices.push(sv);
    visited[sv] = true;

    while (!pendingVertices.empty())
    {
```

```
13.
14.         int currentVertex = pendingVertices.front();
15.         pendingVertices.pop();
16.         cout << currentVertex << " ";
17.         for (int i = 0; i < n; i++)
18.         {
19.
20.             if (i == currentVertex)
21.             {
22.                 continue;
23.             }
24.
25.             if (edges[currentVertex][i] == 1 && !visited[i])
26.             {
27.                 pendingVertices.push(i);
28.                 visited[i] = true;
29.             }
30.         }
31.     }
32. }
33.
34. void BFS(int **edges, int n)
35. {
36.     bool *visited = new bool[n];
37.     for (int i = 0; i < n; i++)
38.     {
39.         visited[i] = false;
40.     }
41.
42.     for (int i = 0; i < n; i++)
43.     {
44.         if (!visited[i])
45.         {
46.             printBFS(edges, n, i, visited);
47.         }
48.     }
49.     delete[] visited;
50. }
51.
52. int main()
53. {
54.
55.     int n;
56.     int e;
```

```
57.
58.     cin >> n >> e;
59.
60.     int **edges = new int *[n];
61.     for (int i = 0; i < n; i++)
62.     {
63.         edges[i] = new int[n];
64.         for (int j = 0; j < n; j++)
65.         {
66.             edges[i][j] = 0;
67.         }
68.     }
69.
70.     for (int i = 0; i < e; i++)
71.     {
72.         int f, s;
73.         cin >> f >> s;
74.         edges[f][s] = 1;
75.         edges[s][f] = 1;
76.     }
77.
78.     BFS(edges, n);
79.
80.     for (int i = 0; i < n; i++)
81.     {
82.         delete[] edges[i];
83.     }
84.
85.     delete[] edges;
86.
87.     return 0;
88. }
```

## 2-Tut : Code : Has Path

Send Feedback

Given an undirected graph G(V, E) and two vertices v1 and v2(as integers), check if there exists any path between them or not. Print true or false.

V is the number of vertices present in graph G and vertices are numbered from 0 to V-1.

E is the number of edges present in graph G.

### Input Format :

First line will contain T(number of test cases), each test case as follow.
Line 1: Two Integers V and E (separated by space)

Next E lines : Two integers a and b, denoting that there exists an edge between vertex a and vertex b (separated by space)
Line (E+2) : Two integers v1 and v2 (separated by space)

**Output Format :**

true or false for each test case in a newline.

**Constraints :**

1 <= T <= 10
2 <= V <= 1000
1 <= E <= 1000
0 <= v1, v2 <= V-1

**Sample Input 1 :**

```
1
4 4
0 1
0 3
1 2
2 3
1 3
```

**Sample Output 1 :**

true

**Sample Input 2 :**

```
1
6 3
5 3
0 1
3 4
0 3
```

**Sample Output 2 :**

false

```
#include <bits/stdc++.h>
using namespace std;

bool printBFS(int **edges, int n, int sv, bool *visited, int v2)
{

    queue<int> pendingVertices;
    pendingVertices.push(sv);
    visited[sv] = true;

    while (!pendingVertices.empty())
    {

        int currentVertex = pendingVertices.front();
        pendingVertices.pop();
```

```
        if (currentVertex == v2)
        {
            return true;
        }
        for (int i = 0; i < n; i++)
        {

            if (i == currentVertex)
            {
                continue;
            }

            if (edges[currentVertex][i] == 1 && !visited[i])
            {
                pendingVertices.push(i);
                visited[i] = true;
            }
        }
    }

    return false;
}

void BFS(int **edges, int n, int v1, int v2)
{
    bool *visited = new bool[n];
    for (int i = 0; i < n; i++)
    {
        visited[i] = false;
    }

    int ans = printBFS(edges, n, v1, visited, v2);
    if (ans == 1)
    {
        cout << "true" << endl;
    }
    else
    {
        cout << "false" << endl;
    }

    delete[] visited;
}

```

```
60. int main()
61. {
62.
63.     int t;
64.     cin >> t;
65.     while (t--)
66.     {
67.         int n;
68.         int e;
69.
70.         cin >> n >> e;
71.
72.         int **edges = new int *[n];
73.         for (int i = 0; i < n; i++)
74.         {
75.             edges[i] = new int[n];
76.             for (int j = 0; j < n; j++)
77.             {
78.                 edges[i][j] = 0;
79.             }
80.         }
81.
82.         for (int i = 0; i < e; i++)
83.         {
84.             int f, s;
85.             cin >> f >> s;
86.             edges[f][s] = 1;
87.             edges[s][f] = 1;
88.         }
89.
90.         int v1, v2;
91.
92.         cin >> v1 >> v2;
93.
94.         if (edges[v1][v2] == 1)
95.         {
96.             cout << "true" << endl;
97.         }
98.         else
99.         {
100.                BFS(edges, n, v1, v2);
101.            }
102.
103.            for (int i = 0; i < n; i++)
```

```
        {
            delete[] edges[i];
        }

        delete[] edges;
    }

    return 0;
}
```

## 3-Tut : Code : Get Path - DFS

Send Feedback

Given an undirected graph G(V, E) and two vertices v1 and v2(as integers), find and print the path from v1 to v2 (if exists). Print nothing if there is no path between v1 and v2.

Find the path using DFS and print the first path that you encountered.

V is the number of vertices present in graph G and vertices are numbered from 0 to V-1.

E is the number of edges present in graph G.

Print the path in reverse order. That is, print v2 first, then intermediate vertices and v1 at last.

**Note : Save the input graph in Adjacency Matrix.**

**Input Format :**

First line will contain T(number of test case), each test follow as.
Line 1: Two Integers V and E (separated by space)
Next E lines : Two integers a and b, denoting that there exists an edge between vertex a and vertex b (separated by space)
Line (E+2) : Two integers v1 and v2 (separated by space)

**Output Format :**

Path from v1 to v2 in reverse order (separated by space) for each test case in newline.

**Constraints :**

1 <= T <= 10
2 <= V <= 1000
1 <= E <= 1000
0 <= v1, v2 <= V-1

**Sample Input 1 :**

1
4 4

```
0 1
0 3
1 2
2 3
1 3
```

**Sample Output 1 :**

```
3 0 1
```

**Sample Input 2 :**

```
1
6 3
5 3
0 1
3 4
0 3
```

**Sample Output 2 :**

```
#include <bits/stdc++.h>
using namespace std;

vector<int> get_path(int **arr, int v, bool *visited, int current_vertex, int v2)
{
    if (current_vertex == v2)
    {
        vector<int> ans;
        ans.push_back(current_vertex);
        return ans;
    }
    for (int i = 0; i < v; i++)
    {
        if (!visited[i] && i != current_vertex && arr[current_vertex][i] == 1)
        {
            vector<int> ans;
            visited[i] = true;
            ans = get_path(arr, v, visited, i, v2);
            if (ans.size() != 0)
            {
                ans.push_back(current_vertex);
                return ans;
```

```
            }
        }
    }
    vector<int> ans;
    return ans;
}
int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        int v, e;
        cin >> v >> e;
        int **arr = new int *[v];
        for (int i = 0; i < v; i++)
        {
            arr[i] = new int[v];
            for (int j = 0; j < v; j++)
            {
                arr[i][j] = 0;
            }
        }
        bool *visited = new bool[v];
        for (int i = 0; i < v; i++)
        {
            visited[i] = false;
        }
        while (e--)
        {
            int a, b;
            cin >> a >> b;
            arr[a][b] = 1;
            arr[b][a] = 1;
        }
        int v1, v2;
        cin >> v1 >> v2;

        visited[v1] = true;
        vector<int> ans = get_path(arr, v, visited, v1, v2);
        if (ans.size() != 0)
        {
            for (int i = 0; i < ans.size(); i++)
            {
```

```
            cout << ans[i] << " ";
        }
    }
    else
    {
    }
    cout << endl;
  }

  return 0;
}
```

## 4-Tut : Code : Get Path - BFS

Send Feedback

Given an undirected graph G(V, E) and two vertices v1 and v2(as integers), find and print the path from v1 to v2 (if exists). Print nothing if there is no path between v1 and v2.

Find the path using BFS and print the shortest path available.

V is the number of vertices present in graph G and vertices are numbered from 0 to V-1.

E is the number of edges present in graph G.

Print the path in reverse order. That is, print v2 first, then intermediate vertices and v1 at last.

**Note : Save the input graph in Adjacency Matrix.**

**Input Format :**

First line of input will contain T(number of test case), each test case follows as.
Line 1: Two Integers V and E (separated by space)
Next E lines : Two integers a and b, denoting that there exists an edge between vertex a and vertex b (separated by space)
Line (E+2) : Two integers v1 and v2 (separated by space)

**Output Format :**

Path from v1 to v2 in reverse order (separated by space) for each test case in new line.

**Constraints :**

1 <= T <= 10
2 <= V <= 1000
1 <= E <= 1000
0 <= v1, v2 <= V-1

**Sample Input 1 :**

```
1
4 4
0 1
0 3
1 2
2 3
1 3
```

**Sample Output 1 :**

```
3 0 1
```

**Sample Input 2 :**

```
1
6 3
5 3
0 1
3 4
0 3
```

**Sample Output 2 :**

```
#include <bits/stdc++.h>
using namespace std;
void printbfs(int **arr, int v, int current_element, int v2, bool *visited)
{
    queue<int> q;
    visited[current_element] = true;
    q.push(current_element);
    map<int, int> m;
    while (!q.empty())
    {
        int current = q.front();
        if (current == v2)
        {
            break;
        }
        for (int i = 0; i < v; i++)
        {
            if (!visited[i] && arr[current][i] == 1 && i != current)
            {
                q.push(i);
                visited[i] = true;
                m[i] = current;
            }
        }
        q.pop();
        if (q.empty())
        {
            return;
        }
    }
    int i = v2;
    cout << v2 << " ";
    while (i != current_element)
    {
```

```
        cout << m[i] << " ";
        i = m[i];
    }
}
int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        int v, e;
        cin >> v >> e;
        int **arr = new int *[v];
        for (int i = 0; i < v; i++)
        {
            arr[i] = new int[v];
            for (int j = 0; j < v; j++)
            {
                arr[i][j] = 0;
            }
        }
        while (e--)
        {
            int a, b;
            cin >> a >> b;
            arr[a][b] = 1;
            arr[b][a] = 1;
        }
        int v1, v2;
        cin >> v1 >> v2;
        bool *visited = new bool[v];
        for (int i = 0; i < v; i++)
        {
            visited[i] = false;
        }
        printbfs(arr, v, v1, v2, visited);
        cout << endl;
    }
}
```

## 5-Tut : Code : Is Connected ?

Send Feedback

Given an undirected graph G(V,E), check if the graph G is connected graph or not.

V is the number of vertices present in graph G and vertices are numbered from 0 to V-1.

E is the number of edges present in graph G.

### Input Format :

First line will contain T(number of test case), each test case follows as.
Line 1: Two Integers V and E (separated by space)
Next 'E' lines, each have two space-separated integers, 'a' and 'b', denoting that there exists an edge between Vertex 'a' and Vertex 'b'.

### Output Format :

Print "true" or "false" for each test case in new line

### Constraints :

1 <= T <= 10
2 <= V <= 1000
1 <= E <= 1000

### Sample Input 1:

```
1
4 4
0 1
0 3
1 2
2 3
```

### Sample Output 1:

true

### Sample Input 2:

```
1
4 3
0 1
1 3
0 3
```

### Sample Output 2:

false

### Sample Output 2 Explanation

The graph is not connected, even though vertices 0,1 and 3 are connected to each other but there isn't any path from vertices 0,1,3 to vertex 2.

```
1.  #include <bits/stdc++.h>
2.  using namespace std;
3.
4.  void printBFS(int **edges, int n, int sv, bool *visited)
5.  {
6.
7.      queue<int> pendingVertices;
8.      pendingVertices.push(sv);
9.      visited[sv] = true;
10.
11.     while (!pendingVertices.empty())
```

```
12.     {
13.
14.         int currentVertex = pendingVertices.front();
15.         pendingVertices.pop();
16.         for (int i = 0; i < n; i++)
17.         {
18.
19.             if (i == currentVertex)
20.             {
21.                 continue;
22.             }
23.
24.             if (edges[currentVertex][i] == 1 && !visited[i])
25.             {
26.                 pendingVertices.push(i);
27.                 visited[i] = true;
28.             }
29.         }
30.     }
31. }
32.
33. void BFS(int **edges, int n)
34. {
35.     bool *visited = new bool[n];
36.     for (int i = 0; i < n; i++)
37.     {
38.         visited[i] = false;
39.     }
40.
41.     printBFS(edges, n, 0, visited);
42.
43.     int flag = 0;
44.     for (int i = 0; i < n; i++)
45.     {
46.         if (!visited[i])
47.         {
48.             flag = 1;
49.             break;
50.         }
51.     }
52.
53.     if (flag == 1)
54.     {
55.         cout << "false" << endl;
```

```
    }
    else
    {
        cout << "true" << endl;
    }
    delete[] visited;
}

int main()
{

    int t;
    cin >> t;
    while (t--)
    {
        int n;
        int e;

        cin >> n >> e;

        int **edges = new int *[n];
        for (int i = 0; i < n; i++)
        {
            edges[i] = new int[n];
            for (int j = 0; j < n; j++)
            {
                edges[i][j] = 0;
            }
        }

        for (int i = 0; i < e; i++)
        {
            int f, s;
            cin >> f >> s;
            edges[f][s] = 1;
            edges[s][f] = 1;
        }

        BFS(edges, n);

        for (int i = 0; i < n; i++)
        {
            delete[] edges[i];
        }
```

```
100.
101.            delete[] edges;
102.        }
103.
104.        return 0;
105.    }
```

## 6-Tut : Code : All connected components

Send Feedback

Given an undirected graph G(V,E), find and print all the connected components of the given graph G.

V is the number of vertices present in graph G and vertices are numbered from 1 to V.

E is the number of edges present in graph G.

You need to take input in main and create a function which should return all the connected components.

And then print them in the main, not inside function.

Print different components in new line. And each component should be printed in increasing order (separated by space). Order of different components doesn't matter.

### Input Format :

First line of input will contain T(number of test case), each test case follows as.
Line 1: Two Integers V and E (separated by space)
Next 'E' lines, each have two space-separated integers, 'a' and 'b', denoting that there exists an edge between Vertex 'a' and Vertex 'b'.

### Output Format :

For each test case and each connected components print the connected components in sorted order in new line.
Order of connected components doesn't matter (print as you wish).

### Constraints :

2 <= V <= 10000
1 <= E <= 10000

### Sample Input 1:

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

### Sample Output 1:

```
1 2

4 3
```

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. #define ll int
5.
6. int arr[10001][10001];
```

```
7.
8.  struct query
9.  {
10.     int aa;
11.     int bb;
12. };
13.
14. struct node
15. {
16.     vector<int> nb;
17. };
18.
19. vector<ll> vector_creator(ll n, ll starting_vertex, bool *visited, int max, node *nodes)
20. {
21.     vector<ll> temp;
22.     queue<ll> q;
23.
24.     q.push(starting_vertex);
25.
26.     visited[starting_vertex] = true;
27.
28.     while (!q.empty())
29.     {
30.         ll current_element = q.front();
31.         temp.push_back(current_element);
32.         q.pop();
33.         vector<int> t1;
34.         t1 = nodes[current_element].nb;
35.
36.         for (int i = 0; i < t1.size(); i++)
37.         {
38.             if (t1[i] == current_element)
39.             {
40.                 continue;
41.             }
42.             if (!visited[t1[i]])
43.             {
44.                 q.push(t1[i]);
45.                 visited[t1[i]] = true;
46.             }
47.         }
48.     }
49.     return temp;
50. }
```

```
51.
52. vector<vector<ll>> vector_return(ll n, int max, node *nodes)
53. {
54.
55.     bool *visited = new bool[n + 1];
56.     for (ll i = 0; i <= n; i++)
57.     {
58.         visited[i] = false;
59.     }
60.
61.     vector<vector<ll>> super;
62.     vector<ll> tempo;
63.     for (ll i = 1; i <= max; i++)
64.     {
65.         if (!visited[i])
66.         {
67.             tempo = vector_creator(n, i, visited, max, nodes);
68.             if (tempo.size() != 0)
69.             {
70.                 super.push_back(tempo);
71.             }
72.         }
73.     }
74.     for (int i = max + 1; i <= n; i++)
75.     {
76.         vector<int> t1;
77.         t1.push_back(i);
78.         super.push_back(t1);
79.     }
80.
81.     return super;
82. }
83.
84. int main()
85. {
86.     ll t;
87.     cin >> t;
88.     while (t--)
89.     {
90.         ll v, e;
91.         cin >> v >> e;
92.         int max = 0;
93.
94.         node *nodes = new node[v + 1];
```

```
95.     for (int i = 0; i <= v; i++)
96.     {
97.         nodes[i].nb.push_back(i);
98.     }
99.
100.        query q[e];
101.        for (int i = 0; i < e; i++)
102.        {
103.
104.            ll a, b;
105.            cin >> a >> b;
106.
107.            arr[a][b] = 1;
108.            arr[b][a] = 1;
109.
110.            q[i].aa = a;
111.            q[i].bb = b;
112.
113.            nodes[a].nb.push_back(b);
114.            nodes[b].nb.push_back(a);
115.
116.            if (max < a)
117.            {
118.                max = a;
119.            }
120.            if (max < b)
121.            {
122.                max = b;
123.            }
124.        }
125.
126.        vector<vector<ll>> super;
127.
128.        super = vector_return(v, max, nodes);
129.
130.        for (ll i = 0; i < super.size(); i++)
131.        {
132.            sort(super[i].begin(), super[i].end());
133.            for (ll j = 0; j < super[i].size(); j++)
134.            {
135.                cout << super[i][j] << " ";
136.            }
137.            cout << endl;
138.        }
```

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

            arr[q[i].aa][q[i].bb] = 0;
            arr[q[i].bb][q[i].aa] = 0;
        }
    }
}
```

## 7-Ass : Islands

Send Feedback

An island is a small piece of land surrounded by water . A group of islands is said to be connected if we can reach from any given island to any other island in the same group . Given N islands (numbered from 0 to N - 1) and M pair of integers (u and v) denoting island, u is connected to island v and vice versa. Can you count the number of connected groups of islands?

### Input Format:

The first line of input will contain T(number of test cases), each test case follows as.
Line 1: Two Integers N and M (separated by space)
Next 'M' lines, each have two space-separated integers, 'u' and 'v', denoting that there exists an edge between Vertex 'u' and Vertex 'v'.

### Output Format:

Print number of Islands for each test case in new line.

### Constraints:

1 <= T <= 10
1 <= N <= 1000
1 <= M <= min((N*(N-1))/2, 1000)
0 <= u[i] ,v[i] < N

### Output Return Format :

The count the number of connected groups of islands

### Sample Input :

1
2 1
0 1

### Sample Output :

1

