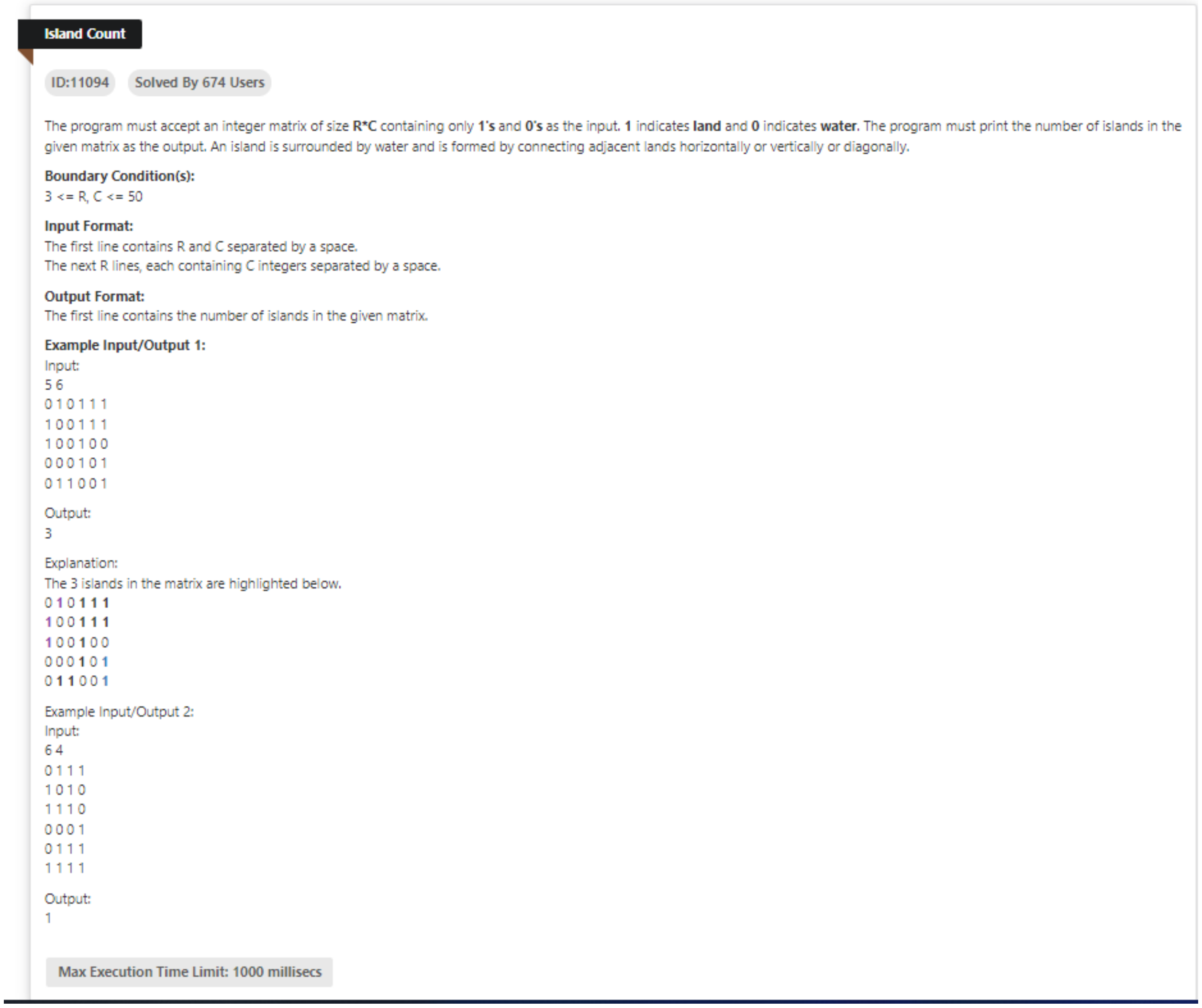
SESSION 1:

Problem 1:



Island Count

ID:11094 Solved By 674 Users

The program must accept an integer matrix of size **R*C** containing only **1's** and **0's** as the input. **1** indicates **land** and **0** indicates **water**. The program must print the number of islands in the given matrix as the output. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically or diagonally.

