



COMPETITIVE PROGRAMMING LAB RECORD

Problem Statement 1:

1. Maximum subarray sum

CO1

Solve the Maximum subarray sum with different time complexities.

Description:

Maximum subarray: The continuous subset of the array which has the maximum sum when compared to all the subsets residing in the array is called as maximum sub array. This problem can be viewed in different perspectives:

- If the input contains all the positive integers → The maximum subarray sum is the sum returned by adding all the elements of the array.
- If the input contains all the negative integers → The maximum subarray sum is the largest integer of the array or in some cases 0.
- If the input is a combination of both +ve and -ve integers → The maximum subarray is obtained by iterating through all the subsets of the array and returning the best optimal value

In this problem, the code is designed in such a way to process input of +ve and mixed integers.

Example:

arr: {4,3,-7,2,1}

Out of all sub arrays we can make the subarray {4,3} has the maximum sum.

Thus, the maximum subarray sum = 7

Approach1:[Brute Force approach using nested loops]

- The idea is to start at all positions in the array and calculate running sums.
- The outer loop picks the beginning element, the inner loop finds the maximum possible sum with the first element picked by the outer loop and compares this maximum with the overall maximum.
- This solves the problem in $O(n^2)$ using two nested loops.

Code1:

```
#include <stdio.h>
int maxSubarraySum ( int A [] , int n)
{
    int max_sum = 0;
    for(int i=0 ;i<n;i++)
    { int sum = 0;
      for(int j=i;j<n;j++)
      {
          sum = sum + A[j];
          if(sum > max_sum)
              max_sum = sum;
      }
    }
```



COMPETITIVE PROGRAMMING LAB RECORD

```
}  
return max_sum;  
}  
int main()  
{  
int i,n;  
scanf("%d",&n);  
int a[n];  
for(i=0;i<n;++i)  
scanf("%d",&a[i]);  
printf("maximum subarray sum:%d",maxSubarraySum(a,n));  
return 0;  
}
```

Output:

```
5  
4 3 -7 2 1  
maximum subarray sum:7
```

Approach2:[Kadane's Approach]

- Define cs and ms to 0 initially
- Iterate the array and add the value of the current element to cs
- Then check if cs>ms,update ms=cs
- If cs<0, cs=0
- After reaching the end of the array print ms value (returns the maximum subarray sum)

Kadane's Approach solves the problem in **O(n)** using dynamic programming

Code:

```
#include<stdio.h>  
int main()  
{  
int i,n,t,cs=0,s=0;  
scanf("%d",&n);  
int a[n];  
for(i=0;i<n;++i)  
scanf("%d",&a[i]);  
for(i=0; i<n; i++) {  
    cs = cs + a[i];  
    if (cs < 0)  
        cs = 0;  
    if(s< cs)
```



COMPETITIVE PROGRAMMING LAB RECORD

```
s = cs;  
}  
printf("maximum subarray sum:%d",s);  
return 0;  
}
```

Output:

```
7  
10 7 3 5 8 2 9  
maximum subarray sum:44
```

Hackerrank Submissions:

[Maximum-sub-array-sum](#)

C

about 1 month ago

Accepted ✓

Problem Statement2:

2. The Median of Two Sorted Arrays

CO1

Program to find the median of two sorted arrays of same size and different size are discussed here. Firstly, let us see what is median of the array? Median is an element which divides the array into two parts - left and right. So the number of elements on the left side of the array will be equal or less than the number of elements on the right side. Now, let us consider the case of an array with odd number of elements. Array = [9,11,16,7,2] Sorted array = [2,7,9,11,16]. In this case, the median of this array is 9, since it divides the array into two parts: [2,7] and [11,16]. Further, let us consider the case of an array with even elements. Array = [1,2,3,4,5,6]. In such a case, we will take the average between the last element of the left part and the first element of the right part. In this case, the median equals = $(3 + 4) / 2 = 3.5$.

Input Format

The input should contain 3 lines.

I. First line of the input should contain two integer values which specify the number of elements in array1 and array2.

II. Second line of the input should contain the elements of the first array.

III. Third line of the input should contain the elements of the second array.

Constraints

All elements must be Integers

Output Format

The output should print only the median value.

Sample Input 1

5 6
-5 3 6 12 15
-12 -10 -6 -3 4 10

Sample Output 1

3

Sample Input 2

4 6
2 3 5 8
10 12 14 16 18 20

Sample Output 2

11

Description:

Median of two sorted arrays: The median is considered as the middle most element of the given data. Here the data is 2 different arrays .There exists two conditions to solve the problem:



COMPETITIVE PROGRAMMING LAB RECORD

- The arrays can be of same size and are sorted
- The arrays are of different size and are sorted

The below given codes deals with both variable and same sized arrays sorted together in ascending order.