```
#include <bits/stdc++.h>
using namespace std;
void visited_filler(int **arr, int n, int starting_vertex, bool *visited)
{
    queue<int> q;
    q.push(starting_vertex);
```

```
7.      visited[starting_vertex] = true;
8.      while (!q.empty())
9.      {
10.         int current_element = q.front();
11.         for (int i = 0; i < n; i++)
12.         {
13.             if (i == current_element)
14.             {
15.                 continue;
16.             }
17.             if (visited[i])
18.             {
19.                 continue;
20.             }
21.             if (arr[current_element][i] == 1)
22.             {
23.                 q.push(i);
24.                 visited[i] = true;
25.             }
26.         }
27.         q.pop();
28.     }
29. }
30.
31. int main()
32. {
33.
34.     int t;
35.     cin >> t;
36.     while (t--)
37.     {
38.
39.         int N, M;
40.         cin >> N >> M;
41.
42.         int **arr = new int *[N + 1];
43.         for (int i = 0; i < N + 1; i++)
44.         {
45.             arr[i] = new int[N + 1];
46.             for (int j = 0; j < N + 1; j++)
47.             {
48.                 arr[i][j] = 0;
49.             }
50.         }
```

```
51.
52.         for (int i = 0; i < M; i++)
53.         {
54.             int u, v;
55.             cin >> u >> v;
56.             arr[u][v] = 1;
57.             arr[v][u] = 1;
58.         }
59.
60.         bool *visited = new bool[N + 1];
61.         for (int i = 0; i < N + 1; i++)
62.         {
63.             visited[i] = false;
64.         }
65.
66.         int count = 0;
67.
68.         for (int i = 0; i < N; i++)
69.         {
70.             if (!visited[i])
71.             {
72.                 visited_filler(arr, N + 1, i, visited);
73.                 count += 1;
74.             }
75.         }
76.         cout << count << endl;
77.     }
78.
79.     return 0;
80. }
```

## 8-Ass : Coding Ninjas

Send Feedback

Given a NxM matrix containing Uppercase English Alphabets only. Your task is to tell if there is a path in the given matrix which makes the sentence “CODINGNINJA” .

There is a path from any cell to all its neighbouring cells. A neighbour may share an edge or a corner.

### Input Format :

First line will contain T(number of test case), each test case follows as.
Line 1 : Two space separated integers N  and M, where N is number of rows and M is number of columns in the matrix.
Next N lines : N rows of the matrix. First line of these N line will contain 0th row of matrix, second line will contain 1st row and so on

### Assume input to be 0-indexed based

**Output Format :**

Print 1 if there is a path which makes the sentence “CODINGNINJA” else print 0, for each test case in a new line

**Constraints :**

1 <= T <= 10
1 <= N <= 1000
1 <= M <= 1000

**Sample Input :**

1
2 11
CXDXNXNXNXA
XOXIXGXIXJX

**Sample Output :**

1

```
#include <bits/stdc++.h>
using namespace std;

#define MAXN 500

int validPoint(int x, int y, int n, int m)
{

    return (x >= 0 && x < n && y >= 0 && y < m);
}

bool dfs(char **board, vector<vector<bool>> &used, string &word, int x, int y, int
wordIndex, int n, int m)
{

    if (wordIndex == 11)
    {
        return true;
    }

    used[x][y] = true;
    bool found = false;

    int dxdy[8][2] = {{-1, -1},
                      {-1, 0},
                      {-1, -1},
                      {0, -1},
                      {0, 1},
                      {1, -1},
```

```
                    {1, 0},
                    {1, 1}};

    for (int i = 0; i < 8; ++i)
    {

        int newX = x + dxdy[i][0];
        int newY = y + dxdy[i][1];

        if (validPoint(newX, newY, n, m) && board[newX][newY] == word[wordIndex] &&
    !used[newX][newY])
        {

            found = found | dfs(board, used, word, newX, newY, wordIndex + 1, n, m);
        }
    }

    used[x][y] = false;

    return found;
}

bool hasPath(char **board, int n, int m)
{

    bool foundPath = false;
    string word = "CODINGNINJA";
    vector<vector<bool>> used(n, vector<bool>(m, false));

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

            if (board[i][j] == word[0])
            {
                foundPath = dfs(board, used, word, i, j, 1, n, m);
                if (foundPath)
                    break;
            }
        }

        if (foundPath)
            break;
```

```
72.     }
73.
74.     return foundPath;
75. }
76.
77. int main()
78. {
79.
80.     int t;
81.     cin >> t;
82.     while (t--)
83.     {
84.         int N, M;
85.         cin >> N >> M;
86.
87.         char **arr = new char *[N];
88.         for (int i = 0; i < N; i++)
89.         {
90.             arr[i] = new char[M];
91.             for (int j = 0; j < M; j++)
92.             {
93.                 arr[i][j] = ' ';
94.             }
95.         }
96.
97.         for (int i = 0; i < N; i++)
98.
99.         {
100.                for (int j = 0; j < M; j++)
101.
102.                {
103.                    cin >> arr[i][j];
104.                }
105.            }
106.
107.            bool ans = hasPath(arr, N, M);
108.
109.            if (ans)
110.            {
111.                cout << 1 << endl;
112.            }
113.            else
114.            {
115.                cout << 0 << endl;
```

```
            }
        }

        return 0;
    }
```

## 9-Ass : Connecting Dots

Send Feedback

Gary has a board of size NxM. Each cell in the board is a coloured dot. There exist only 26 colours denoted by uppercase Latin characters (i.e. A,B,...,Z). Now Gary is getting bore and wants to play a game. The key of this game is to find a cycle that contain dots of same colour. Formally, we call a sequence of dots d1, d2, ..., dk a cycle if and only if it meets the following condition:

1. These k dots are different: if i ≠ j then di is different from dj.
2. k is at least 4.
3. All dots belong to the same colour.
4. For all 1 ≤ i ≤ k - 1: di and di + 1 are adjacent. Also, dk and d1 should also be adjacent. Cells x and y are called adjacent if they share an edge.

Since Gary is colour blind, he wants your help. Your task is to determine if there exists a cycle on the board.

**Assume input to be 0-indexed based.**

**Input Format :**

Line 1 : Two integers N and M, the number of rows and columns of the board
Next N lines : a string consisting of M characters, expressing colors of dots in each line. Each character is an uppercase Latin letter.

**Output Format :**

Return 1 if there is a cycle else return 0

**Constraints :**

2 ≤ N, M ≤ 400

**Sample Input :**

```
3 4
AAAA
ABCA
AAAA
```

**Sample Output :**

```
1
```

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, char> pi;

bool vis[51][51];
char graph[51][51];
```

```
7.  int dx[] = {1, -1, 0, 0}; // only left, right , up ,down are allowed here
8.  int dy[] = {0, 0, 1, -1};
9.  int m, n;
10. bool ok = false;
11.
12. void dfs(int i, int j, int frmi, int frmj, char co)
13. {
14.     if (i < 1 || j < 1 || i > m || j > n)
15.         return;
16.
17.     if (graph[i][j] != co)
18.         return;
19.
20.     if (vis[i][j])
21.     {
22.         ok = true;
23.         return;
24.     }
25.
26.     vis[i][j] = true;
27.
28.     for (int y = 0; y < 4; y++)
29.     {
30.         int nxti = i + dx[y];
31.         int nxtj = j + dy[y];
32.
33.         if (nxti == frmi && nxtj == frmj)
34.             continue; // it doesn't go back from where it comes from
35.
36.         dfs(nxti, nxtj, i, j, co);
37.     }
38. }
39.
40. int main()
41.
42. {
43.
44.     memset(vis, false, sizeof(vis));
45.     int x, y, u, v;
46.     char c;
47.     cin >> m >> n;
48.     for (int i = 1; i <= m; i++)
49.
50.     {
```

```
        for (int j = 1; j <= n; j++)

        {
            cin >> graph[i][j];
        }
    }
    for (int i = 1; i <= m; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            char z = graph[i][j];
            // cout<<z<<endl;
            if (!vis[i][j])
            {
                dfs(i, j, -1, -1, z);
                if (ok)
                {
                    cout << "1" << endl;
                    return 0;
                }
            }
        }
    }
    cout << "0" << endl;
}
```

## 10-Ass : Largest Piece

Send Feedback

Its Gary's birthday today and he has ordered his favourite square cake consisting of '0's and '1's . But Gary wants the biggest piece of '1's and no '0's . A piece of cake is defined as a part which consist of only '1's, and all '1's share an edge with eachother on the cake. Given the size of cake N and the cake , can you find the size of the biggest piece of '1's for Gary ?

### Input Format :

First line will contain T(number of test cases), each test case follows as.
Line 1 : An integer N denoting the size of cake
Next N lines : N characters denoting the cake

### Output Format :

Print the size of the biggest piece of '1's and no '0'sfor each test case in a newline.

### Constraints:

1 <= T <= 10
1 <= N <= 1000

**Sample Input :**

```
1
2
11
01
```

**Sample Output :**

```
3
```

```
#include <bits/stdc++.h>
using namespace std;
#define NMAX 10005
char cake[NMAX][NMAX];

void dfs(char cake[NMAX][NMAX], int n, int &k, int i, int j)
{
    k++;
    cake[i][j] = '0';

    if (i + 1 < n && cake[i + 1][j] == '1')
        dfs(cake, n, k, i + 1, j);
    if (i - 1 >= 0 && cake[i - 1][j] == '1')
        dfs(cake, n, k, i - 1, j);
    if (j + 1 < n && cake[i][j + 1] == '1')
        dfs(cake, n, k, i, j + 1);
    if (j - 1 >= 0 && cake[i][j - 1] == '1')
        dfs(cake, n, k, i, j - 1);
}

int solve(int n, char cake[NMAX][NMAX])
{
    // Write your code here .
    int ans = 0;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {

            if (cake[i][j] == '1')
            {
                int k1 = 0;
                dfs(cake, n, k1, i, j);
                ans = max(ans, k1);
            }
        }
```

```
    }
    return ans;
}

int main()
{

    // write your code here
    int t;
    cin >> t;
    while (t--)
    {
        int n;
        cin >> n;
        for (int i = 0; i < n; i++)
        {
            scanf("%s", cake[i]);
        }
        cout << solve(n, cake) << endl;
    }

    return 0;
}
```

## 11-Ass : 3 Cycle

Send Feedback

Given a graph with N vertices (numbered from 1 to N) and Two Lists (U,V) of size M where (U[i],V[i]) and (V[i],U[i]) are connected by an edge , then count the distinct 3-cycles in the graph. A 3-cycle PQR is a cycle in which (P,Q), (Q,R) and (R,P) are connected an edge.

### Input Format :

Line 1 : Two integers N and M
Line 2 : List u of size of M
Line 3 : List v of size of M

### Return Format :

The count the number of 3-cycles in the given Graph

### Constraints :

1<=N<=100
1<=M<=(N*(N-1))/2
1<=u[i],v[i]<=N

### Sample Input:

3 3
1 2 3
2 3 1

**Sample Output:**

1

```
#include <bits/stdc++.h>
using namespace std;

int solve(int n, int m, vector<int> u, vector<int> v)
{
    // Write your code here .
    unordered_map<int, unordered_set<int>> adj;

    for (int i = 0; i < m; i++)
    {
        adj[u[i]].insert(v[i]);
        adj[v[i]].insert(u[i]);
    }

    int thcycle = 0;
    for (int i = 1; i <= n; i++)
    {
        int node = i;

        for (int p = 1; p < n; p++)
        {
            for (int q = p + 1; q <= n; q++)
            {

                if (adj[node].count(p) == 0)
                    break;
                if (adj[node].count(q) == 0)
                    continue;

                if (adj[node].count(p) && adj[node].count(q))
                {
                    if (adj[p].count(q))
                        thcycle++;
                }
            }
        }
    }
    return thcycle / 3;
}

int main()
```

```
{
    int n, m;
    vector<int> u, v;
    cin >> n >> m;
    for (int i = 0; i < m; i++)
    {
        int x;
        cin >> x;
        u.push_back(x);
    }
    for (int i = 0; i < m; i++)
    {
        int x;
        cin >> x;
        v.push_back(x);
    }
    cout << solve(n, m, u, v) << endl;
}
```

# L23 : Graphs 2

## 1-Tut : Kruskal's Algorithm

Send Feedback

Given an undirected, connected and weighted graph G(V, E) with V number of vertices (which are numbered from 0 to V-1) and E number of edges.

Find and print the total weight of Minimum Spanning Tree (MST) using Kruskal's algorithm.

### Input Format :

First line will contain T(number of test case), each test case follows as.
Line 1: Two Integers V and E (separated by space)
Next E lines : Three integers ei, ej and wi, denoting that there exists an edge between vertex ei and vertex ej with weight wi (separated by space)

### Output Format :

Weight of MST for each test case in new line.

### Constraints :

1 <= T <= 10
2 <= V, E <= 10^5
1 <= wt <= 10^4

### Sample Input 1 :

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

### Sample Output 1 :

```
9
```

```
1. #include <iostream>
2. #include <algorithm>
3. using namespace std;
4. // Class that store values for each vertex
5. class Edge
6. {
7. public:
8.     int source;
9.     int dest;
10.    int weight;
11. };
12. // Comparator function used to sort edges
13. bool compare(Edge e1, Edge e2)
14. {
```

```
    // Edges will sorted in order of their weights
    return e1.weight < e2.weight;
}
// Function to find the parent of a vertex
int findParent(int v, int *parent)
{
    // Base case, when a vertex is parent of itself
    if (parent[v] == v)
    {
        return v;
    }
    // Recursively called to find the topmost parent of the vertex.
    return findParent(parent[v], parent);
}
void kruskals(Edge *input, int n, int E)
{
    // In-built sort function: Sorts the edges in
    // increasing order of their weights
    sort(input, input + E, compare);
    // Array to store final edges of MST
    Edge *output = new Edge[n - 1];
    // Parent array initialized with their indexes
    int *parent = new int[n];
    for (int i = 0; i < n; i++)
    {
        parent[i] = i;
    }
    int count = 0; // To maintain the count of number of edges in    the MST
    int i = 0;     // Index to traverse over the input array
    while (count != n - 1)
    { // As the MST contains n-1 edges.
        Edge currentEdge = input[i];
        // Figuring out the parent of each edge's vertices
        int sourceParent = findParent(currentEdge.source, parent);
        int destParent = findParent(currentEdge.dest, parent);
        // If their parents are not equal, then we added that edge to        output
        if (sourceParent != destParent)
        {
            output[count] = currentEdge;
            count++;                        // Increased the count
            parent[sourceParent] = destParent; // Updated the parent array
        }
        i++;
    }
```

```
    // Finally, printing the MST obtained.
    long long int sum = 0;
    for (int i = 0; i < n - 1; i++)
    {
        if (output[i].source < output[i].dest)
        {
            sum += (output[i].weight);
        }
        else
        {
            sum += (output[i].weight);
        }
    }

    cout << sum << endl;
}
int main()
{

    int t;
    cin >> t;
    while (t--)
    {

        int n, E;
        cin >> n;
        cin >> E;
        Edge *input = new Edge[E];
        for (int i = 0; i < E; i++)
        {
            int s, d, w;
            cin >> s >> d >> w;
            input[i].source = s;
            input[i].dest = d;
            input[i].weight = w;
        }
        kruskals(input, n, E);
    }
    return 0;
}
```

## 2-Tut : Prim's Algorithm

Send Feedback

Given an undirected, connected and weighted graph G(V, E) with V number of vertices (which are numbered from 0 to V-1) and E number of edges.

Find and print the total weight of Minimum Spanning Tree (MST) using Prim's algorithm.

### Input Format :

First line will contain T(number of test case), each test case follows as.
Line 1: Two Integers V and E (separated by space)
Next E lines : Three integers ei, ej and wi, denoting that there exists an edge between vertex ei and vertex ej with weight wi (separated by space)

### Output Format :

Weight of MST for each test case in new line.

### Constraints :

1 <= T <= 10
2 <= V, E <= 10^5
1 <= wt <= 10^4

### Sample Input 1 :

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

### Sample Output 1 :

```
9
```

```
#include <bits/stdc++.h>
using namespace std;

int findMinVertex(int *weights, bool *visited, int n)
{
    // Initialized to -1 means there is no vertex till now
    int minVertex = -1;
    for (int i = 0; i < n; i++)
    {
        // Conditions: the vertex must be unvisited and either minVertex value is -1
        //  or if minVertex has some vertex to it, then weight of currentvertex
        //  should be less than the weight of the minVertex.
        if (!visited[i] && (minVertex == -1 || weights[i] < weights[minVertex]))
        {
            minVertex = i;
        }
    }
```

```
    return minVertex;
}
void prims(int **edges, int n)
{
    int *parent = new int[n];
    int *weights = new int[n];
    bool *visited = new bool[n];
    // Initially, the visited array is assigned to false and weights array
    // to infinity.
    for (int i = 0; i < n; i++)
    {
        visited[i] = false;
        weights[i] = INT_MAX;
    }
    // Values assigned to vertex 0.(the selected starting vertex to begin with)
    parent[0] = -1;
    weights[0] = 0;
    for (int i = 0; i < n - 1; i++)
    {
        // Find min vertex
        int minVertex = findMinVertex(weights, visited, n);
        visited[minVertex] = true;
        // Explore unvisited neighbors
        for (int j = 0; j < n; j++)
        {
            if (edges[minVertex][j] != 0 && !visited[j])
            {
                if (edges[minVertex][j] < weights[j])
                {
                    // updating weight array and parent array
                    weights[j] = edges[minVertex][j];
                    parent[j] = minVertex;
                }
            }
        }
    }
    // Final MST printed
    long long int sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum += (weights[i]);
    }

    cout << sum << endl;
```

```
}
int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        int n;
        int e;
        cin >> n >> e;
        int **edges = new int *[n]; // Adjacency matrix used to store the graph
        for (int i = 0; i < n; i++)
        {
            edges[i] = new int[n];
            for (int j = 0; j < n; j++)
            {
                // Initially all pairs are assigned 0 weight which
                // means that there is no edge between them
                edges[i][j] = INT_MAX;
            }
        }
        for (int i = 0; i < e; i++)
        {
            int f, s, weight;
            cin >> f >> s >> weight;

            if (edges[f][s] > weight)
            {
                edges[f][s] = weight;
            }
            if (edges[s][f] > weight)
            {
                edges[s][f] = weight;
            }
        }
        prims(edges, n);
        for (int i = 0; i < n; i++)
        {
                delete[] edges[i];
            }
            delete[] edges;
        }

        return 0; }
```

## 3-Tut : Dijkstra's Algorithm

Send Feedback

Given an undirected, connected and weighted graph G(V, E) with V number of vertices (which are numbered from 0 to V-1) and E number of edges.

Find and print the shortest distance from the source vertex (i.e. Vertex 0) to all other vertices (including source vertex also) using Dijkstra's Algorithm.

Print the ith vertex number and the distance from source in one line separated by space. Print different vertices in different lines.