Boundary Condition(s):
3 <= R, C <= 50

Input Format:
The first line contains R and C separated by a space.
The next R lines, each containing C integers separated by a space.

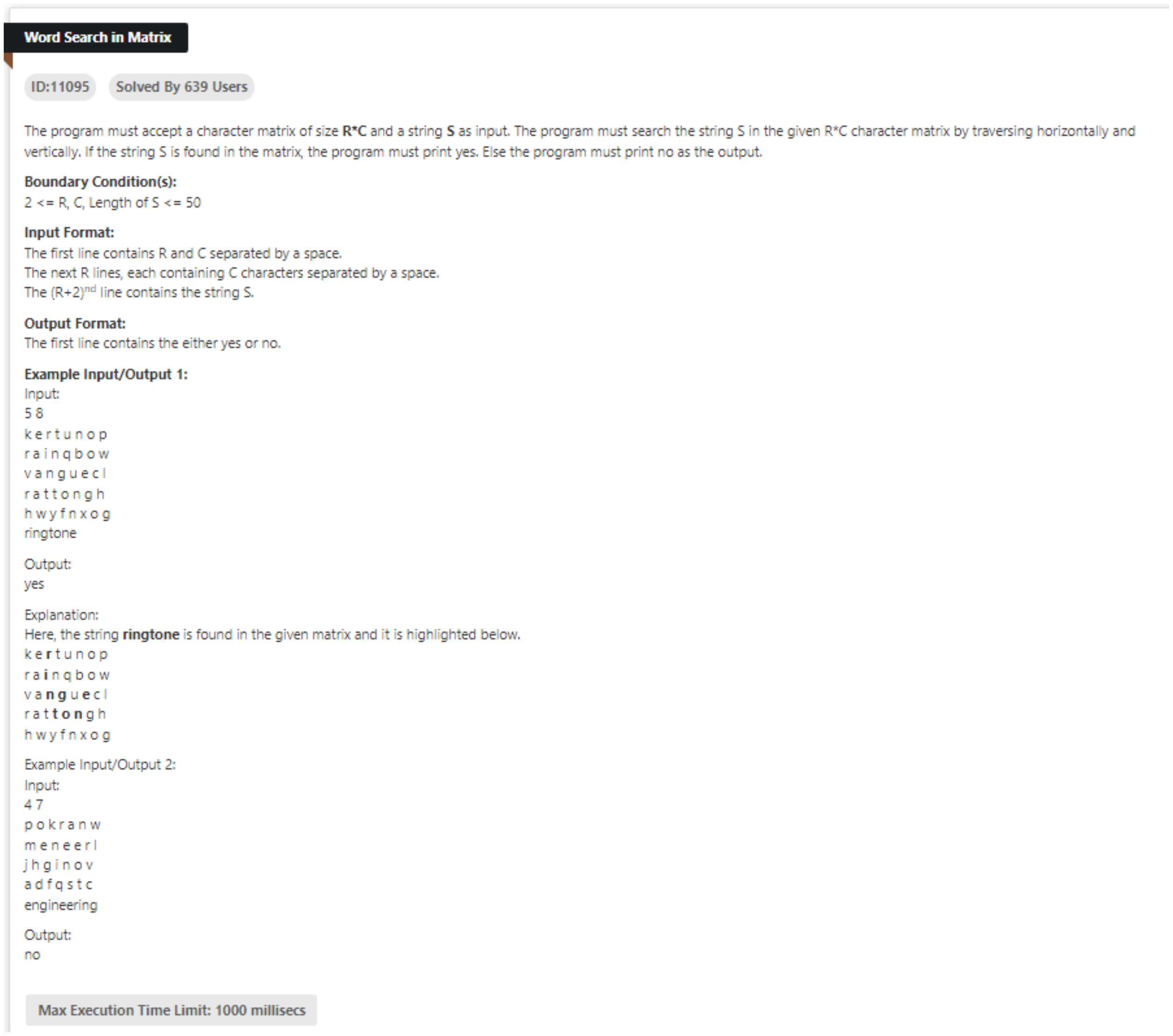
Output Format:
The first line contains the number of islands in the given matrix.