Approach1:

The given two arrays are taken into 1 arr and sorted using insertion sort. Then based on the size of arr the median is found as follow:

- **Case 1:** length is odd: then the median is at $(\text{length})/2^{\text{th}}$ index in the array obtained after merging both the arrays.
- **Case 2:** If the length of the third array is even, then the median will be the average of elements at index $((\text{length})/2)$ and $((\text{length})/2 - 1)$ in the array obtained after merging both the arrays.

Code1:

```
#include<bits/stdc++.h>
using namespace std;
void insertionSort(int arr[], int n,int start)
{
    int i, key, j;
    for (i = start; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
int main()
{
    int n,m;
    cin>>n>>m;
    int* arr=new int[n+m];
    for(int i=0;i<n+m;i++){
        cin>>arr[i];
    }
    insertionSort(arr,n+m,n);
```

```
if((n+m)%2!=0)
cout<<arr[(n+m)/2]<<endl;
else
cout<<(arr[(n+m)/2]+arr[(n+m+1)/2-1])/2<<endl;
return 0;
}
```

Output:

Input (stdin)

```
5 6
-5 3 6 12 15
-12 -10 -6 -3 4 10
```

Your Output (stdout)

```
3
```

Expected Output

```
3
```

Approach2:

Two arrays are taken as input and concatenated into a 3rd array then sorted using sort() function, based on the size of the array median is found as in the above approach.

Code2:

```
n2=input().split(" ")
n=n2[0]
m=n2[1]
l=list(map(int,input().strip().split()))[:int(n)]
a=list(map(int,input().strip().split()))[:int(m)]
m1=a+l
m1.sort()
n1=len(m1)
if(n1%2==0):
x=n1//2
y=m1[x]
y1=m1[x-1]
med=(y+y1)/2
else:
x=n1//2
med=m1[x]
```



COMPETITIVE PROGRAMMING LAB RECORD

```
print(int(med))
```

Output:

Input (stdin)

```
5 6
-5 3 6 12 15
-12 -10 -6 -3 4 10
```

Your Output (stdout)

```
3
```

Expected Output

```
3
```

Approach3:

The 3rd is obtained by using bubble sort technique and the median is found based on length of the 3rd array,

Code 3:

```
#include <stdio.h>
int main()
{
    int n,m,o,p,t,k,med;
    scanf("%d",&n);
    scanf("%d",&m);
    int a[n],b[m];
    o=m+n;
    int c[o];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
        c[i]=a[i];
    }
    k=n;
    for(int i=0;i<m;i++)
    {
        scanf("%d",&b[i]);

        c[k]=b[i];
        k++;
    }
}
```

```
}  
for(int i=0;i<o;i++)  
{  
for(int j=0;j<o-i-1;j++)  
{  
if(c[j]>c[j+1])  
{  
t=c[j];  
c[j]=c[j+1];  
c[j+1]=t;  
}  
}  
}  
if((o%2)==0)  
{  
p=o/2;  
med=(c[p]+c[p-1])/2;  
printf("%d",med);  
}  
else  
{  
p=o/2;  
med=c[p];  
printf("%d",med);  
} }
```

Output:

```
4 6  
2 3 5 8  
10 12 14 16 18 20  
11|
```

Hackerrank Submissions:

[Median of two sorted arrays](#)

Python 3

about 1 month ago

Accepted ✓

Problem Statement3:

3. Division with Binary Search

CO1

We can modify binary search Approach to perform division of two numbers, by defining range $[0, \text{infinity}]$ which serves as initial low and high for the binary search Approach. Now we need to find a mid that satisfies $x/y = \text{mid}$ or $x * \text{mid}$ for given two numbers x and y . Based on the comparison result based on x and $y * \text{mid}$, we either update low, update high or return mid. 1. If $y * \text{mid}$ almost equal to x , we return mid. 2. If $y * \text{mid}$ is less than x , we update low to mid 3. If $y * \text{mid}$ is more than x , we update high to mid We need to care about division by zero and sign of the result etc. Input: one line of input should contain two numbers separated by space. Output: should print division of the numbers as a result.

Input Format

Input: one line of input should contain two numbers separated by space. Input1: 22 7

Constraints

$x, y < \text{infinity}$

Output Format

Output: should print division of the numbers as a result.

Sample Input 1

22 7

Sample Output 1

3.14286

Description:

Division using binary search: Binary search is a Divide and Conquer approach, As per the problem statement it finds the mid as per the given range and after comparison the mid gets updated to low or high value of the range. Thus in each step it reduces the search space to half.

We need to care about division by zero, sign of the result and whether the number is exactly divisible or leaves decimal values.

Approach:

- Start by defining range $[-32767, 32767]$ which serves as initial low and high for the binary search Approach.
- Now find the mid value
- Based on a comparison result between x and $y \times \text{mid}$, either update low or high, or return mid if:
 - $y \times \text{mid}$ is almost equal to x , return mid.
 - $y \times \text{mid}$ is less than x , update low to mid.
 - $y \times \text{mid}$ is more than x , update high to mid.