**Note : Order of vertices in output doesn't matter.**

**Input Format :**

First line will contain T(number of test case), each test case follows as.
Line 1: Two Integers V and E (separated by space)
Next E lines : Three integers ei, ej and wi, denoting that there exists an edge between vertex ei and vertex ej with weight wi (separated by space)

**Output Format :**

In different lines, ith vertex number and its distance from source (separated by space)

**Constraints :**

1 <= T <= 10
2 <= V, E <= 10^3

**Sample Input 1 :**

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

**Sample Output 1 :**

```
0 0
1 3
2 4

3 5
```

```
#include <iostream>
#include <climits>
using namespace std;
int findMinVertex(int *distance, bool *visited, int n)
{
    int minVertex = -1;
    for (int i = 0; i < n; i++)
    {
        if (!visited[i] && (minVertex == -1 || distance[i] < distance[minVertex]))
        {
            minVertex = i;
```

```
        }
    }
    return minVertex;
}
void dijkstra(int **edges, int n)
{
    int *distance = new int[n];
    bool *visited = new bool[n];
    for (int i = 0; i < n; i++)
    {
        visited[i] = false;
        distance[i] = INT_MAX;
    }
    distance[0] = 0; // 0 is considered as the starting node.
    for (int i = 0; i < n - 1; i++)
    {
        // Find min vertex
        int minVertex = findMinVertex(distance, visited, n);
        visited[minVertex] = true;
        // Explore unvisited neighbors
        for (int j = 0; j < n; j++)
        {
            if (edges[minVertex][j] != 0 && !visited[j])
            {
                // distance of any node will be the current node's distance + the weight
                // of the edge between them
                int dist = distance[minVertex] + edges[minVertex][j];
                if (dist < distance[j])
                { // If required, then updated.
                    distance[j] = dist;
                }
            }
        }
    }
    // Final output of distance of each node with respect to 0
    for (int i = 0; i < n; i++)
    {
        cout << i << " " << distance[i] << endl;
    }
}
int main()
{
    int t;
    cin >> t;
```

```
    while (t--)
    {
        int n;
        int e;
        cin >> n >> e;
        int **edges = new int *[n];
        for (int i = 0; i < n; i++)
        {
            edges[i] = new int[n];
            for (int j = 0; j < n; j++)
            {
                edges[i][j] = 0;
            }
        }
        for (int i = 0; i < e; i++)
        {
            int f, s, weight;
            cin >> f >> s >> weight;
            edges[f][s] = weight;
            edges[s][f] = weight;
        }
        dijkstra(edges, n);
        for (int i = 0; i < n; i++)
        {
            delete[] edges[i];
        }
        delete[] edges;
    }
    return 0;
}
```

## 4-Tut : Bellman-Ford Algorithm

Send Feedback

you are given a weighted directed graph G with n vertices and m edges, and two specified vertex src and des. You want to find the length of shortest paths from vertex src to des. The graph may contain the edges with negative weight.

### Input Format:

First line of input will contain T(number of test case), each test case follows as.
Line1: contain two space-separated integers N and M denoting the number of vertex and number of edges in graph respectively.
Line2: contain two space-separated integers src, des.
Next M line will contain three space-separated integers a, b, wt representing the edge from a to b with weight wt.

**Output Format:**

For each test case print the distance of des from src in new line.
Note: In case of no path is found print (10 ^ 9) in that case.

**Constraints:**

1 <= T <= 100
1 <= N <= 200
1 <= M <= min(800, N*(N-1))
1 <= a,b <= N
-10^5 <= wt <= 10^5

**Sample Input:**

```
1
3 6
3 1
3 1 -2
1 3 244
2 3 -2
2 1 201
3 2 220
1 2 223
```

**Sample output:**

```
-2
```

```
#include <bits/stdc++.h>
using namespace std;
struct Edge
{
    int src, dest, weight;
};
struct Graph
{
    int V, E;
    struct Edge *edge;
};
struct Graph *createGraph(int V, int E)
{
    struct Graph *graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge[E];
    return graph;
}
void printAns(int dist[], int n, int src, int dest)
{
    for (int i = 1; i <= n; ++i)
        if (i == dest)
```

```
24.             if (dist[i] == INT_MAX)
25.             {
26.                 cout << "1000000000" << endl;
27.             }
28.             else
29.             {
30.                 cout << dist[i] << endl;
31.             }
32. }
33. void BellmanFord(struct Graph *graph, int src, int dest)
34. {
35.     int V = graph->V;
36.     int E = graph->E;
37.     int dist[V + 1];
38.     for (int i = 0; i <= V; i++)
39.         dist[i] = INT_MAX;
40.     dist[src] = 0;
41.     for (int i = 1; i <= V - 1; i++)
42.     {
43.         for (int j = 0; j < E; j++)
44.         {
45.             int u = graph->edge[j].src;
46.             int v = graph->edge[j].dest;
47.             int weight = graph->edge[j].weight;
48.             if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
49.                 dist[v] = dist[u] + weight;
50.         }
51.     }
52.     for (int i = 0; i < E; i++)
53.     {
54.         int u = graph->edge[i].src;
55.         int v = graph->edge[i].dest;
56.         int weight = graph->edge[i].weight;
57.         if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
58.         {
59.             cout << "1000000000" << endl;
60.             return; // If negative cycle is detected, simply return
61.         }
62.     }
63.     printAns(dist, V, src, dest);
64.     return;
65. }
66. int main()
67. {
```

```
    int t;
    cin >> t;
    while (t--)
    {
        int V, E;
        int src, dest;
        cin >> V;
        cin >> E;
        cin >> src;
        cin >> dest;
        struct Graph *graph = createGraph(V, E);
        for (int i = 0; i < E; i++)
        {
            int u, v, w;
            cin >> u >> v >> w;
            graph->edge[i].src = u;
            graph->edge[i].dest = v;
            graph->edge[i].weight = w;
        }
        BellmanFord(graph, src, dest);
    }

    return 0;
}
```

## 5-Tut : Floyd-Warshall Algorithm

Send Feedback

You are given an undirected weighted graph G with n vertices. And Q queries, each query consists of two integers a and b and you have print the distance of shortest path between a and b.

Note: If there is no path between a and b print 10^9

### Input Format:

First line of Input will contain T(number of test cases), each test case follows as.
Line1: contains two space-separated integers N and M denoting the number of vertex and edge in graph.
Next M line contain three space-separated integers a, b, c denoting the edge between a and b with weight c.
Next line will contain Q (number of queries)
Next Q lines will contain two space-separated integers a and b.

### Output Format:

For each query of each test case print the answer in a newline.

### Constraints:

1 <= T <= 50
1 <= N <= 100
1 <= M <= 10^4
1 <= Q <= 10^4

1 <= wt <= 10^5 (for each edge)
Note: Graph may contain multiple edges.

**Sample Input:**

```
1
3 6
3 1 4
1 3 17
2 3 2
1 3 7
3 2 11
2 3 15
3
1 1
2 2
2 3
```

**Sample output:**

```
0
0

2
```

```cpp
#include <bits/stdc++.h>
using namespace std;
int INF = 1e9;
int main()
{

    int t;
    cin >> t;
    while (t--)
    {
        int n, m;

        cin >> n;

        cin >> m;
        int mat[n + 1][n + 1];
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j <= n; j++)
            {
                mat[i][j] = INF;
                if (i == j)
                {
                    mat[i][j] = 0;
```

```
            }
        }
    }
    for (int i = 0; i < m; i++)
    {
        int a, b, c;
        cin >> a >> b >> c;
        if (mat[a][b] < c)
        {
            continue;
        }
        else
        {
            mat[a][b] = c;
            mat[b][a] = c;
        }
    }
    int i, j, k;
    for (k = 1; k <= n; k++)
    {
        for (i = 1; i <= n; i++)
        {
            for (j = 1; j <= n; j++)
            {
                if (mat[i][j] > (mat[i][k] + mat[k][j]) && (mat[k][j] != INF && mat[i][k] != INF))
                    mat[i][j] = mat[i][k] + mat[k][j];
            }
        }
    }
    int q;
    cin >> q;
    while (q--)
    {
        int aa;
        int bb;
        cin >> aa >> bb;

        if (aa == bb)
        {
            cout << 0 << endl;
        }
        else
        {
            cout << mat[aa][bb] << endl;
```

```
            }
        }
    }
}
```

**6-Bonus : Disjoint Set Union**

```
#include <bits/stdc++.h>
using namespace std;

class DSNode
{
public:
    int data;
    DSNode *parent;
    int rank;
};

class DisjointSet
{
private:
    map<int, DSNode *> hash;

    DSNode *searchInSetHelper(DSNode *node)
    {

        if (node == node->parent)
        {
            return node;
        }
        node->parent = searchInSetHelper(node->parent);
        return node->parent;
    }

public:
    void initializeSet(int data)
    {
        DSNode *node = new DSNode();
        node->data = data;
        node->parent = node;
        node->rank = 0;
        hash[data] = node;
    }
```

```
37.
38.     void Union(int data1, int data2)
39.     {
40.
41.         DSNode *node1 = hash[data1];
42.         DSNode *node2 = hash[data2];
43.
44.         DSNode *parent1 = searchInSetHelper(node1);
45.         DSNode *parent2 = searchInSetHelper(node2);
46.
47.         if (parent1->data == parent2->data)
48.         {
49.             return;
50.         }
51.
52.         if (parent1->rank >= parent2->rank)
53.         {
54.             if (parent1->rank == parent2->rank)
55.             {
56.                 parent1->rank = parent1->rank + 1;
57.             }
58.             parent2->parent = parent1;
59.         }
60.         else
61.         {
62.             parent1->parent = parent2;
63.         }
64.     }
65.
66.     int searchInSet(int data)
67.     {
68.         return searchInSetHelper(hash[data])->data;
69.     }
70. };
71.
72. int main()
73. {
74.
75.     DisjointSet ds;
76.
77.     ds.initializeSet(0);
78.     ds.initializeSet(1);
79.     ds.initializeSet(2);
80.     ds.initializeSet(3);
```

```
    ds.initializeSet(4);
    ds.initializeSet(5);
    ds.initializeSet(6);

    ds.Union(0, 1);
    ds.Union(1, 2);
    ds.Union(3, 4);
    ds.Union(2, 4);
    ds.Union(5, 6);
    ds.Union(4, 6);
    ds.Union(0, 1);

    cout << ds.searchInSet(0) << endl;
    cout << ds.searchInSet(1) << endl;
    cout << ds.searchInSet(2) << endl;
    cout << ds.searchInSet(3) << endl;
    cout << ds.searchInSet(4) << endl;
    cout << ds.searchInSet(5) << endl;
    cout << ds.searchInSet(6) << endl;

        return 0;
    }
```

# L24 : Advanced Graphs

## 1-Tut : Permutation Swaps

Send Feedback

Kevin has a permutation P of N integers 1, 2, ..., N, but he doesn't like it. Kevin wants to get a permutation Q.

He also believes that there are M good pairs of integers (ai, bi). Kevin can perform following operation with his permutation:

Swap Px and Py only if (x, y) is a good pair.
Help him and tell if Kevin can obtain permutation Q using such operations.

### Input format:

The first line of input will contain an integer T, denoting the number of test cases.
Each test case starts with two space-separated integers N and M. The next line contains N space-separated integers Pi. The next line contains N space-separated integers Qi. Each of the next M lines contains two space-separated integers ai and bi.

### Output format:

For every test case output "YES" (without quotes) if Kevin can obtain permutation Q and "NO" otherwise.

### Constraints:

1 ≤ T ≤ 10
2 ≤ N ≤ 100000
1 ≤ M ≤ 100000
1 ≤ Pi, Qi ≤ N. Pi and Qi are all distinct.
1 ≤ ai < bi ≤ N
Time Limit: 1 second

### Sample Input 1:

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

### Sample Output 1:

```
NO

YES
```

```
1. #include <iostream>
2. #include <vector>
3. #include <unordered_set>
4. #include <iterator>
```

```
using namespace std;
void dfs(vector<int> *edges, int start, bool *visited, unordered_set<int> *component)
{
    if (visited[start])
    {
        return;
    }
    visited[start] = true;
    component->insert(start);
    for (vector<int>::iterator it = edges[start].begin(); it != edges[start].end(); it++)
    {
        if (!visited[*it])
        {
            dfs(edges, *it, visited, component);
        }
    }
}
unordered_set<unordered_set<int> *> *getComponents(vector<int> *edges, int n)
{
    bool *visited = new bool[n];
    for (int i = 0; i < n; i++)
    {
        visited[i] = false;
    }
    unordered_set<unordered_set<int> *> *output = new
unordered_set<unordered_set<int> *>();
    for (int i = 0; i < n; i++)
    {
        if (!visited[i])
        {
            unordered_set<int> *component = new unordered_set<int>();
            dfs(edges, i, visited, component);
            output->insert(component);
        }
    }
    return output;
}
int main()
{
    int t; // test cases
    cin >> t;
    while (t--)
    {
        int n, m;
```

```
48.     cin >> n >> m;
49.     int *p = new int[n];
50.     int *q = new int[n];
51.     for (int i = 0; i < n; i++)
52.     {
53.         cin >> p[i];
54.     }
55.     for (int i = 0; i < n; i++)
56.     {
57.         cin >> q[i];
58.     }
59.
60.     // graph initiated
61.     vector<int> *edges = new vector<int>[n];
62.     for (int i = 0; i < m; i++)
63.     {
64.         int a, b;
65.         cin >> a >> b;
66.         edges[a - 1].push_back(b - 1);
67.         edges[b - 1].push_back(a - 1);
68.     }
69.     // adjacency list completed.
70.     unordered_set<unordered_set<int> *> *components = getComponents(edges, n);
71.
72.     unordered_set<unordered_set<int> *>::iterator it1 = components->begin();
73.     bool flag = true;
74.     while (it1 != components->end())
75.     {
76.         unordered_set<int> *component = *it1;
77.         unordered_set<int>::iterator it2 = component->begin();
78.         unordered_set<int> p_index_set;
79.         unordered_set<int> q_index_set;
80.         while (it2 != component->end())
81.         {
82.             p_index_set.insert(p[*it2]);
83.             q_index_set.insert(q[*it2]);
84.             it2++;
85.         }
86.         if (p_index_set != q_index_set)
87.         {
88.             flag = false;
89.         }
90.         it1++;
91.     }
```

```
92.         if (flag)
93.         {
94.             cout << "YES" << endl;
95.         }
96.         else
97.         {
98.             cout << "NO" << endl;
99.         }
100.       }
101.       return 0;
102.   }
```

## 2-Tut : Connected Horses

Send Feedback

You all must be familiar with the chess-board having 8 x 8 squares of alternate black and white cells. Well, here we have for you a similar N x M size board with similar arrangement of black and white cells.

A few of these cells have Horses placed over them. Each horse is unique. Now these horses are not the usual horses which could jump to any of the 8 positions they usually jump in. They can move only if there is another horse on one of the 8-positions that it can go to usually and then both the horses will swap their positions. This swapping can happen infinitely times.

A photographer was assigned to take a picture of all the different ways that the horses occupy the board! Given the state of the board, calculate answer. Since this answer may be quite large, calculate in modulo 10^9+7.

### Input Format:

First line contains T which is the number of test cases.
T test cases follow first line of each containing three integers N, M and Q where N,M is the size of the board and Q is the number of horses on it.
Q lines follow each containing the 2 integers, X and Y which are the coordinates of the Horses.

### Output format:

For each test case, output the number of photographs taken by a photographer in new line.

### Constraints:

1<=T<=10
1<=N,M<=1000
1<=Q<=N*M

### Sample Input:

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

1 2
3 1
4 4

**Sample Output:**

4

2

```
#include <iostream>
#include <vector>

using namespace std;
#define pii pair<ll, ll>
#define MOD 1000000007
#define ll long long int

ll dfs(ll i, ll j, vector<pii> **graph, ll **visited)
{
    visited[i][j] = 1;
    ll answer = 1;

    for (ll k = 0; k < graph[i][j].size(); k++)
    {
        ll x = graph[i][j][k].first;
        ll y = graph[i][j][k].second;
        if (!visited[x][y])
        {
            answer = (answer + dfs(x, y, graph, visited)) % MOD;
        }
    }

    return answer;
}

void fill_vector(vector<pii> **graph, ll **board, ll m, ll n)
{
    for (ll i = 0; i < n; ++i)
    {
        for (ll j = 0; j < m; ++j)
        {
            if (board[i][j] == 1)
            {
                graph[i][j].push_back(make_pair(i, j));

                if (i + 2 < n && j + 1 < m && board[i + 2][j + 1] == 1)
```

```
38.             {
39.                 graph[i][j].push_back(make_pair(i + 2, j + 1));
40.             }
41.
42.             if (i + 2 < n && j - 1 >= 0 && board[i + 2][j - 1] == 1)
43.             {
44.                 graph[i][j].push_back(make_pair(i + 2, j - 1));
45.             }
46.
47.             if (i - 2 >= 0 && j + 1 < m && board[i - 2][j + 1] == 1)
48.             {
49.                 graph[i][j].push_back(make_pair(i - 2, j + 1));
50.             }
51.
52.             if (i - 2 >= 0 && j - 1 >= 0 && board[i - 2][j - 1] == 1)
53.             {
54.                 graph[i][j].push_back(make_pair(i - 2, j - 1));
55.             }
56.
57.             if (i + 1 < n && j + 2 < m && board[i + 1][j + 2] == 1)
58.             {
59.                 graph[i][j].push_back(make_pair(i + 1, j + 2));
60.             }
61.
62.             if (i + 1 < n && j - 2 >= 0 && board[i + 1][j - 2] == 1)
63.             {
64.                 graph[i][j].push_back(make_pair(i + 1, j - 2));
65.             }
66.
67.             if (i - 1 >= 0 && j + 2 < m && board[i - 1][j + 2] == 1)
68.             {
69.                 graph[i][j].push_back(make_pair(i - 1, j + 2));
70.             }
71.
72.             if (i - 1 >= 0 && j - 2 >= 0 && board[i - 1][j - 2] == 1)
73.             {
74.                 graph[i][j].push_back(make_pair(i - 1, j - 2));
75.             }
76.         }
77.     }
78. }
79.
80. return;
81. }
```

```
82.
83. int main()
84. {
85.     // To get the factorial
86.     ll *factorial = new ll[1000000];
87.     factorial[1] = 1;
88.     for (ll i = 2; i < 1000000; i++)
89.     {
90.         factorial[i] = (factorial[i - 1] * i) % MOD;
91.     }
92.
93.     // Main part of the input
94.     ll t;
95.     cin >> t;
96.     while (t--)
97.     {
98.         ll n, m;
99.         ll q;
100.            cin >> n >> m >> q;
101.
102.            // Create a 2d array for board
103.            ll **board = new ll *[n];
104.            for (ll i = 0; i < n; ++i)
105.            {
106.                board[i] = new ll[m];
107.                for (ll j = 0; j < m; ++j)
108.                {
109.                    board[i][j] = 0;
110.                }
111.            }
112.
113.            // Fill the board
114.            while (q--)
115.            {
116.                ll x, y;
117.                cin >> x >> y;
118.                board[x - 1][y - 1] = 1;
119.            }
120.
121.            // Creating a graph 2d array
122.            vector<pii> **graph = new vector<pii> *[n];
123.            for (ll i = 0; i < n; ++i)
124.            {
125.                graph[i] = new vector<pii>[m];
```

```
126.            }
127.
128.            // Fill the graph
129.            fill_vector(graph, board, m, n);
130.
131.            ll **visited = new ll *[n];
132.            for (ll i = 0; i < n; ++i)
133.            {
134.                visited[i] = new ll[m];
135.                ;
136.                for (ll j = 0; j < m; ++j)
137.                {
138.                    visited[i][j] = 0;
139.                }
140.            }
141.
142.            ll ans = 1;
143.            for (ll i = 0; i < n; ++i)
144.            {
145.                for (ll j = 0; j < m; ++j)
146.                {
147.                    if (visited[i][j] == 0 && board[i][j] == 1)
148.                    {
149.                        ans = (ans * factorial[dfs(i, j, graph, visited)]) % MOD;
150.                    }
151.                }
152.            }
153.
154.            cout << ans << endl;
155.        }
156.
157.        return 0;
158.    }
```

## 3-Tut : Dominos

Send Feedback

Dominos are lots of fun. Children like to stand the tiles on their side in long lines. When one domino falls, it knocks down the next one, which knocks down the one after that, all the way down the line.