Example Input/Output 1:
Input:
```
5 6
0 1 0 1 1 1
1 0 0 1 1 1
1 0 0 1 0 0
0 0 0 1 0 1
0 1 1 0 0 1
```
Output:
```
3
```
Explanation:
The 3 islands in the matrix are highlighted below.
```
0 1 0 1 1 1
1 0 0 1 1 1
1 0 0 1 0 0
0 0 0 1 0 1
0 1 1 0 0 1
```
Example Input/Output 2:
Input:
```
6 4
0 1 1 1
1 0 1 0
1 1 1 0
0 0 0 1
0 1 1 1
1 1 1 1
```
Output:
```
1
```
Max Execution Time Limit: 1000 millisecs

Code:

```
import java.util.*;

class islandCount {
    static int R,C;
    private static void dfs(int[][] matrix,int row,int col){
        if(row>=0 && row<R && col>=0 && col<C ){
            if(matrix[row][col]==0){
                return;
            }
            matrix[row][col]=0;
            dfs(matrix,row-1,col);    //top
            dfs(matrix,row+1,col);    //bottom
            dfs(matrix,row,col-1);    //left
            dfs(matrix,row,col+1);    //right
            dfs(matrix,row-1,col-1);  //topLeft
            dfs(matrix,row+1,col+1);  //bottomRight
            dfs(matrix,row-1,col+1);  //topRight
            dfs(matrix,row+1,col-1);  //bottomLeft
        }
    }
    public static void main(String[] args) {
        //Your Code Here
        Scanner in= new Scanner(System.in);
         R=in.nextInt();
         C=in.nextInt();
        int[][] matrix = new int[R][C];
        for(int row=0;row<R;row++){
            for(int col=0;col<C;col++){
                matrix[row][col]=in.nextInt();
            }
        }
        int islandCount=0;
        for(int row=0;row<R;row++){
            for(int col=0;col<C;col++){
                if(matrix[row][col]==1){
                    dfs(matrix,row,col);
                    islandCount++;
                }
            }
        }
        System.out.println(islandCount);
    }
}
```

Problem 2:



Word Search in Matrix

ID:11095 Solved By 639 Users

The program must accept a character matrix of size **R*C** and a string **S** as input. The program must search the string S in the given R*C character matrix by traversing horizontally and vertically. If the string S is found in the matrix, the program must print yes. Else the program must print no as the output.

Boundary Condition(s):
2 <= R, C, Length of S <= 50

Input Format:
The first line contains R and C separated by a space.
The next R lines, each containing C characters separated by a space.
The (R+2)nd line contains the string S.

Output Format:
The first line contains the either yes or no.

Example Input/Output 1:
Input:
```
5 8
k e r t u n o p
r a i n q b o w
v a n g u e c l
r a t t o n g h
h w y f n x o g
ringtone
```

Output:
```
yes
```

Explanation:
Here, the string **ringtone** is found in the given matrix and it is highlighted below.
```
k e r t u n o p
r a i n q b o w
v a n g u e c l
r a t t o n g h
h w y f n x o g
```

Example Input/Output 2:
Input:
```
4 7
p o k r a n w
m e n e e r l
j h g i n o v
a d f q s t c
engineering
```