Code1:

```
#include<bits/stdc++.h>
using namespace std;
float binarydiv(float low,float high,int x,int y)
{
float mid=low+(high-low)/2.0;
if(mid*y==x)
return mid;
if(mid*y>x){
return binarydiv(low,mid,x,y);
}
else
return binarydiv(mid,high,x,y);
}
int main()
{
int x,y;
cin>>x>>y;
if(y==0){
cout<<"division by zero0"<<endl;
return 0;
}
float ans=binarydiv(-32767,32767,x,y);
cout<<ans<<endl;
return 0;
}
```

Output:

Input (stdin)

22 7

Your Output (stdout)

3.14286

Expected Output

3.14286

Code 2:

```
x,y=list(map(int,input().split()))
```



COMPETITIVE PROGRAMMING LAB RECORD

```
sign=1
count=0
if x*y<0:
    sign=-1
m=abs(x)
n=abs(y)
if n==0:
    print("division by zero")
else:
    low=0
    high=m
    mid=high
    while(mid*n!=m):
        if(count<=high):
            mid=(high+low)//2
        else:
            mid=(high+low)/2
        if(mid*n>m):
            high=mid
        else:
            low=mid
        count=count+1
    print(round(mid,5)*sign)
```

Output:

```
1200 30
40
```

Hackerrank Submissions:

Division with Binary Search	C++20	about 1 month ago	Accepted ✓	10
Division with Binary Search	Python 3	4 days ago	Accepted ✓	10



COMPETITIVE PROGRAMMING LAB RECORD

Problem Statement4:

4.Find the Triplet

CO1

Given an array of integers, find a triplet having maximum product in the array.

Input: First line of input should specify the number of elements in the array.

Second line of input should specify each element separated by space.

Output: should print Triplets.

Test Cases:

Sample Input 1

5

-4 1 -8 9 6

Sample Output 1

-4 -8 9

Sample Input 2

5

1 7 2 -2 5

Sample Output 2

7 2 5

Description:

Triplet having maximum product: The problem statement is about finding 3 numbers of the array which can result in maximum product of the entire array of 3 numbers.

This can be solved in different approaches the best approach for solving the problem is given below

Approach:

- Sort the array in ascending order
- Compare $(a[0]*a[1]*a[n-1]) > (a[n-1]*a[n-2]*a[n-3])$
- Return the highest value

This deals the problem solution logically as after sorting the array, there are 2 possible ways to get the maximum product

→ Either 2 -ve numbers and the max number of the array pair up to give maximum product

→ Else, last 3 consecutive numbers result in maximum product.

Code:

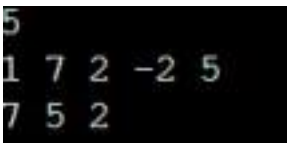
```
#include <iostream>
#include<bits/stdc++.h>
using namespace std;
void max_t(int a[],int n)
{
if(n<3)
```

```
cout<<-1;
sort(a,a+n);
if((a[0]*a[1]*a[n-1])>(a[n-1]*a[n-2]*a[n-3]))
cout<<a[0]<<" "<<a[1]<<" "<<a[n-1];
else
cout<<a[n-1]<<" "<<a[n-2]<<" "<<a[n-3];
}
int main()
{
int n;
cin>>n;
int a[n];
for(int i=0;i<n;i++)
cin>>a[i];
max_t(a,n);
return 0;}
```

Output:



```
5
-4 1 -8 9 6
-8 -4 9
```



```
5
1 7 2 -2 5
7 5 2
```

Problem Statement5:

5.3n+1

CO1

Consider the following Approach to generate a sequence of numbers. Start with an integer n . If n is even, divide by 2. If n is odd, multiply by 3 and add 1. Repeat this process with the new value of n , terminating when $n = 1$. For example, the following sequence of numbers will be generated for $n = 22$: 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1. It is conjectured (but not yet proven) that this Approach will terminate at $n = 1$ for every integer n . Still, the conjecture holds for all integers up to at least 1, 000, 000. For an input n , the cycle-length of n is the number of numbers generated up to and including the 1. In the example above, the cycle length of 22 is 16. Given any two numbers i and j , you are to determine the maximum cycle length over all numbers between i and j , including both endpoints. The input will consist of a series of pairs of integers i and j , one pair of integers per line. All integers will be less than 1,000,000 and greater than 0. Output For each pair of input integers i and j , output i, j in the same order in which they appeared in the input and then the maximum cycle length for integers between and including i and j . These three numbers should be separated by one space, with all three numbers on one line and with one line of output for each line of input.

Description:

In this $3n+1$ problem, start with an integer n . If n is even, divide by 2. If n is odd, multiply by 3 and add 1. Repeat this process with the new value of n , terminating when $n = 1$.

Here, we are given any two numbers i and j , and we have to determine the maximum cycle length over all numbers between i and j , including both endpoints.