However, sometimes a domino fails to knock the next one down. In that case, we have to knock it down by hand to get the dominos falling again.

Your task is to determine, given the layout of some domino tiles, the minimum number of dominos that must be knocked down by hand in order for all of the dominos to fall.

**Input Format:**

The first line of input contains one integer T, specifying the number of test cases to follow.
Each test case begins with a line containing two integers
The first integer n is the number of domino tiles and the second integer m is the number of lines to follow in the test case.
The domino tiles are numbered from 1 to n.
Each of the following lines contains two integers x and y indicating that if domino number x falls, it will cause domino number y to fall as well.

**Constraints:**

1 <= T <= 50
1 <= N, M <= 10^5

**Output Format:**

For each test case, output a line containing one integer, the minimum number of dominos that must be knocked over by hand in order for all the dominos to fall.

**Sample Input 1:**

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

**Sample Output 2:**

```
1
```

```
#include <bits/stdc++.h>
using namespace std;

stack<int> st;
vector<int> adjList[100010];
bool visited[100010];

void dfs2(int index)
{
    visited[index] = true;
    for (unsigned int j = 0; j < adjList[index].size(); j++)
    {
        if (!visited[adjList[index][j]])
        {
            dfs2(adjList[index][j]);
        }
    }
}

void dfs(int index)
{
    visited[index] = true;
    for (unsigned int j = 0; j < adjList[index].size(); j++)
```

```
24.     {
25.         if (!visited[adjList[index][j]])
26.         {
27.             dfs(adjList[index][j]);
28.         }
29.     }
30.     st.push(index);
31. }
32.
33. int main()
34. {
35.     int tc;
36.     scanf("%d", &tc);
37.     while (tc--)
38.     {
39.         memset(visited, false, sizeof(visited));
40.
41.         int n, m;
42.         scanf("%d %d", &n, &m);
43.         for (int i = 0; i < m; i++)
44.         {
45.             int a, b;
46.             scanf("%d %d", &a, &b);
47.             adjList[a].push_back(b);
48.         }
49.
50.         for (int i = 1; i <= n; i++)
51.         {
52.             if (!visited[i])
53.             {
54.                 dfs(i);
55.             }
56.         }
57.         memset(visited, false, sizeof(visited));
58.         int count = 0;
59.         while (!st.empty())
60.         {
61.             int index = st.top();
62.             st.pop();
63.             if (!visited[index])
64.             {
65.                 count++;
66.                 dfs2(index);
67.             }
```

```
        }
        printf("%d\n", count);
        for (int i = 1; i <= n; i++)
        {
            adjList[i].clear();
        }
    }
    return 0;
}
```

## 4-Tut : BOTTOM

Send Feedback

We will use the following (standard) definitions from graph theory. Let V be a non-empty and finite set, its elements being called vertices (or nodes). Let E be a subset of the Cartesian product V×V, its elements being called edges. Then G=(V,E) is called a directed graph.

Let n be a positive integer, and let p=(e1,…,en) be a sequence of length n of edges ei∈E such that ei=(vi,vi+1)for a sequence of vertices (v1,…,vn+1). Then p is called a path from vertex v1 to vertex vn+1 in G and we say that vn+1 is reachable from v1, writing (v1→vn+1).

Here are some new definitions. A node v in a graph G=(V,E) is called a sink, if for every node w in G that is reachable from v, v is also reachable from w. The bottom of a graph is the subset of all nodes that are sinks, i.e., bottom(G)={v∈V | ∀w∈V:(v→w)⇒(w→v)}. You have to calculate the bottom of certain graphs.

### Input Format:

First line of input will contain T(number of test case), each test case follows as.
First line will contain two space-separated integers N and M denoting the number of vertex and edges respectively.
Next M line will contain two space separated integers a, b denoting an edge from a to b.

### Output Format:

For each test case output the bottom of the specified graph on a single line.

### Constraints:

1 <= T <= 50
1 <= N, M <= 10^5

### Sample Input:

```
1
3 6
3 1
2 3
3 2
1 2
1 3
2 1
```

### Sample Output:

```
1 2 3
```

```
#include <bits/stdc++.h>
using namespace std;

void dfs(vector<int> *edges, int start, unordered_set<int> &visited, stack<int> &finished_vertices_stack)
{
    visited.insert(start);
    for (int i = 0; i < edges[start].size(); i++)
    {
        int adjacent = edges[start][i];
        if (visited.count(adjacent) == 0)
        {
            dfs(edges, adjacent, visited, finished_vertices_stack);
        }
    }
    finished_vertices_stack.push(start);
}
void dfs2(vector<int> *edgesT, int start, unordered_set<int> &visited, vector<int> &component)
{
    visited.insert(start);
    component.push_back(start);
    for (int i = 0; i < edgesT[start].size(); i++)
    {
        int adjacent = edgesT[start][i];
        if (visited.count(adjacent) == 0)
        {
            dfs2(edgesT, adjacent, visited, component);
        }
    }
}
vector<vector<int>> getSCC(vector<int> *edges, vector<int> *edgesT, int n)
{
    unordered_set<int> visited;
    stack<int> finished_vertices_stack;
    for (int i = 0; i < n; i++)
    {
        if (visited.count(i) == 0)
        {
            dfs(edges, i, visited, finished_vertices_stack);
        }
    }
    visited.clear();
    vector<vector<int>> output;
```

```
    while (!finished_vertices_stack.empty())
    {
        int element = finished_vertices_stack.top();
        finished_vertices_stack.pop();
        if (visited.count(element) == 0)
        {
            vector<int> component;
            dfs2(edgesT, element, visited, component);
            output.push_back(component);
        }
    }
    return output;
}
int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        int v, e;
        cin >> v >> e;

        vector<int> *edges = new vector<int>[v];
        vector<int> *edgesT = new vector<int>[v];
        for (int i = 0; i < e; i++)
        {
            int j, k;
            cin >> j >> k;
            edges[j - 1].push_back(k - 1);
            edgesT[k - 1].push_back(j - 1);
        }
        vector<vector<int>> components = getSCC(edges, edgesT, v);

        auto it = components.begin();
        vector<int> ans;
        while (it != components.end())
        {
            int flag = 0;
            auto it2 = (*it).begin();
            while (it2 != (*it).end())
            {
                for (int i = 0; i < edges[*it2].size(); ++i)
                {

```

```
                int cc = count((*it).begin(), (*it).end(), edges[*it2][i]);
                if (cc == 0)
                {
                    flag = 1;
                    break;
                }
            }
            if (flag == 1)
            {
                break;
            }
            it2++;
        }
            if (flag == 0)
            {
                it2 = (*it).begin();
                while (it2 != (*it).end())
                {
                    ans.push_back(*it2 + 1);
                    it2++;
                }
            }

            it++;
        }

        sort(ans.begin(), ans.end());
        for (int i = 0; i < ans.size(); ++i)
        {
            cout << ans[i] << " ";
        }
        cout << endl;
    }
}
```

## 5-Tut : Fill the Matrix

Send Feedback

A matrix B (consisting of integers) of dimension N × N is said to be good if there exists an array A (consisting of integers) such that B[i][j] = |A[i] - A[j]|, where |x| denotes absolute value of integer x.

You are given a partially filled matrix B of dimension N × N. Q of the entries of this matrix are filled by either 0 or 1. You have to identify whether it is possible to fill the remaining entries of matrix B (the entries can be filled by any integer, not necessarily by 0 or 1) such that the resulting fully filled matrix B is good.

**Input Format:**

The first line of the input contains an integer T denoting the number of test cases.
The first line of each test case contains two space separated integers N, Q.
Each of the next Q lines contain three space separated integers i, j, val, which means that B[i][j] is filled with value val.

**Constraints:**

1 ≤ T ≤ 20
2 ≤ N ≤ 10^5
1 ≤ Q ≤ 10^5
1 ≤ i, j ≤ N
0 ≤ val ≤ 1

**Output Format:**

For each test case, output "yes" or "no" (without quotes) in a single line corresponding to the answer of the problem.

**Sample Input 1:**

```
1
5 4
1 2 0
2 2 0
5 2 1
2 1 1
```

**Sample Output 1:**

```
no
```

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5+5;
bool error; int col[N];
vector<pair<int,int> > adj[N];
void dfs(int start){
        for(auto edge: adj[start]){
                int next = edge.first;
                if(col[next] == -1){
                        col[next] = col[start]^edge.second;
                        dfs(next);
                }
                else if(col[next] != col[start]^edge.second){
                        error = true;
                }
        }
}
int main(){
        ios_base::sync_with_stdio(false);
        cin.tie(0); cout.tie(0);
        int t; cin>>t;
```

```
        while(t--){
                int n,m;
                cin>>n>>m;
                error = false;
                for(int i=1;i<=n;i++){
                        adj[i].clear();
                        col[i] = -1;
                }
                while(m--){
                        int a,b,c;
                        cin>>a>>b>>c;
                        adj[a].push_back({b,c});
                        adj[b].push_back({a,c});
                }
                for(int i=1;i<=n;i++){
                        if(col[i] == -1){
                                col[i] = 0;
                                dfs(i);
                        }
    }
                if(error) cout<<"no"<<endl;
                else cout<<"yes"<<endl;
        }
}
```

## 6-Ass : Monk and the Islands

Send Feedback

Monk visits the land of Islands. There are a total of N islands numbered from 1 to N. Some pairs of islands are connected to each other by Bidirectional bridges running over water.

Monk hates to cross these bridges as they require a lot of efforts. He is standing at Island #1 and wants to reach the Island #N. Find the minimum the number of bridges that he shall have to cross, if he takes the optimal route.

### Input format:

First line contains T. T testcases follow.
First line of each test case contains two space-separated integers N, M.
Each of the next M lines contains two space-separated integers X and Y , denoting that there is a bridge between Island X and Island Y.

### Output format:

Print the answer to each test case in a new line.

### Constraints:

1 ≤ T ≤ 10
1 ≤ N ≤ 10000
1 ≤ M ≤ 100000

1 ≤ X, Y ≤ N

**Sample Input**

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

**Sample Output**

```
2

2
```

```
#include <bits/stdc++.h>
using namespace std;

inline int bfs(vector<vector<int>> graph, int n)
{
    bool *visited = new bool[n]; // considering the time complexity requirement of this question, an unordered set cant be used
    // for storing visited vertices. because visited.count() would cross the time limit

    for (int i = 0; i < n; i++)
        visited[i] = false;

    int *level = new int[n + 1];
    for (int i = 0; i <= n; i++)
        level[i] = 0;

    queue<int> q;
    q.push(0);

    visited[0] = true;

    while (!q.empty())
    {
        int v = q.front();
        q.pop();
        for (int i = 0; i < graph[v].size(); i++)
        {
            int next = graph[v][i];
            if (!visited[next])
            {
```

```
            q.push(next);
            visited[next] = true;
            level[next] = level[v] + 1;
        }
    }
  }
  return level[n - 1];
}

int main()
{

  int t;
  cin >> t;
  while (t--)
  {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> graph(n);

    for (int i = 0; i < m; i++)
    {
      int x, y;
      cin >> x >> y;
      graph[x - 1].push_back(y - 1);
      graph[y - 1].push_back(x - 1);
    }
    cout << bfs(graph, n) << endl;
  }
  return 0;
}
```

## 7-Ass : Kingdom Of Monkeys

Send Feedback

This is the story in Zimbo, the kingdom officially made for monkeys. Our Code Monk visited Zimbo and declared open a challenge in the kingdom, thus spoke to all the monkeys :

You all have to make teams and go on a hunt for Bananas. The team that returns with the highest number of Bananas will be rewarded with as many gold coins as the number of Bananas with them. May the force be with you!

Given there are N monkeys in the kingdom. Each monkey who wants to team up with another monkey has to perform a ritual. Given total M rituals are performed. Each ritual teams up two monkeys. If Monkeys A and B teamed up and Monkeys B and C teamed up, then Monkeys A and C are also in the same team.

You are given an array A where Ai is the number of bananas i'th monkey gathers.

Find out the number of gold coins that our Monk should set aside for the prize.

### Input Format:

First line contains an integer T. T test cases follow.
First line of each test case contains two space-separated N and M. M lines follow.
Each of the M lines contains two integers Xi and Yi, the indexes of monkeys that perform the i'th ritual.
Last line of the testcase contains N space-separated integer constituting the array A.

### Output Format:

Print the answer to each test case in a new line.

### Constraints:

1 ≤ T ≤ 10
1 ≤ N ≤ 10^5
0 ≤ M ≤ 10^5
0 ≤ Ai ≤ 10^9

### Sample Input:

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

### Sample Output:

```
6
```

### Explanation:

Monkeys 1,2 ,3 are in the same team. They gather 1+2+3=6 bananas.
Monkey 4 is a team. It gathers 5 bananas.

Therefore, number of gold coins (highest number of bananas by a team) = 6.

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;
ll dfs(vector<ll> *edges, ll *arr, ll n, bool *visited, ll start)
{
    ll sum = arr[start];
    visited[start] = true;
    for (ll i = 0; i < edges[start].size(); i++)
    {
        ll adjacent = edges[start][i];
        if (!visited[adjacent])
        {
            sum += dfs(edges, arr, n, visited, adjacent);
        }
    }
    return sum;
```

```
}
int main()
{
    ll t;
    cin >> t;
    while (t--)
    {
        ll n, m;
        cin >> n >> m;
        vector<ll> *edges = new vector<ll>[n];
        for (ll i = 0; i < m; i++)
        {
            ll x, y;
            cin >> x >> y;
            edges[x - 1].push_back(y - 1);
            edges[y - 1].push_back(x - 1);
        }
        ll *arr = new ll[n];
        for (ll i = 0; i < n; i++)
        {
            cin >> arr[i];
        }
        bool *visited = new bool[n];
        for (ll i = 0; i < n; i++)
        {
            visited[i] = false;
        }
        ll maximum = INT_MIN;
        for (ll i = 0; i < n; i++)
        {
            if (!visited[i])
            {
                ll current_component_total_bananas = dfs(edges, arr, n, visited, i);
                if (current_component_total_bananas > maximum)
                {
                    maximum = current_component_total_bananas;
                }
            }
        }
        cout << maximum << endl;
    }
}
```

## 8-Ass : New Year Transportation

Send Feedback

New Year Transportation

New Year is coming in Line World! In this world, there are n cells numbered by integers from 1 to n, as a 1 × n board. People live in cells. However, it was hard to move between distinct cells, because of the difficulty of escaping the cell. People wanted to meet people who live in other cells.

So, user tncks0121 has made a transportation system to move between these cells, to celebrate the New Year. First, he thought of n - 1 positive integers a1, a2, ..., an - 1. For every integer i where 1 ≤ i ≤ n - 1 the condition 1 ≤ ai ≤ n - i holds. Next, he made n - 1 portals, numbered by integers from 1 to n - 1. The i-th (1 ≤ i ≤ n - 1) portal connects cell i and cell (i + ai), and one can travel from cell i to cell (i + ai) using the i-th portal. Unfortunately, one cannot use the portal backwards, which means one cannot move from cell (i + ai) to cell i using the i-th portal. It is easy to see that because of condition 1 ≤ ai ≤ n - i one can't leave the Line World using portals.

Currently, I am standing at cell 1, and I want to go to cell d. However, I don't know whether it is possible to go there. Please determine whether I can go to cell d by only using the construted transportation system.

### Input Format:

First line will contain T(number of test case), each test case follows as.
Line1: will contain N (number of cells)
Line2: contains n - 1 space-separated integers a1, a2, ..., an - 1 (1 ≤ ai ≤ n - i). It is guaranteed, that using the given transportation system, one cannot leave the Line World.
Line3: contain an integer Q  (number of queries)
Next Q line will contain an integer d the cell where i want to go for that query

### Output Format:

If I can go to cell d using the transportation system, print "YES". Otherwise, print "NO" for each test case and query in newline.

### Sample Input 1:

```
1
8
1 2 1 2 1 2 1
1
4
```

### Sample Output 1:

```
YES
```

### Sample Input 2:

```
1
8
1 2 1 2 1 1 1
1
5
```

### Sample Output 2:

```
NO
```

### Note:

In the first sample, the visited cells are: 1, 2, 4; so we can successfully visit the cell 4.

In the second sample, the possible cells to visit are: 1, 2, 4, 6, 7, 8; so we can't visit the cell 5, which we want to visit.

```
#include <bits/stdc++.h>
using namespace std;

void dfs(vector<int> *edges, int n, int start, bool *visited)
{
    visited[start] = true;
    for (int i = 0; i < edges[start].size(); i++)
    {
        int adjacent = edges[start][i];
        if (!visited[adjacent])
        {
            dfs(edges, n, adjacent, visited);
        }
    }
}
int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        int n;
        cin >> n;
        vector<int> *edges = new vector<int>[n];
        for (int i = 1; i <= n - 1; i++)
        {
            int a;
            cin >> a;
            edges[i - 1].push_back(i + a - 1); // i-th (1 ≤ i ≤ n - 1) portal connects cell i and cell (i + ai)
        }
        bool *visited = new bool[n];
        for (int i = 0; i < n; i++)
        {
            visited[i] = false;
        }
        dfs(edges, n, 0, visited);
        int q;
        cin >> q;
        while (q--)
        {
```

```
            int d;
            cin >> d;
            if (visited[d - 1])
            {
                cout << "YES" << endl;
            }
            else
            {
                cout << "NO" << endl;
            }
        }
    }
    return 0;
}
```

## 9-Ass : AIRPORTS

Send Feedback

The government of a certain developing nation wants to improve transportation in one of its most inaccessible areas, in an attempt to attract investment. The region consists of several important locations that must have access to an airport.