Output:
```
no
```

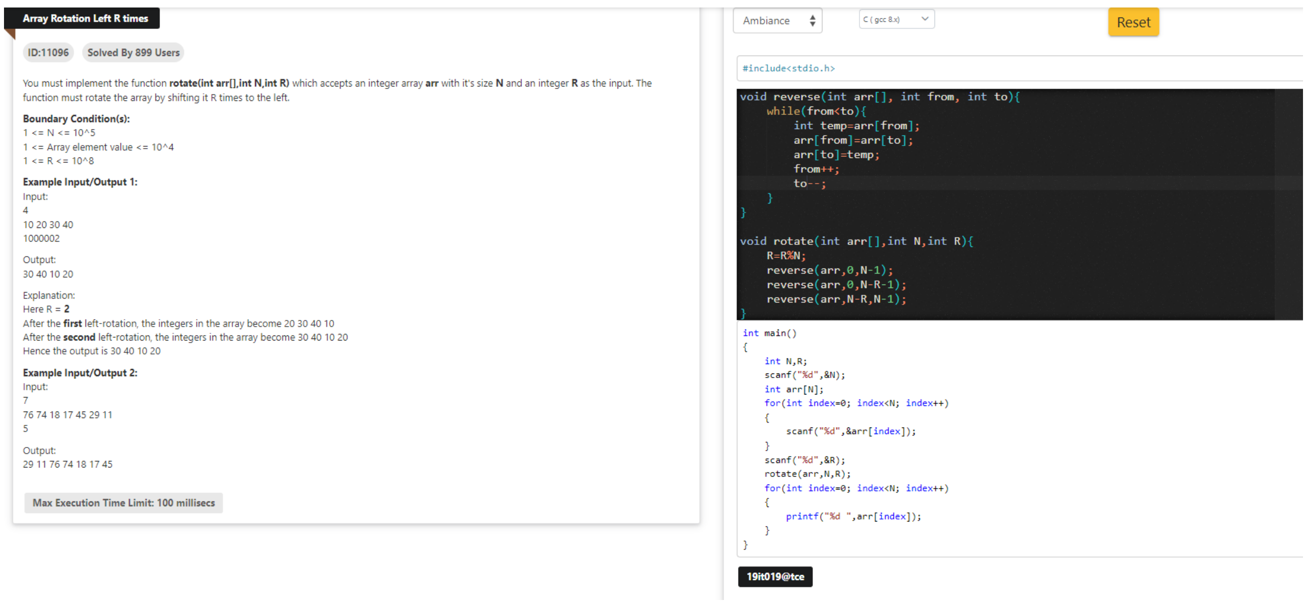
Max Execution Time Limit: 1000 millisecs

Code:

```
#include<stdio.h>
#include<stdlib.h>
int R,C,found=0;
void search(char matrix[R][C],int row,int col,char word[50],int index){
    if(row>=0 && row<R && col>=0 && col<C){
     if(word[index]==NULL){
        found=1;return;
    }
    if(matrix[row][col]!=word[index]){
        return;
    }
    char backup=matrix[row][col]; //getting the backup of the element -- same
element should not be considered
                                   // so setting to hypen,to avoid repeating of
same element
    matrix[row][col]='-'; //setting the "-" hypen as hypen is not a character
    search(matrix,row,col-1,word,index+1); //left
    search(matrix,row,col+1,word,index+1); //right
    search(matrix,row-1,col,word,index+1); //up
    search(matrix,row+1,col,word,index+1); //bottom
    matrix[row][col]=backup; // setting back the element with the backup element
    }
}
int main()
{
    scanf("%d%d",&R,&C);
    char matrix[R][C],str[2],word[50];

    for(int row=0;row<R;row++){
        for(int col=0;col<C;col++){
           scanf("%s",str);  //getting the input as string to avoid space
problem("%d ") when we get elements as integer
           matrix[row][col]=str[0];
        }
    }
    scanf("%s",word);
    for(int row=0;row<R;row++){
        for(int col=0;col<C;col++){
             if(matrix[row][col] == word[0]){
                 search(matrix,row,col,word,0);
                 if(found){
                     printf("yes");
                     return;
                 }
             }
        }
    }
    printf("no");
}
```

SESSION 2:

Problem 1:



Array Rotation Left R times

ID:11096 Solved By 899 Users

You must implement the function **rotate(int arr[],int N,int R)** which accepts an integer array **arr** with it's size **N** and an integer **R** as the input. The function must rotate the array by shifting it R times to the left.