Approach:

- Entering the starting(x) and ending(y) elements of the sequence.
- Initializing $t=0, a3=0, c=1$
- Assigning maximum of x, y to $a2$, minimum of x, y to $a1$
- Now run the loop for $i=a2$ until $i \geq a1$ by decrementing i
- For every iteration assigning i value to t and assigning 1 to c
While($t \neq 1$), if t is even $t=t/2$ and $c++$, else $t=t*3+1$ and $c++$
- Assigning the maximum of $a3$ and c to $a3$ for every iteration Finally print $x, y, a3$.

Code1:

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

```
int x,y;
while(cin>>x>>y){
int t=0;
int a3=0,c=1;
int a2=max(x,y);
int a1=min(x,y);
for(int i=a2;i>=a1;i--) {
t=i;
c=1;
while(t!=1) {
if(t%2==0)
t=t/2;
else
{
t=t*3;
t++;
}
c++;
}
a3=max(a3,c);
}
cout<<x<<" "<<y<<" "<<a3<<endl;
}
return 0;
}
```

Output:

```
100 200
100 200 125
```

```
201 210
201 210 89
```

Code2:

```
#include<bits/stdc++.h>
using namespace std;
int cycleLength(int n){
int c=0;
while(1){
```



COMPETITIVE PROGRAMMING LAB RECORD

```
c++;  
if(n==1)  
break;  
(n%2!=0)?n=(3*n)+1:n/=2;  
}  
return c;  
}  
int main(){  
int x,y,c,mn,mx,i=0;  
while(cin>>x>>y){  
mn=min(x,y),mx=max(x,y);  
c=0;  
for(int k=mn;k<=mx;k++)  
c=max(c,cycleLength(k));  
cout<<mn<<' '<<mx<<' '<<c<<endl; }  
return 0;  
}
```

Output:

```
1 10  
1 10 20
```

```
100 200  
100 200 125
```

```
201 210  
201 210 89
```

```
900 1000  
900 1000 174
```

UVA online judge submission:

27810624	100 The 3n + 1 problem	Accepted	PYTH3	1.240	2022-09-14 05:35:46
----------	------------------------	----------	-------	-------	------------------------

Problem Statement6:

6. Matching –String with wild card –pattern.

CO2

Check the given string is matches with pattern containing wild card characters („*” and „?”), where the „*” can match to any number of characters including zero characters and „?” can match to any single character in the given input string. Check if the given input string is matches with given input pattern or not.

Input:

Input should contain two lines. First line of input should contain input string. Second line of input should contain pattern string.

Output: The output should print either „0” or „1”. ‘1’ in the output indicates that the given string is matches with the given pattern. ‘0’ in the output indicates that the given string is not matched with the given pattern.

Sample Input 1:

abcabcccd
a?c*d

Sample Output1

1

Sample Input 2:

abcabcccd
a?c*c

Sample Output2

0

Description:

Given a text and a wildcard pattern, implement wildcard pattern matching algorithm that finds if wildcard pattern is matched with text. The matching should cover the entire text (not partial text). The wildcard pattern can include the characters ‘?’ and ‘*’

Case 1: The character is ‘*’ .

Here two cases will arise as follows:

>>We can ignore ‘*’ character and move to next character in the Pattern.

>>‘*’ character matches with one or more characters in Text. Here we will move to next character in the string.

Case 2: The character is ‘?’

We can ignore current character in Text and move to next character in the Pattern and Text.

Case 3: The character is not a wildcard character

If current character in Text matches with current character in Pattern, we move to next character in the Pattern and Text. If they do not match, wildcard pattern and Text do not match.

Approach:

- Enter the String.
- Enter the Pattern that you want to check with the string.
- call wildcard function which returns 1 if the pattern matches string or else returns 0 if not matches.
- To wildcard function we pass string and pattern.
- In the function we take 2 variables for the length of string and pattern and a matrix to verify
- each element from pattern matches the string or not .
- First make the total matrix to false to represent not match.
- Now check the pattern with string for the given conditions if matches make it true else remain same.
- Lastly return the end value of the matrix.
- If the returned value is 1 print matched else print not matched.

CODE 1:

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

using namespace std;

// definition of wildcard () function

bool wildcard (string str, string pattern)

{

    int i, j;    // variables declaration

    int m=str.length(); // finding the length of the string

    int n = pattern.length(); // finding the length of the pattern

    bool mat[m+1][n+1];
```

//Initially initializing the whole matrix with the false value.