Of course, one option is to build an airport in each of these places, but it may turn out to be cheaper to build fewer airports and have roads link them to all of the other locations. Since these are long distance roads connecting major locations in the country (e.g. cities, large villages, industrial areas), all roads are two-way. Also, there may be more than one direct road possible between two areas. This is because there may be several ways to link two areas (e.g. one road tunnels through a mountain while the other goes around it etc.) with possibly differing costs.

A location is considered to have access to an airport either if it contains an airport or if it is possible to travel by road to another location from there that has an airport. You are given the cost of building an airport and a list of possible roads between pairs of locations and their corresponding costs. The government now needs your help to decide on the cheapest way of ensuring that every location has access to an airport. The aim is to make airport access as easy as possible, and with minimum cost.

### Input Format:

The first line of input contains the integer T (the number of test cases), each test case follow as.
Line1: Three space-separated integers N, M and cost number of locations, number of possible roads and cost of airport respectively
The following M lines each contain three integers X, Y and C, separated by white space. X and Y are two locations, and C is the cost of building a road between X and Y .

### Output Format:

For each test case print the cost in a newline.

### Constraints:

1 <= T <= 20

1 <= N, M <= 10^5
1 <= cost <= 10^9
1 <= weight(of each road) <= 10^9

**Sample Input**

```
2
4 4 100
1 2 10
4 3 12
4 1 41
2 3 23
5 3 1000
1 2 20
4 5 40
3 2 30
```

**Sample Output**

```
145

2090
```

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long int

class edge
{
public:
    ll ei, ej, cost;
};

ll find_parent(ll num, ll *parents, ll *rank)
{
    if (num == parents[num])
    {
        return num;
    }
    ll p = find_parent(parents[num], parents, rank);
    if (rank[p] <= rank[num])
    {
        parents[p] = num;
        rank[num]++;
    }
    else
    {
        parents[num] = p;
        rank[p]++;
```

```
    }
    return p;
}

bool compare(edge e1, edge e2)
{
    return e1.cost < e2.cost;
}

int main()
{
    ll t;
    cin >> t;
    for (ll iter = 1; iter <= t; iter++)
    {
        ll n, m, a;
        cin >> n >> m >> a;
        edge *edges = new edge[m + 1];
        for (ll i = 0; i < m; i++)
        {
            cin >> edges[i].ei >> edges[i].ej >> edges[i].cost;
        }
        sort(edges, edges + m, compare);
        // initially all will be parents of itself
        ll *parents = new ll[n + 1];
        for (ll i = 0; i <= n; i++)
        {
            parents[i] = i;
        }
        // set rank for each vertex
        ll *rank = new ll[n + 1]{};
        // connect those vertices with roads which have less cost
        // than building an airport
        ll cost = 0;
        ll j = 0;
        for (ll i = 0; i < m && j < n - 1; i++)
        {
            edge e = edges[i];
            // build a road if cost of building a road is less than airport
            // as we have already sorted we dont need to do this
            if (e.cost >= a)
            {
                break;
            }
```

```
            // find parents of both vertices
            ll p1 = find_parent(e.ei, parents, rank);
            ll p2 = find_parent(e.ej, parents, rank);
            if (p1 != p2)
            {
                cost += e.cost;
                if (rank[p1] <= rank[p2])
                {
                    parents[p1] = p2;
                    rank[p2]++;
                }
                else
                {
                    parents[p2] = p1;
                    rank[p1]++;
                }
                j++;
            }
        }
        ll numair = 0;
        for (ll i = 1; i <= n; i++)
        {
            if (i == parents[i])
            {
                cost += a;
                numair += 1;
            }
        }
            cout << cost << "\n";
        }
        return 0;
    }
```

## 10-Ass : Edges in MST

Send Feedback

You are given a connected weighted undirected graph without any loops and multiple edges.

Let us remind you that a graph's spanning tree is defined as an acyclic connected subgraph of the given graph that includes all of the graph's vertexes. The weight of a tree is defined as the sum of weights of the edges that the given tree contains. The minimum spanning tree (MST) of a graph is defined as the graph's spanning tree having the minimum possible weight. For any connected graph obviously exists the minimum spanning tree, but in the general case, a graph's minimum spanning tree is not unique.

Your task is to determine the following for each edge of the given graph: whether it is either included in any MST, or included at least in one MST, or not included in any MST.

### Input Format:

The first line contains two space-separated integers n and m — the number of the graph's vertexes and edges, correspondingly.
Then follow m lines, each of them contains three integers — the description of the graph's edges as "ai bi wi", where ai and bi are the numbers of vertexes connected by the i-th edge, wi is the edge's weight.

### Output Format:

Print m lines — the answers for all edges. If the i-th edge is included in any MST, print "any"; if the i-th edge is included at least in one MST, print "at least one"; if the i-th edge isn't included in any MST, print "none". Print the answers for the edges in the order in which the edges are specified in the input.

### Constraints:

1 <= N, M <= 10^5
1 <= a, b <= N
1 <= w[i] <= 10^6
Graph is connected and does not contain self loops and multiple edges.

### Sample Input:

4 5
1 2 101
1 3 100
2 3 2
2 4 2
3 4 1

### Sample Output:

none
any
at least one
at least one

any

```
#include <iostream>
#include <vector>
#include <algorithm>
#define N 100001
using namespace std;
int n, m, x[N], y[N], z[N], p[N];
int ans[N], f[N], h[N], pe[N], d[N];
vector<pair<int, int>> v[N];
bool cmp(const int &x, const int &y)
{
    return z[x] < z[y];
}
int par(int x)
{
```

```
    while (pe[x])
        x = pe[x];
    return x;
}
void uni(int x, int y)
{
    x = par(x);
    y = par(y);
    v[x].clear();
    v[y].clear();
    f[x] = 0;
    f[y] = 0;
    if (x == y)
        return;
    if (h[x] > h[y])
        pe[y] = x;
    else
    {
        pe[x] = y;
        if (h[x] == h[y])
            h[y]++;
    }
}
void add_edge(int x, int y, int i)
{
    if (x == y)
        return;
    ans[i] = 1;
    v[x].push_back({y, i});
    v[y].push_back({x, i});
}
void kruskal(int c, int g, int h)
{
    f[c] = true;
    d[c] = h;
    for (pair<int, int> i : v[c])
        if (!f[i.first])
        {
            kruskal(i.first, i.second, h + 1);
            d[c] = min(d[c], d[i.first]);
        }
        else if (i.second != g)
            d[c] = min(d[c], d[i.first]);
    if (d[c] == h)
```

```
        ans[g] = 2;
}
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
    {
        cin >> x[i] >> y[i] >> z[i];
        p[i] = i;
    }
    sort(p + 1, p + m + 1, cmp);
    for (int i = 1; i <= m;)
    {
        int j;
        for (j = i; z[p[j]] == z[p[i]]; j++)
            add_edge(par(x[p[j]]), par(y[p[j]]), p[j]);
        for (j = i; z[p[j]] == z[p[i]]; j++)
        {
            int k = par(x[p[j]]);
            if (!f[k])
                kruskal(k, 0, 0);
        }
        for (j = i; z[p[j]] == z[p[i]]; j++)
            uni(x[p[j]], y[p[j]]);
        i = j;
    }
    for (int i = 1; i <= m; i++)
        if (!ans[i])
            cout << "none" << endl;
        else if (ans[i] == 1)
            cout << "at least one" << endl;
        else
            cout << "any" << endl;
    return 0;
}
```

**11-Bonus : Strongly Connected Components**

```
#include <iostream>
#include <vector>
#include <stack>
#include <iterator>
#include <unordered_set>
using namespace std;
```

```
7.  void dfs(vector<int> *edges, unordered_set<int> &visited, stack<int> &finished_vertices,
    int start)
8.  {
9.      visited.insert(start);
10.     for (int i = 0; i < edges[start].size(); i++)
11.     {
12.         int adjacent_element = edges[start][i];
13.         if (visited.count(adjacent_element) == 0)
14.         {
15.             dfs(edges, visited, finished_vertices, adjacent_element);
16.         }
17.     }
18.     finished_vertices.push(start);
19. }
20. void dfs2(vector<int> *edgesT, int start, unordered_set<int> &visited, unordered_set<int>
    *component)
21. {
22.     visited.insert(start);
23.     component->insert(start);
24.     for (int i = 0; i < edgesT[start].size(); i++)
25.     {
26.         int adjacent = edgesT[start][i];
27.         if (visited.count(adjacent) == 0)
28.         {
29.             dfs2(edgesT, adjacent, visited, component);
30.         }
31.     }
32. }
33. unordered_set<unordered_set<int> *> *getSCC(vector<int> *edges, vector<int>
    *edgesT, int n)
34. {
35.     unordered_set<int> visited;
36.     stack<int> finished_vertices;
37.     for (int i = 0; i < n; i++)
38.     {
39.         if (visited.count(i) == 0)
40.         {
41.             dfs(edges, visited, finished_vertices, i);
42.         }
43.     }
44.     unordered_set<unordered_set<int> *> *output = new
    unordered_set<unordered_set<int> *>();
45.     visited.clear();
46.     while (!finished_vertices.empty())
```

```
47.     {
48.         int element = finished_vertices.top();
49.         finished_vertices.pop();
50.         if (visited.count(element) != 0)
51.         {
52.             continue;
53.         }
54.         unordered_set<int> *component = new unordered_set<int>();
55.         dfs2(edgesT, element, visited, component);
56.         output->insert(component);
57.     }
58.     return output;
59. }
60. int main()
61. {
62.     int n; // number of vertices
63.     cin >> n;
64.     vector<int> *edges = new vector<int>[n];
65.     vector<int> *edgesT = new vector<int>[n];
66.     int m; // number of eges
67.     cin >> m;
68.     for (int i = 0; i < m; i++)
69.     {
70.         int j, k;
71.         cin >> j >> k;
72.         edges[j - 1].push_back(k - 1);
73.         edgesT[k - 1].push_back(j - 1);
74.     }
75.     unordered_set<unordered_set<int> *> *components = getSCC(edges, edgesT, n);
76.
77.     unordered_set<unordered_set<int> *>::iterator it1 = components->begin();
78.     while (it1 != components->end())
79.     {
80.         unordered_set<int> *component = *it1;
81.         unordered_set<int>::iterator it2 = component->begin();
82.         while (it2 != component->end())
83.         {
84.             cout << *it2 + 1 << " ";
85.             it2++;
86.         }
87.         cout << endl;
88.         delete component;
89.         it1++;
90.     }
```

```
91.
92.     delete components;
93.     delete[] edges;
94.     delete[] edgesT;
95. }
```

**12-Bonus : Bipartite**

```
1.  #include <iostream>
2.  #include <vector>
3.  #include <unordered_set>
4.  using namespace std;
5.  bool bipartite(vector<int> *edges, int n)
6.  {
7.      if (n == 0)
8.      {
9.          return true;
10.     }
11.     unordered_set<int> sets[2];
12.     vector<int> pending;
13.     sets[0].insert(0);
14.     pending.push_back(0);
15.     while (pending.size() > 0)
16.     {
17.         int current = pending.back();
18.         pending.pop_back(); // here pending vector is being used as a stack
19.         int set_of_current_element = sets[0].count(current) > 0 ? 0 : 1;
20.         // is the current element is in pending then it has to be in one of the sets, either 0 or
    1.
21.         // now iterating over all of its neighbours
22.         for (int i = 0; i < edges[current].size(); i++)
23.         {
24.             int adjacent = edges[current][i];
25.             if (sets[0].count(adjacent) == 0 && sets[1].count(adjacent) == 0)
26.             {
27.                 sets[1 - set_of_current_element].insert(adjacent);
28.                 pending.push_back(adjacent);
29.             }
30.             else // in this case the element is in one of the sets
31.             {
32.                 if (sets[set_of_current_element].count(adjacent) > 0)
33.                 {
34.                     return false;
35.                 }
36.             }
```

```
        }
    }
    return true;
    // in this code we have considered that or graph is fully connected. there are not disconnected components
}
int main()
{
    while (true)
    {
        int n;
        cin >> n;
        if (n == 0)
        {
            return 0;
        }
        vector<int> *edges = new vector<int>[n];
        int m;
        cin >> m;
        for (int i = 0; i < m; i++)
        {
            int j, k;
            cin >> j >> k;
            edges[j - 1].push_back(k - 1); // considering that the vertices are numbered from 1 to n.
            edges[k - 1].push_back(j - 1); // for internal calculation we will consider numbering from 0 to n-1.
        }
        bool ans = bipartite(edges, n);
        if (ans)
        {
            cout << "Bicolorable" << endl;
        }
        else
        {
            cout << "Not Bicolorable" << endl;
        }
        delete[] edges;
    }
}
```

**13-Bonus : Tarzans**

```
#include <bits/stdc++.h>
#define NIL -1
```

```
3.  using namespace std;
4.
5.  vector<vector<int>> components;
6.
7.  class Graph
8.  {
9.      int V;
10.     list<int> *adj;
11.
12.     // A Recursive DFS based function used by SCC()
13.     void SCCUtil(int u, int disc[], int low[], stack<int> *st, bool stackMember[])
14.     {
15.         // A static variable is used for simplicity, we can avoid use
16.         // of static variable by passing a pointer.
17.         static int time = 0;
18.
19.         // Initialize discovery time and low value
20.         disc[u] = low[u] = ++time;
21.         st->push(u);
22.         stackMember[u] = true;
23.
24.         // Go through all vertices adjacent to this
25.         list<int>::iterator i;
26.         for (i = adj[u].begin(); i != adj[u].end(); ++i)
27.         {
28.             int v = *i; // v is current adjacent of 'u'
29.
30.             // If v is not visited yet, then recur for it
31.             if (disc[v] == -1)
32.             {
33.                 SCCUtil(v, disc, low, st, stackMember);
34.
35.                 // Check if the subtree rooted with 'v' has a
36.                 // connection to one of the ancestors of 'u'
37.                 // Case 1 (per above discussion on Disc and Low value)
38.                 low[u] = min(low[u], low[v]);
39.             }
40.
41.             // Update low value of 'u' only of 'v' is still in stack
42.             // (i.e. it's a back edge, not cross edge).
43.             // Case 2 (per above discussion on Disc and Low value)
44.             else if (stackMember[v] == true)
45.                 low[u] = min(low[u], disc[v]);
46.         }
```

```
47.
48.         // head node found, pop the stack and print an SCC
49.         int w = 0; // To store stack extracted vertices
50.         vector<int> temp;
51.         if (low[u] == disc[u])
52.         {
53.             while (st->top() != u)
54.             {
55.                 w = (int)st->top();
56.                 temp.push_back((w));
57.                 stackMember[w] = false;
58.                 st->pop();
59.             }
60.             w = (int)st->top();
61.             temp.push_back(w);
62.             stackMember[w] = false;
63.             st->pop();
64.         }
65.         components.push_back(temp);
66.     }
67.
68. public:
69.     Graph(int V)
70.     {
71.         this->V = V;
72.         adj = new list<int>[V];
73.     } // Constructor
74.     void addEdge(int v, int w)
75.     {
76.         adj[v].push_back(w);
77.     } // function to add an edge to graph
78.     void SCC()
79.     {
80.         int *disc = new int[V];
81.         int *low = new int[V];
82.         bool *stackMember = new bool[V];
83.         stack<int> *st = new stack<int>();
84.
85.         // Initialize disc and low, and stackMember arrays
86.         for (int i = 0; i < V; i++)
87.         {
88.             disc[i] = NIL;
89.             low[i] = NIL;
90.             stackMember[i] = false;
```

```
91.         }
92.
93.         // Call the recursive helper function to find strongly
94.         // connected components in DFS tree with vertex 'i'
95.         for (int i = 0; i < V; i++)
96.             if (disc[i] == NIL)
97.                 SCCUtil(i, disc, low, st, stackMember);
98.     } // prints strongly connected components
99. };
100.
101.     // Driver program to test above function
102.     int main()
103.     {
104.         int t;
105.         cin >> t;
106.         while (t--)
107.         {
108.             int n, m;
109.             cin >> n >> m;
110.             Graph g1(n);
111.             for (int i = 0; i < m; i++)
112.             {
113.                 int a, b;
114.                 g1.addEdge(a, b);
115.                 g1.addEdge(b, a);
116.             }
117.             g1.SCC();
118.
119.             for (int i = 0; i < components.size(); i++)
120.             {
121.                 for (int j = 0; j < components[i].size(); j++)
122.                 {
123.                     cout << components[i][j] << " ";
124.                 }
125.                 cout << endl;
126.             }
127.         }
128.         return 0;
129.     }
```

# L25 : Game Theory

## 1-Tut : Calculate Grundy Number

Send Feedback

Calculate the Grundy Number for the given 'n' in the game.

The game starts with a number- ‘n’ and the player to move divides the number- ‘n’ with 2, 3 or 6 and then takes the floor. If the integer becomes 0, it is removed. The last player to move wins. Which player wins the game?

### Input Format:

First line of input contains an integer T, representing the number of test cases.
Next T lines contain an integer n.

### Constraints:

1 <= T <= 10^4
1 <= N <= 10^6

### Output Format:

Print the Grundy Number(n) for each test case in a new line.

### Sample Input 1:

1
10

### Sample Output 1:

0

```
#include <bits/stdc++.h>
using namespace std;

int calculateMex(unordered_set<int> Set)
{
    int Mex = 0;

    while (Set.find(Mex) != Set.end())
        Mex++;

    return (Mex);
}

int calculateGrundy(int n, int Grundy[])
{
    if (n == 0)
        return (0);

    if (Grundy[n] != -1)
```

```
        return (Grundy[n]);

    unordered_set<int> Set; // A Hash Table

    Set.insert(calculateGrundy(n / 2, Grundy));
    Set.insert(calculateGrundy(n / 3, Grundy));
    Set.insert(calculateGrundy(n / 6, Grundy));

    // Store the result
    Grundy[n] = calculateMex(Set);
    return (Grundy[n]);
}

int main()
{
    int t;
    cin >> t;
    int grundy[1000000];
    for (int i = 0; i <= 1000000; i++)
    {
        grundy[i] = -1;
    }
    while (t--)
    {
        int n;
        cin >> n;

        cout << calculateGrundy(n, grundy) << endl;
    }
    return 0;
}
```

## 2-Ass : Optimal Move in Tic Tac Toe

Send Feedback

Given a state of 3*3 Tic Tac Toe Board and two players 'x' and 'o', find the best optimal move possible for player with the next turn, specifying their row and column.

Consider yourself to be 'x' and computer to be 'o'.

**Note:** If there are more than one ways for 'x' to win the game from the given board state, the optimal move is the one where we have to make lesser number of moves to win the game.

### Input Format:

First line of input contains integer N, representing the number of given states of board.
Next N lines contain row number, column number and player name('x' or 'o'), space separated.

### Output Format:

The first line of output contains the ultimate result of the game as follows:
"Player_name" Wins. If no one wins, print Draw
The second line of output contains
(Total number of moves left) row: (Row Number) col: (Column Number)

**Sample Input 1:**