```
for(i=0;i<=m;i++)  
{  
    for(j=0;j<=n;j++)  
        mat[i][j]=false;  
}
```

//because empty string and empty pattern always match

```
mat[0][0]=true;
```

//if the string is null

```
for(i=1;i<=n;i++)  
{  
    if(pattern[i-1]=='*')  
        mat[0][i]=mat[0][i-1];  
}
```

```
for(i=1;i<=m;i++)
```

```
{  
    for(j=1;j<=n;j++)  
    {  
        //if the character of pattern is *  
        if (pattern[j - 1] == '*')  
            mat[i][j] = mat[i][j - 1] || mat[i - 1][j];
```

//if the character is ? then consider them to match

```
else if (pattern[j - 1] == '?')  
    mat[i][j] = mat[i - 1][j - 1];
```

```
else if(str[i - 1] == pattern[j - 1])
```

```
mat[i][j] = mat[i - 1][j - 1];

//if it does not match any condition
else mat[i][j] = false;
}
}
return mat[m][n];
}
int main()
{
    string str, pat; // declaration of strings
    cin>>str;
    cin>>pat;
    bool result = wildcard(str, pat);
    if(result==1)
        cout<<"1";
    else
        cout<<"0";
}
```

OUTPUT:

Test case1:

```
abcabcccd
a?c*c
0
```

Test case 2:

```
abcabcccd
a?c*d
1
```



COMPETITIVE PROGRAMMING LAB RECORD

PROBLEM STATEMENT 7:

7. Rotten Oranges

CO2

Given a grid of dimension $n \times m$ where each cell in the grid can have values 0, 1 or 2 which has the following meaning:

0 : Empty cell

1 : Cells have fresh oranges

2 : Cells have rotten oranges

We have to determine what is the minimum time required to rot all oranges. A rotten orange at index $[i,j]$ can rot other fresh orange at indexes $[i-1,j]$, $[i+1,j]$, $[i,j-1]$, $[i,j+1]$ (**up, down, left and right**) in unit time.

Input

$\{\{0,1,2\},\{0,1,2\},\{2,1,1\}\}$

Output:

1

Explanation:

0 1 2

0 1 2

2 1 1

Oranges at position (0,2),(1,2),(2,0) will rot oranges at (0,1),(1,1),(2,2) and (2,1) in unit time.

Description:

In this problem they will give us a grid with some dimensions with values 0,1,2 which means 0 for no orange 1 for fresh orange and 2 for rotten orange. In this, for every unit time if there is a fresh orange beside a rotten orange the fresh orange will also get rotten if there is not orange nothing will happen. If there is a possibility to left with a fresh orange then the minimum time required is -1 i.e., not all oranges become rotten.

Approach :

- Create a variable **no = 2** and **changed = false**.
- Run a loop until there is no cell of the matrix which is changed in an iteration.
- Run a nested loop and traverse the matrix:

- If the element of the matrix is equal to **no** then assign the adjacent elements to **no + 1** if the adjacent element's value is equal to 1, i.e. not rotten, and update **changed** to true.
- Traverse the matrix and check if there is any cell that is **1**.
- If 1 is present return -1
- Else return **no - 2**.

CODE1:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
bool issafe(int i, int j,int R,int C)
{
    if (i >= 0 && i < R && j >= 0 && j < C)
        return true;
    return false;
}
int main()
{
    int R,C;
    cin>>R>>C;
    int v[R][C];
    for(int i=0;i<R;i++)
    {
        for(int j=0;j<C;j++)
        {
            cin>>v[i][j];
        }
    }
```

```
int m,n;

bool changed = false;

int no = 2;

while (true)
{
    for (int i = 0; i < R; i++)
    {
        for (int j = 0; j < C; j++)
        {
            if (v[i][j] == no)
            {
                if (issafe(i + 1, j,R,C)&& v[i + 1][j] == 1)
                {
                    v[i + 1][j] = v[i][j] + 1;
                    changed = true;
                }
                if (issafe(i, j + 1,R,C)&& v[i][j + 1] == 1)
                {
                    v[i][j + 1] = v[i][j] + 1;
                    changed = true;
                }
                if (issafe(i - 1, j,R,C)&& v[i - 1][j] == 1)
                {
                    v[i - 1][j] = v[i][j] + 1;
                    changed = true;
                }
                if (issafe(i, j - 1,R,C)&& v[i][j - 1] == 1)
```

```
{  
    v[i][j - 1] = v[i][j] + 1;  
    changed = true;  
}  
}  
}  
}  
if (!changed)  
    break;  
changed = false;  
no++;  
}  
for (int i = 0; i < R; i++)  
{  
    for (int j = 0; j < C; j++)  
    {  
        if (v[i][j] == 1)  
            n=-1;  
    }  
}  
m=no-2;  
if(n!=-1)  
cout << "Max time incurred: " << m;  
else  
cout<<"Max time incurred:"<<n;  
return 0;  
}
```




COMPETITIVE PROGRAMMING LAB RECORD

OUTPUT:

Test case1.

```
3 3
0 1 2 0 1 2 2 1 1
Max time incurred: 1
```

Test case 2.

```
3 4
2 1 0 0
1 0 0 2
2 0 1 1
Max time incurred: 2
```

PROBLEM STATEMENT8 :

8. Minimum-Cost –Path

CO2

Find a path in an $n \times n$ grid from the upper-left corner to the lower-right corner such that we only move down and right and diagonally lower cells from a given cell, i.e., from a given cell (i, j) , cells $(i+1, j)$, $(i, j+1)$ and $(i+1, j+1)$ can be traversed. Assume that all costs are positive integers. Each square contains a number, and the path should be constructed so that the sum of numbers along the path is as small as possible.

1	2	3
4	8	2
1	5	3

1	2	3
4	8	2
1	5	3

The path is $(0, 0) \rightarrow (0, 1) \rightarrow (1, 2) \rightarrow (2, 2)$. The cost of the path is 8 $(1 + 2 + 2 + 3)$

Sample Input 1

1 2 3
4 8 2
1 5 3

Sample Output 1

8

Description:

A matrix of the different cost is given. Also, the destination cell is provided. We have to find minimum cost path to reach the destination cell from the starting cell (0, 0).

Each cell of the matrix represents the cost to traverse through that cell.

From a cell, we cannot move anywhere, we can move either to the right or to the bottom or to the lower right diagonal cell, to reach the destination.

Approach :

- First we will choose the last element present in the array and will reach the first element within minimum cost path.
- In order to move from one cell to other we will check the minimum cost cell in between the immediate up, left and diagonal cells.
- In this way we will find the path to the source and add all the cost.
- Then we will return the cost of the path.

CODE1 :

```
#include <limits.h>
#include <stdio.h>
int min(int x, int y, int z){
    if (x < y)
        return (x < z) ? x : z;
    else
        return (y < z) ? y : z;}
int main()
{
    int m,n,i,j;
    printf("Enter the number of rows and columns:");
    scanf("%d %d",&m,&n);
```

```
int cost[m][n];

int tc[m][n];

printf("Enter the cost:");

for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
        scanf("%d",&cost[i][j]);
}

tc[0][0] = cost[0][0];

for (i = 1; i <= m; i++)
    tc[i][0] = tc[i - 1][0] + cost[i][0];

for (j = 1; j <= n; j++)
    tc[0][j] = tc[0][j - 1] + cost[0][j];

for (i = 1; i <= m; i++)
    for (j = 1; j <= n; j++)
        tc[i][j] = min(tc[i - 1][j - 1], tc[i - 1][j],
                        tc[i][j - 1])
                    + cost[i][j];

printf("%d",tc[m-1][n-1]);

return 0;

}
```

OUTPUT:

Test case 1:

```
Enter the number of rows and columns:3 3
Enter the cost:1 2 5 4 8 2 1 5 3
8
```

COMPETITIVE PROGRAMMING LAB RECORD

Test case 2:

```
Enter the number of rows and columns:4 3
Enter the cost:1 6 7 8 2 3 1 9 0 3 3 1
4
```

PROBLEM STATEMENT9 :

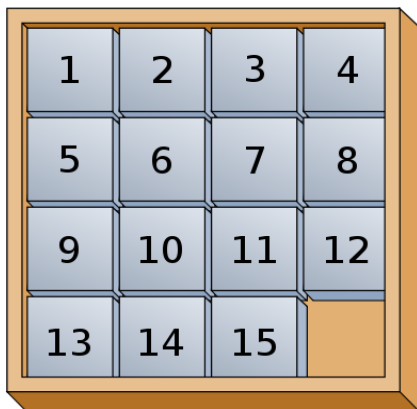
9. 15-Puzzle Problem

CO3

The 15-puzzle is a very popular game: you have certainly seen it even if you don't know it by that name. It is constructed with 15 sliding tiles, each with a different number from 1 to 15, with all tiles packed into a 4 by 4 frame with one tile missing. The object of the puzzle is to arrange the e tiles so that they are ordered as below:

The only legal operation is to exchange the missing tile with one of the 2, 3, or 4 tiles it shares an edge with. Consider the following sequence of moves:

We denote moves by the neighbour of the missing tile is swapped with it. Legal values are "R", "L," "U," and "D" for right, left, up, and down, based on the movements of the hole. Given an initial configuration of a 15-puzzle you must determine a sequence of steps that take you to the final state. Each solvable 15-puzzle input requires at most 45 steps to be solved with our judge solution; you are limited to using at most 50 steps to solve the puzzle.



Input: The first line of the input contains an integer n indicating the number of puzzle set inputs. The next 4n lines contain n puzzles at four lines per puzzle. Zero denotes the missing tile.

Output: For each input set you must produce one line of output. If the given initial configuration is not solvable, print the line "This puzzle is not solvable." If the puzzle is solvable, then print the move sequence as described above to solve the puzzle.

Sample Input 1

2



COMPETITIVE PROGRAMMING LAB RECORD

2 3 4 0

1 5 7 8

9 6 10 12

13 14 11 15

13 1 2 4

5 0 3 7

9 6 10 12

15 8 11 14

Sample Output 1

LLDRDRDR

This puzzle is not solvable

DESCRIPTION :

The 15 puzzle is a sliding puzzle having 15 square tiles numbered 1–15 in a frame that is 4 tiles high and 4 tiles wide, leaving one unoccupied tile position. Tiles in the same row or column of the open position can be moved by sliding them horizontally or vertically, respectively. The goal of the puzzle is to place the tiles in numerical order.