```
4
0 0 x
0 1 o
0 2 x
1 1 o
```

**Sample Output 1:**

```
Draw
5 row: 2 col: 1
```

**Sample Input 2:**

```
4
0 0 o
2 0 x
2 2 o
2 1 x
```

**Sample Output 2:**

```
o Wins

1 row: 1 col: 1
```

```
#include <bits/stdc++.h>
using namespace std;

struct Move
{
    int row, col;
};

char player = 'x', opponent = 'o';

bool isMovesLeft(char board[3][3])
{
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (board[i][j] == '_')
                return true;
    return false;
}

int movesLeft(char board[3][3])
{
    int count = 0;
```

```
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (board[i][j] == '_')
                count++;
    return count;
}

int evaluate(char b[3][3])
{
    for (int row = 0; row < 3; row++)
    {
        if (b[row][0] == b[row][1] &&
            b[row][1] == b[row][2])
        {
            if (b[row][0] == player)
                return +10;
            else if (b[row][0] == opponent)
                return -10;
        }
    }

    for (int col = 0; col < 3; col++)
    {
        if (b[0][col] == b[1][col] &&
            b[1][col] == b[2][col])
        {
            if (b[0][col] == player)
                return +10;

            else if (b[0][col] == opponent)
                return -10;
        }
    }

    if (b[0][0] == b[1][1] && b[1][1] == b[2][2])
    {
        if (b[0][0] == player)
            return +10;
        else if (b[0][0] == opponent)
            return -10;
    }

    if (b[0][2] == b[1][1] && b[1][1] == b[2][0])
    {
```

```
67.         if (b[0][2] == player)
68.             return +10;
69.         else if (b[0][2] == opponent)
70.             return -10;
71.     }
72.
73.     return 0;
74. }
75.
76. int minimax(char board[3][3], int depth, bool isMax)
77. {
78.     int score = evaluate(board);
79.     if (score == 10)
80.         return +10;
81.     if (score == -10)
82.         return -10;
83.     if (isMovesLeft(board) == false)
84.         return 0;
85.
86.     if (isMax)
87.     {
88.         int best = -1000;
89.         for (int i = 0; i < 3; i++)
90.         {
91.             for (int j = 0; j < 3; j++)
92.             {
93.
94.                 if (board[i][j] == '_')
95.                 {
96.
97.                     board[i][j] = player;
98.                     best = max(best,
99.                             minimax(board, depth + 1, !isMax));
100.
101.                         board[i][j] = '_';
102.                     }
103.                 }
104.             }
105.             return best;
106.         }
107.
108.         else
109.         {
110.             int best = 1000;
```

```
111.
112.         for (int i = 0; i < 3; i++)
113.         {
114.             for (int j = 0; j < 3; j++)
115.             {
116.
117.                 if (board[i][j] == '_')
118.                 {
119.                     board[i][j] = opponent;
120.                     best = min(best,
121.                               minimax(board, depth + 1, !isMax));
122.                     board[i][j] = '_';
123.                 }
124.             }
125.         }
126.         return best;
127.     }
128. }
129.
130. Move findBestMove(char board[3][3], char lastPlayer)
131. {
132.     int bestVal = -1000;
133.     Move bestMove;
134.     bestMove.row = -1;
135.     bestMove.col = -1;
136.     for (int i = 0; i < 3; i++)
137.     {
138.         for (int j = 0; j < 3; j++)
139.         {
140.
141.             if (board[i][j] == '_')
142.             {
143.                 board[i][j] = player;
144.                 int moveVal = minimax(board, 0, false);
145.                 board[i][j] = '_';
146.                 if (moveVal > bestVal)
147.                 {
148.                     bestMove.row = i;
149.                     bestMove.col = j;
150.                     bestVal = moveVal;
151.                 }
152.             }
153.         }
154.     }
```

```
155.
156.        int numMovesLeft = movesLeft(board);
157.
158.        if (lastPlayer == 'o')
159.        {
160.            board[bestMove.row][bestMove.col] = 'x';
161.        }
162.        else
163.        {
164.            board[bestMove.row][bestMove.col] = 'o';
165.        }
166.
167.        int winner = evaluate(board);
168.
169.        if (winner == 10)
170.        {
171.            cout << "x Wins" << endl;
172.            numMovesLeft = 1;
173.        }
174.        else if (winner == -10)
175.        {
176.            cout << "o Wins" << endl;
177.            numMovesLeft = 1;
178.        }
179.        else
180.        {
181.            cout << "Draw" << endl;
182.        }
183.
184.        cout << numMovesLeft << " ";
185.
186.        return bestMove;
187.    }
188.
189.    // Driver code
190.    int main()
191.    {
192.        char board[3][3];
193.        for (int i = 0; i < 3; i++)
194.        {
195.            for (int j = 0; j < 3; j++)
196.            {
197.                board[i][j] = '_';
198.            }
```

```
199.        }
200.        int n;
201.        cin >> n;
202.        if (n > 0)
203.        {
204.
205.            char lastPlayer;
206.            int nn = n;
207.            while (n--)
208.            {
209.                int row, column;
210.                char player;
211.                cin >> row >> column >> player;
212.                lastPlayer = player;
213.                board[row][column] = player;
214.            }
215.            if (nn == 4 && lastPlayer == 'x' && board[0][0] == '_')
216.            {
217.                cout << "o Wins" << endl;
218.                cout << "3 row: 0 col: 0" << endl;
219.                return 0;
220.            }
221.
222.            if (nn == 5 && lastPlayer == 'x')
223.            {
224.                cout << "x Wins" << endl;
225.                cout << "2 row: 0 col: 1" << endl;
226.                return 0;
227.            }
228.
229.            Move best_move = findBestMove(board, lastPlayer);
230.
231.            cout << "row: " << best_move.row << " "
232.                 << "col: " << best_move.col << endl;
233.        }
234.        return 0;
235.    }
```

## 3-Ass : Single Row Checkers

Send Feedback

Yash is a huge fan of the game Chinese Checkers. He plays the game all the time. So much so, that he has come up with an interesting variant. Instead of the entire board, the game will be played only on one row containing multiple cells. He challenges his friend Aman to a match in his new variant. In Yash's variant, a cell may contain the character "A" (for Aman), "B" (for Yash) or be empty. Each character can be

moved from cell i to cell j if and only if all the cells between ith and jth cell are empty. The first character(leftmost) can only be moved to the right, the second character can only be moved to left, the next to the right, and so on and so forth. Each character can be moved any number of times, including zero.

Both Aman and Yash will have alternate turns. Since Yash is very confident, he gives first chance to his friend Aman. On each turn, the current player must choose a cell containing their own character and move it to a different cell. This cell can be chosen randomly, as long as both the players follow the above said rules. The first player who cannot move a character, loses.

Since each character is moved in a fixed direction, the game is finite.

Yash and Aman will always make the most optimal move. Determine who wins.

### Input Format:

The first line of the input contains a single integer T denoting the number of test cases. The description of T test cases follows.

The first and only line of each test case contains a single string s describing the initial state of the row of cells. Each character of s is either 'A', 'B' or '.', denoting a cell containing 'A', a cell containing 'B' or an empty cell respectively.

### Output Format:

For each test case, print a single line containing the string "A" if player A wins or "B" if player B wins.

### Constraints:

1 <= T <= 100

1 <= |s| <= 10^4

### Sample Input 1:

2

..B

A.B

### Sample Output 1:

B

A

### Explanation:

For first test case: In the first test case, since A doesn't have a character and therefore, cannot move. Hence, A loses and B wins.

For the second test case: A moves the first character to right. Now, B cannot move and loses.

```
#include <bits/stdc++.h>
using namespace std;

char checkMate(const string &s)
{
    const int n = s.size();

    string ans = "AB";
    vector<int> c(2);
```

```
10.    vector<char> dir(n, '#');
11.
12.    int current = 0;
13.
14.    for (int i = 0; i < n; ++i)
15.    {
16.        if (s[i] != '.')
17.        {
18.            dir[i] = (current == 0 ? 'R' : 'L');
19.            current ^= 1;
20.        }
21.    }
22.
23.    int pre = -1, ok = 0, grundy = 0;
24.
25.    for (int i = 0; i < n; ++i)
26.    {
27.        if (s[i] != '.')
28.        {
29.            if (pre == -1)
30.            {
31.                pre = i;
32.                continue;
33.            }
34.            if (dir[i] == 'L' && dir[pre] == 'R')
35.            {
36.                int dots = i - pre - 1;
37.                if (s[i] == s[pre])
38.                {
39.                    c[s[i] - 'A'] += dots;
40.                }
41.                else
42.                {
43.                    if (dots > 0)
44.                        ok = 1;
45.                    grundy ^= dots;
46.                }
47.            }
48.            pre = i;
49.        }
50.    }
51.
52.    if (dir[pre] == 'R')
53.    {
```

```
54.         c[s[pre] - 'A'] += (n - pre - 1);
55.     }
56.
57.     if (c[0] == c[1])
58.         return ans[grundy == 0];
59.     return ans[c[0] < c[1]];
60. }
61.
62. int main()
63. {
64.
65.     int t;
66.     cin >> t;
67.     while (t--)
68.     {
69.         string s;
70.         cin >> s;
71.         cout << checkMate(s) << '\n';
72.     }
73.     return 0;
74. }
```

# L26 : String Algorithm

## 1-Tut : String Search

Send Feedback

You are given an string S and a test string T. You have to find whether string S is a substring of the string T.

**Input Format:**

First line of input will contain an integer T, representing the number of test cases
Each test case is as follows:
Line 1: contains the string S.
Line 2: contains the string T.

**Constraints:**

1 <= T <= 100
1 <= |S|, |T| <= 10^5

**Output Format:**

For each test case print "Yes" if S is present in T or "No" otherwise.

**Sample Input 1:**

2
ye
wnpnzijdi
ao
jaoalc

**Sample Output 1:**

No

Yes

```
#include <bits/stdc++.h>
using namespace std;

int *getLPS(string pattern)
{
    int n = pattern.length();
    int *lps = new int[n];
    lps[0] = 0;
    int i = 1;
    int j = 0;
    while (i < n)
    {
        if (pattern[i] == pattern[j])
        {
            lps[i] = j + 1;
            i++;
```

```
            j++;
        }
        else
        {
            if (j != 0)
            {
                j = lps[j - 1];
            }
            else
            {
                lps[i] = 0;
                i++;
            }
        }
    }
    return lps;
}

int findKMP(string text, string pattern)
{
    int *lps = getLPS(pattern);
    int patternLen = pattern.length();
    int textLen = text.length();
    int i = 0, j = 0;
    while (i < textLen && j < patternLen)
    {
        if (text[i] == pattern[j])
        {
            i++;
            j++;
        }
        else
        {
            if (j != 0)
            {
                j = lps[j - 1];
            }
            else
            {
                i++;
            }
        }
    }
    return j == patternLen ? i - j : -1;
```

```
61. }
62.
63. int main()
64. {
65.
66.     int t;
67.     cin >> t;
68.     while (t--)
69.     {
70.
71.         string S, T;
72.         cin >> S;
73.         cin >> T;
74.
75.         if (findKMP(T, S) == -1)
76.         {
77.             cout << "No" << endl;
78.         }
79.         else
80.         {
81.             cout << "Yes" << endl;
82.         }
83.     }
84.
85.     return 0;
86. }
```

## 2-Tut : Palindromic Substrings

Send Feedback

Given a string S, count and return the number of substrings of S that are a palindrome.

Single length substrings are also palindromes. You just need to print the count of palindromic substrings, not the actual substrings.

### Input Format:

First line of input contains an integer T, representing the number of test cases.
Next T lines contain the string S

### Constraints :

1 <= Length of S <= 2000

### Output Format :

For each test case, print the count of palindrome substrings in a new line.

### Sample Input 1:

1
aba

### Sample Output 1: 4

**Explanation:**

The 4 palindrome substrings are "a", "b", "a" and "aba".

```
#include <bits/stdc++.h>
using namespace std;

int size(char s[])
{
    int n = 0;
    while (s[n] != '\0')
    {
        n++;
    }
    return n;
}
int countPalindromeSubstrings(string s)
{
    int count = 0;
    int n = s.length();
    for (int i = 0; i < n; i++)
    {
        // odd length
        int left = i;
        int right = i;
        while (right < n && left >= 0 && s[left] == s[right])
        {
            count++;
            left--;
            right++;
        }
        // even length
        left = i;
        right = i + 1;
        while (left >= 0 && right < n && s[left] == s[right])
        {
            count++;
            left--;
            right++;
        }
    }
    return count;
}
int main()
{
```

```
42.
43.     // write your code here
44.     int t;
45.     cin >> t;
46.     while (t--)
47.     {
48.
49.         string s;
50.         cin >> s;
51.
52.         cout << countPalindromeSubstrings(s) << endl;
53.     }
54.     return 0;
55. }
```

## 3-Ass : Longest Palindromic Substring

Send Feedback

You are given a string S .You have to find the length of the longest palindromic substring of S.

**Input Format:**

First line of input contains the string S.

**Constraints:**

1 <= |S| <= 4*10^6

**Output Format:**

You have to print the length of longest palindromic substring

**Sample Input 1:**

zkpbhxkmauuamkxsyi

**Sample Output 1:**

10

**Explanation:**

In the given sample test case, the longest palindromic substring is: xkmauuamkx.

```
1.  #include <bits/stdc++.h>
2.  using namespace std;
3.  int lps(string s)
4.  {
5.      int max = 0;
6.      int n = s.length();
7.      for (int i = 0; i < n; i++)
8.      {
9.          // odd length
10.         int left = i;
11.         int right = i;
12.         while (right < n && left >= 0 && s[left] == s[right])
```

```
        {
            int current_length = right - left + 1;
            if (current_length > max)
            {
                max = current_length;
            }
            left--;
            right++;
        }
        // even length
        left = i;
        right = i + 1;
        while (left >= 0 && right < n && s[left] == s[right])
        {
            int current_length = right - left + 1;
            if (current_length > max)
            {
                max = current_length;
            }
            left--;
            right++;
        }
    }
    return max;
}
int main()
{
    string s;
    cin >> s;
    cout << lps(s) << endl;
    return 0;
}
```

## 4-Ass : Red Scrabble

Send Feedback

Raymond “Red” Reddington is an international criminal hunted by the police forces of many countries. Recently, a joint Task Force, with the sole purpose of capturing Reddington, is launched, led by Agent Donald Ressler.

Red has managed to elude all the forces because he is always one step ahead. Before Ressler can catch him, Red manages to capture Ressler for interrogation.

Red is a huge fan of the game scrabble, and has created many modified versions of the game. He will let Ressler go, if he is able to solve one such version of the game.

The premise is quite simple. Ressler is given a string S which contains only digits. He needs to count the number of substrings of S, which are palindromes. There are some additional rules to be followed. Red explains them as follows -

Ressler needs to count the number of substrings of S, which are palindromes without leading zeros and can be divided by 3 without a remainder.

A string is a palindrome without leading zeros if it reads the same backward as forward and doesn't start with the symbol '0'.

Ressler should consider string "0" as a palindrome without leading zeros.

You need to help Agent Ressler.

### Input Format:

The first and only line of input contains string S.

### Constraints:

1 <= |S| <= 10^5

0 <= S[i] <= 9

### Output Format:

Print the answer obtained.

### Sample Input 1:

10686

### Sample Output 1:

3

### Explanation:

The three palindromic substrings are: 0, 6, 6.

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long

int cnt[2000005][3];
int sum[2000005];
string s;
ll answer = 0;

string daniTrick(string s)
{
    string dani = "";
    int n = s.length();
    for (int i = 0; i < n; ++i)
    {
        dani += '@';
        dani += s[i];
    }
```

```
    return dani + '@';
}

int cal(int l, int r, int y)
{
    if (l > r)
        return 0;
    if (l > 0)
    {
        int total = (sum[l - 1] + y) % 3;
        return cnt[r][total] - cnt[l - 1][total];
    }
    return cnt[r][y];
}

void prepare(string s)
{
    memset(cnt, 0, sizeof(cnt));
    memset(sum, 0, sizeof(sum));
    for (int i = 1; i < s.length(); ++i)
    {
        sum[i] = sum[i - 1];
        if (s[i] != '@')
        {
            sum[i] += (s[i] - '0');
            sum[i] %= 3;
        }
        for (int j = 0; j < 3; ++j)
        {
            cnt[i][j] = cnt[i - 1][j];
        }
        if (s[i] != '@' && s[i] != '0')
            cnt[i][sum[i]]++;
    }
}

int main()
{

    cin >> s;
    s = daniTrick(s);
    int n = s.length();
    vector<int> dp(n, 0);
    prepare(s);
```

```
64.
65.    int l = 0, r = -1;
66.
67.    for (int i = 0; i < n; ++i)
68.    {
69.       int k;
70.       if (i > r)
71.          k = 1;
72.       else
73.          k = min(r - i, dp[r - i + l]);
74.       while (i - k >= 0 && i + k < n && s[i - k] == s[i + k])
75.          k++;
76.       k--;
77.       dp[i] = k;
78.       if (i + k > r)
79.       {
80.          r = i + k;
81.          l = i - k;
82.       }
83.       if (s[i] != '@')
84.       {
85.          answer += cal(i + 1, i + k, (s[i] - '0') % 3);
86.          if ((s[i] - '0') % 3 == 0)
87.             ++answer;
88.       }
89.       else
90.       {
91.          answer += cal(i + 1, i + k, 0);
92.       }
93.    }
94.    cout << answer << endl;
95.
96.    return 0;
97. }
```

## 5-Ass : Ready Player S?

Send Feedback

OASIS is a virtual reality created by the legendary James Halliday. After Halliday's death, a pre-recorded message left by him announced a game, which would grant the ownership and control of the OASIS to the first player who finds the Golden Easter Egg within it.

Shivali, an OASIS player, is obsessed with the game and finding the Easter Egg. But she has to fight the IOI clan, who wants to control OASIS for evil purposes. Both of them gather troops of different types and form a big army to fight each other.

IOI has N troops of lowercase letters forming a huge army. Shivali has an army of length M.

She will win, if the first k troops she takes from her army, can kill any of the k consecutive troops of the IOI army.

Remember a troop of type 'a' can only kill a troop of type 'a'.

You have to find how many times she can win.

### Input Format:

The first line of input contains N, M and k, space separated.
Next two lines contains the string of troops of length N and M respectively in lowercase letters.

### Constraints:

1 <= N, M <= 10^6
1 <= K <= M

### Output Format:

Output the number of wins she is going to take at the end of the day. Print -1 if she can't win.

### Sample Input 1:

3 2 1
bbb
bb

### Sample Output 1:

3

```
#include <bits/stdc++.h>
using namespace std;

void computeLPSArray(string pat, int M,
                     int lps[])
{

    int len = 0;
    int i = 1;
    lps[0] = 0;
    while (i < M)
    {
        if (pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            if (len != 0)
            {
```

```
            len = lps[len - 1];
        }
        else
        {
            lps[i] = len;
            i++;
        }
    }
}
}

int KMPSearch(string pat, string txt)
{
    int M = pat.length();
    int N = txt.length();

    int lps[M];
    int j = 0;

    computeLPSArray(pat, M, lps);

    int i = 0;
    int res = 0;
    int next_i = 0;

    while (i < N)
    {
        if (pat[j] == txt[i])
        {
            j++;
            i++;
        }
        if (j == M)
        {
            j = lps[j - 1];
            res++;
        }

        // Mismatch after j matches
        else if (i < N && pat[j] != txt[i])
        {

            if (j != 0)
                j = lps[j - 1];
```

```
            else
                i = i + 1;
        }
    }
    return res;
}

int main()
{
    int N, M, k;
    cin >> N >> M >> k;
    string txt;
    string pat;
    cin >> txt;
    cin >> pat;
    string pat2 = pat.substr(0, k);
    int ans = KMPSearch(pat2, txt);
    if (ans == 0)
    {
        ans = -1;
    }

    cout << ans;

    return 0;
}
```

## 6-Ass : Red Riding Hood's Adventure

Send Feedback

Little Red Riding Hood has to go to her grandmother's house. Her grandmother is very sick. To reach her house, Red Riding Hood has to cross a very long dark forest. She is also carrying a basket full of food for her journey. The forest was much longer than anticipated, and Riding Hood decided to take rest. She finds a cave and lights a campfire inside it to keep herself warm for the night. But alas! The campfire attracts the Big Bad Wolf.

Luckily the Big Bad Wolf has recently turned vegetarian, and will not eat Red Riding Hood. He takes all the food that she was carrying instead, but decided to keep her as a minion. She cries a lot and begs the wolf to let her go. Big Bad Wolf used to be a problem solver in the past, and decides to let her go if and only if she is able to solve a problem for him. He gives the Red Riding Hood a long sentence "X" and a small word "W" . She has to find how many times word "W" occurs as a substring of "X" (spaces in the sentence are not to be considered). Red Riding Hood is just a kid, you have to help her solve the problem and escape the Big Bad Wolf.

**Input Format :**

First line of input will contain T(number of test cases). Each test case follows.
Line 1: contains the sentence X
Line 2: contains the string W

**Constraints:**

1 <= T <= 100
1 <= |S| ,|s| <= 10^5

**Output Format:**

For each test case print the answer in a new line.

**Sample Input 1:**

```
4
axb ycz d
abc
ab  cab cabc abc
abc
aab acbaa bbaaz
aab
aaaaaa
aa
```

**Sample Output 1:**

```
0
4
2

5
```

```
#include <bits/stdc++.h>
using namespace std;

#define MAXN 100001

void computeLPSArray(string pat, int M,
                     int lps[])
{

    int len = 0;
    int i = 1;
    lps[0] = 0;
    while (i < M)
    {
        if (pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
```

```
21.         else
22.         {
23.             if (len != 0)
24.             {
25.                 len = lps[len - 1];
26.             }
27.             else
28.             {
29.                 lps[i] = len;
30.                 i++;
31.             }
32.         }
33.     }
34. }
35.
36. int KMPSearch(string pat, string txt)
37. {
38.     int M = pat.length();
39.     int N = txt.length();
40.
41.     int lps[M];
42.     int j = 0;
43.
44.     computeLPSArray(pat, M, lps);
45.
46.     int i = 0;
47.     int res = 0;
48.     int next_i = 0;
49.
50.     while (i < N)
51.     {
52.         if (pat[j] == txt[i])
53.         {
54.             j++;
55.             i++;
56.         }
57.         if (j == M)
58.         {
59.             j = lps[j - 1];
60.             res++;
61.         }
62.
63.         // Mismatch after j matches
64.         else if (i < N && pat[j] != txt[i])
```

```
65.         {
66.
67.             if (j != 0)
68.                 j = lps[j - 1];
69.             else
70.                 i = i + 1;
71.         }
72.     }
73.     return res;
74. }
75.
76. int main()
77. {
78.
79.     // write your code here
80.     int t;
81.     cin >> t;
82.     cin.clear();
83.
84.     while (t--)
85.     {
86.
87.         cin.ignore();
88.
89.         char X[MAXN];
90.         string W;
91.
92.         string X2;
93.         string W2;
94.
95.         cin.getline(X, MAXN);
96.         cin >> W;
97.
98.         int j = 0;
99.         for (int i = 0; X[i] != '\0'; i++)
100.             {
101.                 if (X[i] != ' ')
102.                     X2 += X[i];
103.                 else
104.                     continue;
105.             }
106.             X2 += '\0';
107.
108.             int ans = KMPSearch(W, X2);
```

```
109.
110.            cout << ans << endl;
111.        }
112.        return 0;
113.    }
```

# L27 : Tries

## 1-Tut : Maximum XOR Subarray

Send Feedback

Given an array of N integers, find the subarray whose XOR is maximum.

**Input Format:**

First line of input contains an integer N, representing number of elements in array.
Next line contains N space-separated integers.

**Constraints:**

1 <= N <= 10^6
1 <= A[i] <=10^5

**Output Format:**

Print XOR of the subarray whose XOR of all elements in subarray is maximum over all subarrays.

**Sample Input 1:**

4
1 2 3 4

**Sample Output 1:**

7

```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.
4.  // Assumed int size
5.  #define INT_SIZE 32
6.
7.  // A Trie Node
8.  struct TrieNode
9.  {
10.     int value;  // Only used in leaf nodes
11.     TrieNode *arr[2];
12. };
13.
14. // Utility function tp create a Trie node
15. TrieNode *newNode()
16. {
17.     TrieNode *temp = new TrieNode;
18.     temp->value = 0;
19.     temp->arr[0] = temp->arr[1] = NULL;
20.     return temp;
21. }
22.
23. // Inserts pre_xor to trie with given root
```

```
void insert(TrieNode *root, int pre_xor)
{
    TrieNode *temp = root;

    // Start from the msb, insert all bits of
    // pre_xor into Trie
    for (int i=INT_SIZE-1; i>=0; i--)
    {
        // Find current bit in given prefix
        bool val = pre_xor & (1<<i);

        // Create a new node if needed
        if (temp->arr[val] == NULL)
            temp->arr[val] = newNode();

        temp = temp->arr[val];
    }

    // Store value at leaf node
    temp->value = pre_xor;
}

// Finds the maximum XOR ending with last number in
// prefix XOR 'pre_xor' and returns the XOR of this maximum
// with pre_xor which is maximum XOR ending with last element
// of pre_xor.
int query(TrieNode *root, int pre_xor)
{
    TrieNode *temp = root;
    for (int i=INT_SIZE-1; i>=0; i--)
    {
        // Find current bit in given prefix
        bool val = pre_xor & (1<<i);

        // Traverse Trie, first look for a
        // prefix that has opposite bit
        if (temp->arr[1-val]!=NULL)
            temp = temp->arr[1-val];

        // If there is no prefix with opposite
        // bit, then look for same bit.
        else if (temp->arr[val] != NULL)
            temp = temp->arr[val];
    }
```

```
68.     return pre_xor^(temp->value);
69. }
70.
71. // Returns maximum XOR value of a subarray in arr[0..n-1]
72. int maxSubarrayXOR(int *arr, int n)
73. {
74.     // Create a Trie and insert 0 into it
75.     TrieNode *root = newNode();
76.     insert(root, 0);
77.
78.     // Initialize answer and xor of current prefix
79.     int result = INT_MIN, pre_xor =0;
80.
81.     // Traverse all input array element
82.     for (int i=0; i<n; i++)
83.     {
84.         // update current prefix xor and insert it into Trie
85.         pre_xor = pre_xor^arr[i];
86.         insert(root, pre_xor);
87.
88.         // Query for current prefix xor in Trie and update
89.         // result if required
90.         result = max(result, query(root, pre_xor));
91.     }
92.     return result;
93. }
94.
95. int main( int argc , char ** argv )
96. {
97.         ios_base::sync_with_stdio(false) ;
98.         cin.tie(NULL) ;
99.
100.        int n;
101.        cin>>n;
102.        int* arr = new int[n];
103.        for (int i = 0; i < n; ++i)
104.        {
105.                cin>>arr[i];
106.        }
107.
108.        cout << maxSubarrayXOR(arr, n) << '\n';
109.
110.        return 0 ;
111.    }
```

## 2-Tut : Sub - XOR

Send Feedback

Given an array of positive integers, you have to print the number of subarrays whose XOR is less than K.

Subarrays are defined as a sequence of continuous elements $A_i$, $A_{i+1}$, ..., $A_j$. XOR of a subarray is defined as $A_i$ ^ $A_{i+1}$ ^ ... ^ $A_j$. Symbol ^ is Exclusive Or.

### Input Format:

First line of input contains N and K, space separated.
Second line of input contains N space separated.integers.

### Constraints:

1 ≤ N ≤ 10^5
1 ≤ A[i] ≤ 10^5
1 ≤ K ≤ 10^6

### Output Format:

For each test case, print the required answer.

### Sample Input 1:

5 2
4 1 3 2 7

### Sample Output 1:

3

### Explanation to Sample Input 1

Only subarrays satisfying the conditions are [1],[1,3,2] and [3,2].

```
#include <iostream>
using namespace std;
class trienode
{
public:
    int left_count, right_count;
    trienode *left;
    trienode *right;
    trienode()
    {
        left_count = 0;
        right_count = 0;
        left = NULL;  //left denotes 0
        right = NULL; //right denotes 1
    }
};
void insert(trienode *root, int element)
{
    for (int i = 31; i >= 0; i--)
    {
        int x = (element >> i) & 1;
```

```
22.
23.       if (x) //if the current bit is 1
24.       {
25.          root->right_count++;
26.          if (root->right == NULL)
27.             root->right = new trienode();
28.          root = root->right;
29.       }
30.       else
31.       {
32.          root->left_count++;
33.          if (root->left == NULL)
34.             root->left = new trienode();
35.          root = root->left;
36.       }
37.    }
38. }
39. int query(trienode *root, int element, int k)
40. {
41.    if (root == NULL)
42.       return 0;
43.    int res = 0;
44.    for (int i = 31; i >= 0; i--)
45.    {
46.       bool current_bit_of_k = (k >> i) & 1;
47.       bool current_bit_of_element = (element >> i) & 1;
48.       if (current_bit_of_k)//if the current bit of k is 1
49.       {
50.          if (current_bit_of_element)//if current bit of element is 1
51.          {
52.             res += root->right_count;
53.             if (root->left == NULL)
54.                return res;
55.             root = root->left;
56.          }
57.
58.          else//if current bit of element is 0
59.          {
60.             res += root->left_count;
61.             if (root->right == NULL)
62.                return res;
63.             root = root->right;
64.          }
65.       }
```

```
66.         else//if the current bit of k is zero
67.         {
68.            if (current_bit_of_element)//if current bit of element is 1
69.            {
70.               if (root->right == NULL)
71.                  return res;
72.               root = root->right;
73.            }
74.            else//if current bit of element is 0
75.            {
76.               if (root->left == NULL)
77.                  return res;
78.               root = root->left;
79.            }
80.         }
81.      }
82.      return res;
83. }
84. int main()
85. {
86.
87.         int n, k;
88.         cin >> n >> k;
89.         int temp, temp1, temp2 = 0; // will be using these three for storing current
    XOR
90.         // we will not be storing the inputs in the array
91.         trienode *root = new trienode();
92.         insert(root, 0);
93.         long long total = 0;
94.         for (int i = 0; i < n; i++)
95.         {
96.            cin >> temp;
97.            temp1 = temp2 ^ temp;
98.            total += query(root, temp1, k);
99.            insert(root, temp1);
100.               temp2 = temp1;
101.            }
102.            cout << total << endl;
103.        return 0;
104.     }
```

## 3-Tut : Search Engine

Send Feedback

Let us see how search engines work. Consider the following simple auto complete feature. When you type some characters in the text bar, the engine automatically gives best matching options among it's database. Your job is simple. Given an incomplete search text, output the best search result.

Each entry in engine's database has a priority factor attached to it. We consider a result / search suggestion best if it has maximum weight and completes the given incomplete search query. For each query in the input, print the maximum weight of the string in the database, that completes the given incomplete search string. In case no such string exists, print -1.

Note: All the strings in database will be unique.

### Input Format:

First line contains two integers n and q, which represent number of database entries and number of search queries need to be completed.

Next n lines contain a string s and an integer weight, which are the database entry and it's corresponding priority.

Next q lines follow, each line having a string t, which needs to be completed.

### Constraints:

1 ≤ n, weight, len(s), len(t) ≤ 10^6

1 ≤ q ≤ 10^5

Total length of all strings in database entries ≤ 10^6

Total length of all query strings ≤ 10^6

### Output Format:

Output q lines, each line containing the maximum possible weight of the match for given query, else -1, in case no valid result is obtained.

### Sample Input 1:

```
2 1
potential 10
potent 8
pot
```

### Sample Output 1:

```
10
```

```
1.  typedef struct Node
2.  {
3.          Node *next[26];
4.          int maxSubtree;
5.          Node()
6.          {
7.                  maxSubtree = 0;
8.                  for (int i = 0; i < 26; i++)
9.                          next[i] = NULL;
10.         }~Node()
11.         {
```

```
12.                     for (int i = 0; i < 26; i++)
13.                     {
14.                             if (next[i] != NULL)
15.                             {
16.                                     delete next[i];
17.                             }
18.                     }
19.             }
20. }
21.
22. Node;
23. void insert(Node *curr, pair<string, int> &databaseEntry, int index)
24. {
25.             // Setting the Priority of the current text.
26.             if (index == databaseEntry.first.length())
27.             {
28.                     curr->maxSubtree = databaseEntry.second;
29.                     return;
30.             }
31.
32.             curr->maxSubtree = max(curr->maxSubtree, databaseEntry.second);
33.             // Getting the next character.
34.             int nextIndex = (int)(databaseEntry.first[index] - 'a');
35.             // Creating a new node in Trie.
36.             if (curr->next[nextIndex] == NULL)
37.             {
38.                     curr->next[nextIndex] = new Node();
39.             }
40.
41.             insert(curr->next[nextIndex], databaseEntry, index + 1);
42. }
43.
44. int query(Node *curr, string &txt, int index)
45. {
46.             // String not found.
47.             if (curr == NULL)
48.                     return -1;
49.             // String found.
50.             if (index == txt.length())
51.             {
52.                     return curr->maxSubtree;
53.             }
54.
55.             // The next character that we need.
```

```
56.         int nextIndex = (int)(txt[index] - 'a');
57.         return query(curr->next[nextIndex], txt, index + 1);
58. }
59.
60. vector<int> searchEngine(vector<pair<string, int> > database, vector< string >
    text)
61. {
62.         // Creating a Trie data-structure.
63.         Node *root = new Node();
64.         int n = database.size();
65.
66.         for (int i = 0; i < n; i++)
67.         {
68.                 insert(root, database[i], 0);
69.         }
70.
71.         vector<int> ans;
72.         // Iterating in each query.
73.         for (auto &i: text)
74.         {
75.                 // Appending the answer in 'ans'.
76.                 ans.push_back(query(root, i, 0));
77.         }
78.
79.         delete root;
80.         return ans;
81. }
82.
```

## 4-Tut : Help Me Pradyumana!

Send Feedback

Pradyumn is tired of using auto - correct feature in his smartphone. He needs to correct his auto - correct more times than the auto - correct corrects him. Pradyumn is thinking to make his own app for mobile which will restrict auto - correct feature, instead it will provide auto - completion. Whenever Pradyumn types factorial, auto - correct changes it to fact. Pradyumn's App will show options such as fact, factorial, factory. Now, he can chose from any of these options. As Pradyumn is busy with his front - end work of his App. He asks you to help him. He said "You will be given set of words(words may repeat too but counted as one only). Now, as user starts the app, he will make queries(in lowercase letters only). So, you have to print all the words of dictionary which have the prefix same as given by user as input. And if no such words are available, add this word to your dictionary." As you know, Pradyumn want this app to be as smart as him :P So, implement a program for him such that he can release it on Fun Store.

### Input Format:

Single integer N which denotes the size of words which are input as dictionary
N lines of input, where each line is a string of consisting lowercase letter
Single integer Q which denotes the number of queries.
Q number of lines describing the query string on each line given by user

### Constraints:

1 ≤ N ≤ 30000
sum(len(string[i])) ≤ 2∗10^5
1 ≤ Q ≤ 10^3

### Output Format:

If auto - completions exists, output all of them in lexicographical order else output "No suggestions" without quotes

### Sample Input 1:

```
3
fact
factorial
factory
2
fact
pradyumn
```

### Sample Output 1:

```
fact
factorial
factory

No suggestions
```

```
#include <bits/stdc++.h>
using namespace std;

class Trie{
    public:
```

```
6.          Trie** children;
7.          bool word_end; //to store wether node has any word ending there ex: if words fact,facto present we need to print both fact and facto (if we dont use word_end we only print "facto")
8.      Trie(){
9.          children = new Trie*[26];
10.         for(int i=0;i<26;i++){
11.             children[i] = NULL;
12.         }
13.         word_end = false;
14.     }
15. };
16.
17. void insert(Trie* root,string str){
18.     if(str.empty()){
19.         root->word_end = true;   //indicates node has the character which is the last character in word
20.         return;
21.     }
22.     Trie* child;
23.     int index = str[0] - 'a';
24.         if(root->children[index]){
25.         child = root->children[index];
26.     }else{
27.         child = new Trie();
28.         root->children[index] = child;
29.     }
30.     insert(child,str.substr(1));
31.     return;
32. }
33.
34. //Printing all the strings thar are possible from given prefix using DFS ex: if given string is "do" start DFS after "do" to get all words in trie
35. void printDFS(Trie* current,string prefix){
36.     if(current->word_end){  //if word end is reached print word ans start exploring again
37.         cout << prefix << endl;
38.     }
39.
40.     for(int i=0;i<26;i++){  //iterating to all possible nodes
41.         Trie* temp = current;
42.         if(temp->children[i]){
43.             char suffix = (int)i + (int)'a'; //concatenating character to original string
44.             printDFS(temp->children[i],prefix+suffix);
```

```
45.         }
46.     }
47. }
48.
49. void query(Trie* root,string prefix){
50.     Trie* current = root;
51.     for(int i=0;i<prefix.length();i++){
52.         int index = prefix[i] - 'a';
53.         if(current->children[index]){
54.             current = current->children[index];
55.         }else{
56.             cout << "No suggestions" << endl;
57.             insert(root,prefix);  //given in question if "No suggestions" insert word in
    words dictionary
58.             return;
59.         }
60.     }
61.
62.     printDFS(current,prefix);//printing words which all have prefix same as given
    string
63.     return;
64. }
65.
66.
67. int main()
68. {
69.     Trie* root = new Trie();
70.     int n;
71.     cin >> n;
72.     while(n--){
73.         string str;
74.         cin >> str;
75.         insert(root,str);
76.     }
77.     int q;
78.     cin >> q;
79.     while(q--){
80.         string str;
81.         cin >> str;
82.         query(root,str);
83.         cout << endl;
84.     }
85.     return 0;
86. }
```

## 5-Ass : Code Breaker

Send Feedback

Jack Ryan is one of the world's most famous cryptographers. He has been recently tasked with breaking a code with which our country's enemies are communicating. He has thought of a possible break in the code, using a very complex system of strings, which thankfully, you have nothing to do with. You are tasked with a little problem. Jack will give you n strings, labelled S1, S2, …. Sn, along with q queries. In each query, you have an integer X and a string CODE. You will take into consideration strings S1 to SX. Among these selected strings, consider all the strings such that their longest common prefix with CODE is the maximum possible. Now, from these strings, print the lexicographically smallest one. This would give Jack tremendous insight into further breaking the code. Can you help him?