ALGORITHM :

- 1.Start
- 2.Declare global variables:
 - n:size of puzzle
 - L:steps made to reach target puzzle
 - steps: number of steps taken
 - a: actual puzzle problem
 - temp: temporary puzzle
 - t: target puzzle
- 3.Take puzzle(a) as input.
- 4.Take target puzzle(t) as input.
- 5.Find the position of row and column index where space is present as x,y respectively.
- 6.Do the temporary assignment of the current puzzle to 'temp'.
- 7.If x is not 0,move space up and check the number of mismatches.
8. If the number of mismatches are less than previous puzzle mismatches,then assign temp to a and record the step as 'U' denoting up.
9. Repeat step 6 to 9 for down movement if x is not n-1.



COMPETITIVE PROGRAMMING LAB RECORD

- 10.Repeat step 6 to 9 for left movement if y is not 0.
- 11.Repeat step 6 to 9 for down movement if y is not n-1.
- 12.Increment the step count.
- 13.Repeat steps 5 to 12 till we reach target puzzle and step count is less than 50.
- 14.If step count is greater than 50 ,print “Solution not found” and end the program.
- 15.Else,print the steps recorded.

CODE:

```
#include <iostream>
using namespace std;
int m=0,n=4;
int cal(int temp[10][10],int t[4][4])
{
    int i,j,m=0;
    for(i=0;i < n;i++)
    for(j=0;j < n;j++)
    {
        if(temp[i][j]!=t[i][j])
            m++;
    }
    return m;
}
int check(int a[10][10],int t[4][4])
{
    int i,j,f=1;
    for(i=0;i < n;i++)
    for(j=0;j < n;j++)
    if(a[i][j]!=t[i][j])
        f=0;
    return f;
}
int main()
{
    int p,i,j,man,n=4,a[10][10],temp[10][10],r[10][10];
    int m=0,x=0,y=0,d=1000,dmin=0,l=0;
    char name[50];
    int t[4][4]={ 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0};
    cin>>man;
    while(man!=0){
        for(i=0;i < n;i++)
        for(j=0;j < n;j++)
        cin>>a[i][j];
        while(!(check(a,t)) && l<=50)
        {
            l++;
            d=1000;
            for(i=0;i < n;i++)
```



```
for(j=0;j < n;j++)
{
if(a[i][j]==0)
{
x=i;
y=j;
}
}
//To move upwards
for(i=0;i < n;i++)
for(j=0;j < n;j++)
temp[i][j]=a[i][j];
if(x!=0)
{
p=temp[x][y];
temp[x][y]=temp[x-1][y];
temp[x-1][y]=p;
}
m=cal(temp,t);
dmin=l+m;
if(dmin < d)
{
d=dmin;
for(i=0;i < n;i++)
for(j=0;j < n;j++)
r[i][j]=temp[i][j];
name[l-1]='U';
}
//To move downwards
for(i=0;i < n;i++)
for(j=0;j < n;j++)
temp[i][j]=a[i][j];
if(x!=n-1)
{
p=temp[x][y];
temp[x][y]=temp[x+1][y];
temp[x+1][y]=p;
}
m=cal(temp,t);
dmin=l+m;
if(dmin < d)
{
d=dmin;
for(i=0;i < n;i++)
for(j=0;j < n;j++)
r[i][j]=temp[i][j];
name[l-1]='D';
}
```

```
}
//To move right side
for(i=0;i < n;i++)
for(j=0;j < n;j++)
temp[i][j]=a[i][j];
if(y!=n-1)
{
p=temp[x][y];
temp[x][y]=temp[x][y+1];
temp[x][y+1]=p;
}
m=cal(temp,t);
dmin=l+m;
if(dmin < d)
{
d=dmin;
for(i=0;i < n;i++)
for(j=0;j < n;j++)
r[i][j]=temp[i][j];
name[l-1]='R';
}
//To move left
for(i=0;i < n;i++)
for(j=0;j < n;j++)
temp[i][j]=a[i][j];
if(y!=0)
{
p=temp[x][y];
temp[x][y]=temp[x][y-1];
temp[x][y-1]=p;
}
m=cal(temp,t);
dmin=l+m;
if(dmin < d)
{
d=dmin;
for(i=0;i < n;i++)
for(j=0;j < n;j++)
r[i][j]=temp[i][j];
name[l-1]='L';
}
for(i=0;i < n;i++)
for(j=0;j < n;j++)
{
a[i][j]=r[i][j];
temp[i][j]=0;
}
}
```

```
if(check(a,t)==1)
{
    for(int k=0;k<l;k++)
    {
        cout<<name[k];
    }
    cout<<endl;
}
else
    cout<<"This puzzle is not solvable"<<endl;
man--;}
return 0;
}
```

OUTPUT:

1.

```
2
2 3 4 0 1 5 7 8 9 6 10 12 13 14 11 15
LLDDRDRDR
13 1 2 4 5 0 3 7 9 6 10 12 15 8 11 14
This puzzle is not solvable
```

2.

```
1
2 3 4 0 1 5 7 8 9 6 10 12 13 14 11 15
LLDDRDRDR
```



COMPETITIVE PROGRAMMING LAB RECORD

PROBLEM STATEMENT10 :

10. Tug of War

C03

Tug of war is a contest of brute strength, where two teams of people pull in opposite directions on a rope. The team that succeeds in pulling the rope in their direction is declared the winner. A tug of war is being arranged for the office picnic. The picnickers must be fairly divided into two teams. Every person must be on one team or the other, the number of people on the two teams must not differ by more than one, and the total weight of the people on each team should be as nearly equal as possible.

Input:

The input begins with a single positive integer on a line by itself indicating the number of test cases following, each described below and followed by a blank line.

The first line of each case contains n , the number of people at the picnic. Each of the next n lines gives the weight of a person at the picnic, where each weight is an integer between 1 and 450. There are at most 100 people at the picnic. Finally, there is a blank line between each two consecutive inputs.

Output:

For each test case, your output will consist of a single line containing two numbers: the total weight of the people on one team, and the total weight of the people on the other team. If these numbers differ, give the smaller number first. The output of each two consecutive cases will be separated by a blank line.

Sample Input 1

```
1
3
100
90
200
```

Sample Output 1

```
190 200
```

DESCRIPTION :

Given a set of n integers, divide the set in two subsets of $n/2$ sizes each such that the absolute difference of the sum of two subsets is as minimum as possible. If n is even, then sizes of two subsets must be strictly $n/2$ and if n is odd, then size of one subset must be $(n-1)/2$ and size of other subset must be $(n+1)/2$.

Algorithm :

- First read the no of testcases and n .
- Now declare a arraylist arr and then add each element to the arraylist .
- Now sort arr and seperate positive arraylist arrp and negative arraylist arrn .
- Sort both arrp and arrn and store last two greatest values in left and right correspondingly.
- Remove those elements from arrp .
- Now by iterating the loop for arrn remove each element from max(left,right) .
- Then add elements from arrp to min(left,right) .
- Print left and right .

CODE1 :

```
import java.io.*;
import java.util.*;
public class Main
{
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int t;
        t=sc.nextInt();
        while(t!=0){
            int n=sc.nextInt();
            ArrayList<Integer>arr=new ArrayList<>(n);
            for(int i=0;i<n;i++)
                arr.add(sc.nextInt());
```

COMPETITIVE PROGRAMMING LAB RECORD

```
Collections.sort(arr);

ArrayList<Integer>arr1=new ArrayList<>(n);
ArrayList<Integer>arr2=new ArrayList<>(n);
int left,right;
left=arr.get(n-1);
right=arr.get(n-2);
for(int i=n-3;i>=0;i--)
{
    if(arr.get(i)>0){
        if(left<=right)
            left=left+arr.get(i);
        else
            right=right+arr.get(i);
    }
    else{
        if(left<=right)
            right=right+arr.get(i);
        else
            left=left+arr.get(i);
    }
}
if(left<right)
    System.out.println(left+" "+right);
else
    System.out.println(right+" "+left);
t--;
}

}
```



COMPETITIVE PROGRAMMING LAB RECORD

}

OUTPUT :

Test case1.

```
1
3
100 90 200
190 200
```

Test case 2.

```
2
5
245 312 156 29 50
391 401
7
273 168 58 26 48 70 389
507 525
```

PROBLEM STATEMENT 11:

11. Find first set bit

CO3

Given an integer an N. The task is to return the position of first set bit found from the right side in the binary representation of the number.

Note: If there is no set bit in the integer N, then return 0 from the function.

Test case 1

Input: N = 18

Output: 2

Test case 2

Input: N = 12

Output: 3

Expected Time Complexity: $O(\log N)$.

Constraints:

$0 \leq N \leq 10^8$

DESCRIPTION :

The problem is to return the position of the first 1 from right to left, in the binary representation of an Integer.

Approach :

- Read the input and initialise count to 1
- Repeat the while loop until $n > 0$
- Perform and operation between 1 and n if it is zero then right shift n by '1' bit
- Increment count
- Else print count

CODE 1:

```
n=int(input())
```

```
count=1
```

```
while(n!=0):
```



```
if(n&1==0):
```

```
    n=n>>1
```

```
    count+=1
```

```
else:
```

```
    print(count)
```

```
    break
```

OUTPUT:

Test case 1:

```
18
2
```

Test case 2.

```
12
3
```

Description:

- Read the number
- Then do $\log_2(\text{number (logical AND) } - \text{number})$ is the index of the right most set bit.
- To know the position, add 1 to the index

CODE 2 :

```
import math
```

```
n=int(input())
```

```
print(math.log2(n&((~n)+1))+1)
```

OUTPUT:

Test case1:

```
18
2.0
```

Test case 2

```
12
3.0
```