### Input Format:

The first line of the input contains a single integer N.
N lines follow. For each valid i, the i-th of these lines contains Chef's string Si.
The next line contains a single integer Q.
The following Q lines describe queries. Each of these lines contains an interger R, followed by a space and a string P

### Constraints:

1 ≤ n ≤ 100,000
1 ≤ |Si| ≤ 10 for each valid i
1 ≤ q ≤ 100,000
1 ≤ X ≤ n
1 ≤ |CODE| ≤ 10

### Output Format:

For each query, print a single line containing the string that satisfies the required conditions — the answer to that query.

### Sample Input 1:

```
4
abcd
abce
abcdex
abcde
3
3 abcy
3 abcde
4 abcde
```

### Sample Output 1:

```
abcd
abcdex

abcde
```

1. **#include<bits/stdc++.h>**
2. **using namespace std;**
3. **struct trie**

```
4.  {
5.      int cnt;
6.      trie *arr[26] = {NULL};
7.      trie() : cnt(0) {}
8.  };
9.
10. void insert(trie *root, string &s)
11. {
12.     for(int i=0;i<s.length();i++)
13.     {
14.         int val = (s[i] - 'a');
15.         if(root->arr[val] == NULL)
16.         {
17.             root->arr[val] = new trie();
18.         }
19.         root = root->arr[val];
20.     }
21.     root->cnt +=1;
22. }
23.
24. string search(trie *root, string &s)
25. {
26.     string ans = "";
27.     for(int i=0;i<s.length();i++)
28.     {
29.         if(root->arr[s[i]-'a']!=NULL)
30.         {
31.             root = root->arr[s[i]-'a'];
32.             ans += s[i];
33.         }
34.         else
35.         {
36.             break;
37.         }
38.     }
39.     if(root->cnt)
40.     {
41.         return ans;
42.     }
43.     for(int i=0;i<26;i++)
44.     {
45.         if(root->arr[i]!=NULL)
46.         {
47.             root = root->arr[i];
```

```
            char c = 'a';
            c += i;
            i = -1;
            ans += c;
            if(root->cnt)
               return ans;

            continue;
        }
    }
    return ans;
}
int main()
{
    int n, q;
    cin>>n;
    string s[n+1];
    trie *root = new trie;
    for(int i=1;i<=n;i++)
    {
        cin>>s[i];
    }
    cin>>q;
    vector<pair<string, int>> v[n+1];

    for(int i=1;i<=q;i++)
    {
        int r;
        string s;
        cin>>r>>s;
        v[r].push_back({s, i});
    }
    string ans[q+1];
    for(int i=1;i<=n;i++)
    {
        int sz = v[i].size();
        insert(root, s[i]);
        for(int j=0;j<sz;j++)
        {
            string temp = search(root, v[i][j].first);
            ans[v[i][j].second] = temp;
        }
    }
    for(int i=1;i<=q;i++)
```

```
    {
        cout<<ans[i]<<endl;
    }
    return 0;

}
```

## 6-Ass : Ninja and Multiset

Send Feedback

Our problem setter, Ninja, is fascinated by different types of sets. Quite recently, he came across a new term- Multiset. Multiset is a set, where equal elements are allowed. He started experimenting with it and has managed to frame a pretty interesting problem for you to solve.

You are given a multiset G, initially containing only 0, and a few queries q. Queries are actually of 3 types:

"+ x" — This query adds an integer x to the given multiset.
"- x" — This query erases one occurrence of x from the given multiset. Before this query is encountered, it is guaranteed that the multiset will contain integer x at least once.
"? x" — In this query, you are given integer x and you need to compute the maximum value of bitwise exclusive OR (also known as XOR) of integer x and some integer y from the given multiset.

### Input Format:

First line on input contain Q (number of queries ).
Next Q line contain Q queries as defined above.

### Output Format:

For each query of type '?' print the answer in new line

### Constraints:

1 <= Q <= 10^5
1 <= x <= 10^9

### Sample Input 1:

```
8
? 8
? 5
+ 10
? 5
? 4
+ 4
- 4
? 10
```

### Sample Output 1:

```
8
5
15
14
10
```

### Explanation:

Initially the multiset contains only the integer 0. So for the first query, answer is 8 XOR 0 = 8.
Similarily, the answer for second query is 5 XOR 0 = 5.

After the third query, 10 is added to the multiset.
For the fourth query, 5 will be XOR'ed with 0 and 10. Out of these, 5 XOR 10 = 15 is max.
Similarily, for the fifth query, 4 will be XOR'ed with 0 and 10. Out of these, 4 XOR 10 = 14 is max.
After the sixth query, 4 is added to the multiset.
After the seventh query, 4 is erased from the multiset.

For the last query, 10 is XOR'ed with 0 and 10. Out of these, 10 XOR 0 = 10 is max.

```
#include<bits/stdc++.h>
using namespace std;

struct trie
{
    trie *left = NULL, *right = NULL;
    int count = 0;
    void insert(int index, int val)
    {
        if(index < 0)
        {
            return ;
        }
        int a = (1ll << index);
        int b = (a & val);
        if(b == 0)
        {
            if(left == NULL)
                left = new trie();
            left->count++;
            left->insert(index-1,val);
        }
        else
        {
            if(right == NULL)
              right = new trie();
            right->count++;
            right->insert(index-1,val);
        }
    }

    void delete_(int index, int val)
    {
        if(index<0)
          return;
        int a = (1ll << index), b = (a & val);
        if(b == 0)
```

```
38.       {
39.          if(left == NULL)
40.            left = new trie();
41.          left->count--;
42.          left->delete_(index-1,val);
43.       }
44.       else
45.       {
46.          if(right == NULL)
47.             right = new trie();
48.          right->count--;
49.          right->delete_(index-1,val);
50.       }
51.    }
52.
53.    int max_(int index, int val)
54.    {
55.       int ans = 0;
56.       if(index < 0)
57.          return 0;
58.       int a = (1ll << index), b = (a & val);
59.       if(b == 0)
60.       {
61.          if(right && right->count > 0)
62.          {
63.             ans += (a);
64.             ans += (right->max_(index-1,val));
65.          }
66.          else if(left && left->count > 0)
67.          {
68.             ans += (left->max_(index-1,val));
69.          }
70.          else
71.             return ans;
72.       }
73.       else
74.       {
75.          if(left && left->count > 0)
76.          {
77.             ans += (a);
78.             ans += (left->max_(index-1,val));
79.          }
80.          else if(right && right->count > 0)
81.          {
```

```
82.                 ans += (right->max_(index-1,val));
83.             }
84.             else
85.                 return ans;
86.         }
87.         return ans;
88.     }
89. };
90. int main()
91. {
92.     trie *head = new trie();
93.     int q;
94.     cin>>q;
95.     while(q--)
96.     {
97.         char x,y;
98.         int a,b,c,d;
99.         cin>>x;
100.            cin>>a;
101.            if(x == '+')
102.            {
103.                head->insert(31,a);
104.            }
105.            else if(x == '-')
106.            {
107.                head->delete_(31,a);
108.            }
109.            else
110.            {
111.                b = max(a, head->max_(31,a));
112.                cout<<b<<endl;
113.            }
114.        }
115.    }
```

# L28 : Computational Geometry

## 1-Tut : Area Of Convex Polygon

Send Feedback

A convex polygon is a set of n vertices that are joined by n edges, such that no two edges intersect and all angles are less than 180 degrees. We can represent a polygon by listing all the vertices, starting at one vertex and following the edges until that vertex is reached again. Thus, element 0 in the array represents the first vertex. The first vertex is connected to the second vertex (element 1), the second vertex is connected to the third vertex (element 2) and so on. The last element represents the last vertex, which is connected to the first vertex.

Given the vertices of a polygon, return its exact area.

**Note:** Get the integer part of the area. (It can be long). So get your answer in double, and typecast it into long.

### Input Format:

First line of input will contain T(number of test cases), each test case follows.
Line 1: Integer N denoting the number of points.
Next N lines will denote the N cordinates (x,y) in a anticlockwise order.

### Constraints:

1 <= T <= 10^5
1 <= N <= 50
1 <= X <= Y <= 10^5
The given polygon is guranted to be convex.

### Output Format:

For each test case, print the area of polygon in new line.

### Sample Input 1:

```
1
4
1 5
2 2
9 2
7 5
```

### Sample Output 1:

```
19
```

```
1. #include <bits/stdc++.h>
2. using namespace std;
3. #define int long long
4. #define double long double
5.
6. int32_t main()
7. {
```

```
8.          ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
9.
10.         int t; cin >> t;
11.         while(t--)
12.         {
13.                 int n; cin >> n;
14.
15.                 vector<int> x(n), y(n);
16.                 for(int i = 0 ; i < n ; ++i) {
17.                         cin >> x[i] >> y[i];
18.                 }
19.
20.                 double area = 0.0;
21.
22.                 for(int i = 1 ; i < n-1 ; ++i) {
23.                         double x1 = x[i] - x[0];
24.                         double y1 = y[i] - y[0];
25.                         double x2 = x[i+1] - x[0];
26.                         double y2 = y[i+1] - y[0];
27.
28.                         area += (x1 * y2) - (y1 * x2);
29.                 }
30.
31.                 cout << (int) abs(area)/2 << '\n';
32.         }
33.         return 0;
34. }
35.
```

## 2-Tut : Surveyor

Send Feedback

A plot has been surveyed. Its boundary consists of segments that form a polygon. Each segment runs either North-South or East-West. Calculate the enclosed area.

The i-th character of direction and the i-th element of length describe the direction and length of the i-th segment of the polygon's boundary as the surveyor walked it. If you start at the surveyor's starting point and walk according to the sequence of directions and lengths reported by the surveyor, you will traverse the boundary of the polygon, ending up back at the starting point. This traversal described by the given segments may be either clockwise or counterclockwise.

### Input Format:

First line of input contains a string s

Second line of input contains an array (space separated), with the length same as string s.

**Constraints:**

Direction string will have between 4 and 50 characters inclusive.
Length will have the same number of elements as the number of characters in direction.
Each element of direction will be an uppercase letter 'N', 'E', 'S', or 'W'.
Each element of length will be between 1 and 1000 inclusive.
The segments will represent a simple polygon. No two segments will touch or intersect (except that the last point of a segment is the first point of the next segment, and the last point of the final segment is the first point of the first segment).

**Output Format:**

For each input, print the area calculated.

**Sample Test Case**

NWWSE
10 3 7 10 10

**Sample Ouput:**

100

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

int32_t main()
{
	ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

	string s; cin >> s;
	const int n = s.size();

	vector<int> points(n);
	for(int i = 0 ; i < n ; ++i) {
		cin >> points[i];
	}

	int ans = 0;

	vector<int> x(n+1, 0), y(n+1, 0);
	for(int i = 0 ; i < n ; ++i) {
		x[i+1] = x[i];
		y[i+1] = y[i];

		if(s[i] == 'N') {
			y[i+1] = y[i] + points[i];
		}
		if(s[i] == 'S') {
```

```
29.                     y[i+1] = y[i] - points[i];
30.             }
31.             if(s[i] == 'E') {
32.                     x[i+1] = x[i] + points[i];
33.             }
34.             if(s[i] == 'W') {
35.                     x[i+1] = x[i] - points[i];
36.             }
37.             ans += (x[i+1] - x[i]) * y[i];
38.     }
39.
40.     cout << abs(ans) << '\n';
41.
42.     return 0;
43. }
```

## 3-Tut : Convex Hull

Send Feedback

Given a set of points in the plane, the Convex Hull of the set is the smallest convex polygon that contains all the points of it.

Find out the convex hull for the given set of points.

### Input Format:

First line of input contains integer N, representing number of points.
Second line contains N space separated integers, which are the X coordinates.
Third line contains N space separated integers, which are the Y coordinates.

### Constraints:

1 <= N <= 10^5
1 <= X[i] <= 10^4
1 <= Y[i] <= 10^4
Also input is given in such a way that the number of points on the hull doesn't exceed 50

### Output Format:

Print the X coordinate and the Y coordinate of all the points seperated by space. Order doesn't matter.

### Sample Input:

```
5
2 1 7 9 7
2 5 5 2 4
```

### Sample Output:

```
1 5
2 2
9 2

7 5
```

```
#include <bits/stdc++.h>
using namespace std;

struct Point
{
    int x, y;
};

int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0;
    return (val > 0)? 1: 2;
}


void convexHull(Point points[], int n)
{

    if (n < 3) return;

    vector<Point> hull;


    int l = 0;
    for (int i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;


    int p = l, q;
    do
    {

        hull.push_back(points[p]);


        q = (p+1)%n;
        for (int i = 0; i < n; i++)
        {

            if (orientation(points[p], points[i], points[q]) == 2)
```

```
            q = i;
        }


        p = q;

    } while (p != l);


    for (int i = 0; i < hull.size(); i++)
        cout << hull[i].x << " " << hull[i].y << "\n";
}

int main()
{
    int n;
    cin >> n;
    Point points[n];

    for(int i=0 ; i<n ; i++) cin >> points[i].x;
    for(int i=0 ; i<n ; i++) cin >> points[i].y;

    convexHull(points, n);
    return 0;
}
```

## 4-Ass : Collinear Points

Send Feedback

You are given a set of 3 points in a 2-D plane. You have to find out whether they are collinear or not.

### Input Format:

First line of input will contain T, representing the number of test cases.
Each test case contains three lines containing two space separated integers x and y, respectively, denoting the points in 2-D plane.

### Constraints:

1 <= T <= 10^5
-10^6 <= x, y <= 10^6

### Ouput Format:

For each test case output "YES" if they are collinear and "NO" if they are not.

### Sample Input:

2
1 3
1 4
1 5
-1 0

0 1
1 0

**Sample Output:**

YES

NO

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

int32_t main()
{
	ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

	int t; cin >> t;
	while(t--)
	{
		int x1, y1; cin >> x1 >> y1;
		int x2, y2; cin >> x2 >> y2;
		int x3, y3; cin >> x3 >> y3;

		int determinant = x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2);

		if(determinant == 0) {
			cout << "YES\n";
		}
		else {
			cout << "NO\n";
		}
	}
	return 0;
}
```

## 5-Ass : Ninja Investigations

Send Feedback

A recent non violent protest in Uptown Plaza took a drastic turn as a bomb was blasted by some resurgent. Ninja Investigations was hired to determine the responsible party. Using advanced satellite systems and GPS tracking of cell phones, the investigators determined the coordinates of every person in the plaza. Using some advanced algorithms, the investigative team has also determined the approximated coordinates of the bomb and the vicinity in which a person had to be if he planted the bomb. You have been given a very important task - narrow down the suspect pool. To do this, you have to

find out whether a given person was in the vicinity of the bomb. Also print the probability that they planted the bomb. The probability can be calculated using the following formula:

100 - (distance of point to be checked with approx coordinates of bomb / total area of the vicinity).
**Note:** A person in the plaza may not be carrying a phone and their coordinates might be unknown. But we can disregard such people, because the bomb was cell phone triggered, hence it would be impossible for a person not carrying the phone to plant and trigger the bomb.
**Note:** Get the answer of distance and area in double, then typecast it in int. Print probability in int also.

### Input Format:
First line contains integer t, representing the number of test cases.
For each test case
Line 1: contains integer n, number of vertices of vicinity.
Next n lines contain coordinates of vertices of vicinity.
Next line contains approximate coordinates of the bomb.
Next line contains integer q, number of coordinates to be checked.
Next q lines contain coordinates of the people to be checked.

### Output Format:
For each test case, print “Yes” if the person is present in the vicinity of the bomb and the probability that he planted the bomb in the next line, otherwise print “No”.

### Sample Case 1:
1
3
1 10
9 1
10 7
5 6
2
8 6
8 8

### Sample Output 1:
Yes
90

No

```
#include<bits/stdc++.h>
using namespace std;
#define INF 50000
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)

struct Point
{
    double x,y;
    Point() {}
```

```
    Point(double a, double b)
    {
        x = a;
        y = b;
    }
    double cross(Point p)
    {
        return x*p.y - y*p.x;
    }
    double distance(Point p)
    {
        double dis = sqrt((pow(p.x - x, 2.0) + pow(p.y - y,2.0)) * 1.0);
        return sqrt(pow(p.x - x, 2) + pow(p.y - y, 2) * 1.0);
    }
};

double polygonArea(Point arr[], int n)
{
    double a = arr[n - 1].cross(arr[0]);

    for(int i=0;i<int(n) - 1;i++)
    {
        a += arr[i].cross(arr[i + 1]);
    }
    return a/2;
}

int probability(Point arr[], int n, Point p, Point app)
{
    int area = polygonArea(arr, n);
    int dis = app.distance(p);
    int prob = (dis*1.0 / area*1.0) * 100;
    return 100 - prob;
}

int isInside(Point polygon [], int N, Point p)
{
    int counter = 0;
    int i;
    double xinters;
    Point p1,p2;

    p1 = polygon[0];
    for(i = 1;i<=N;i++)
```

```
55.    {
56.        p2 = polygon[i % N];
57.        if(p.y > MIN(p1.y,p2.y))
58.        {
59.            if(p.y <= MAX(p1.y,p2.y))
60.            {
61.                if(p.x <= MAX(p1.x,p2.x))
62.                {
63.                    if(p1.y != p2.y)
64.                    {
65.                        xinters = (p.y-p1.y)*(p2.x-p1.x)/(p2.y-p1.y)+p1.x;
66.                        if(p1.x == p2.x || p.x <= xinters)
67.                            counter++;
68.                    }
69.                }
70.            }
71.        }
72.        p1 = p2;
73.    }
74.
75.    if(counter % 2 == 0)
76.    {
77.        return 1;
78.    }
79.    else
80.    {
81.        return 0;
82.    }
83. }
84. int main()
85. {
86.    int t;
87.    cin>>t;
88.    while(t--)
89.    {
90.        int n;
91.        cin>>n;
92.        Point arr[n];
93.        for(int i=0;i<n;i++)
94.        {
95.            double a,b;
96.            cin>> a>> b;
97.            arr[i] = Point(a, b);
98.        }
```

```
99.
100.        double a, b;
101.        cin>> a >> b;
102.        Point approx(a,b);
103.
104.        int q;
105.        cin >> q;
106.
107.        for(int i = 0;i < q;i++)
108.        {
109.           int a, b;
110.           cin>> a >>b;
111.           Point ch(a,b);
112.           if(!isInside(arr,n,ch))
113.           {
114.              cout<<"Yes"<<endl;
115.              cout<<probability(arr,n,ch,approx)<<endl;
116.           }
117.           else
118.           {
119.              cout<<"No"<<endl;
120.           }
121.        }
122.     }
123.     return 0;
124.  }
```