Boundary Condition(s):
1 <= N <= 10^5
1 <= Array element value <= 10^4
1 <= R <= 10^8

Example Input/Output 1:
Input:
4
10 20 30 40
1000002

Output:
30 40 10 20

Explanation:
Here R = **2**
After the **first** left-rotation, the integers in the array become 20 30 40 10
After the **second** left-rotation, the integers in the array become 30 40 10 20
Hence the output is 30 40 10 20

Example Input/Output 2:
Input:
7
76 74 18 17 45 29 11
5

Output:
29 11 76 74 18 17 45

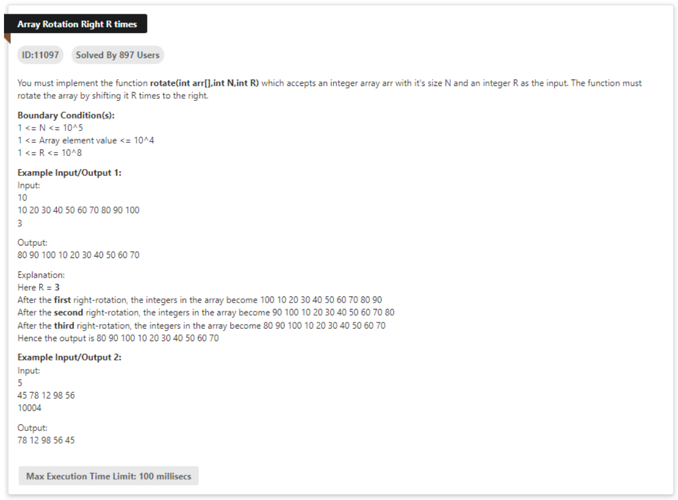
Max Execution Time Limit: 100 millisecs

```c
#include<stdio.h>
void reverse(int arr[], int from, int to){
    while(from<to){
        int temp=arr[from];
        arr[from]=arr[to];
        arr[to]=temp;
        from++;
        to--;
    }
}

void rotate(int arr[],int N,int R){
    R=R%N;
    reverse(arr,0,N-1);
    reverse(arr,0,N-R-1);
    reverse(arr,N-R,N-1);
}
int main()
{
    int N,R;
    scanf("%d",&N);
    int arr[N];
    for(int index=0; index<N; index++)
    {
        scanf("%d",&arr[index]);
    }
    scanf("%d",&R);
    rotate(arr,N,R);
    for(int index=0; index<N; index++)
    {
        printf("%d ",arr[index]);
    }
}
```

19it019@tce

```c
void reverse(int arr[], int from, int to){
    while(from<to){
        int temp=arr[from];
        arr[from]=arr[to];
        arr[to]=temp;
        from++;
        to--;
    }
}

void rotate(int arr[],int N,int R){
    R=R%N;
    reverse(arr,0,N-1);
    reverse(arr,0,N-R-1);
    reverse(arr,N-R,N-1);
}
```

Problem 2:



Array Rotation Right R times

ID:11097 Solved By 897 Users

You must implement the function **rotate(int arr[],int N,int R)** which accepts an integer array arr with it's size N and an integer R as the input. The function must rotate the array by shifting it R times to the right.

Boundary Condition(s):
1 <= N <= 10^5
1 <= Array element value <= 10^4
1 <= R <= 10^8

Example Input/Output 1:
Input:
10
10 20 30 40 50 60 70 80 90 100
3

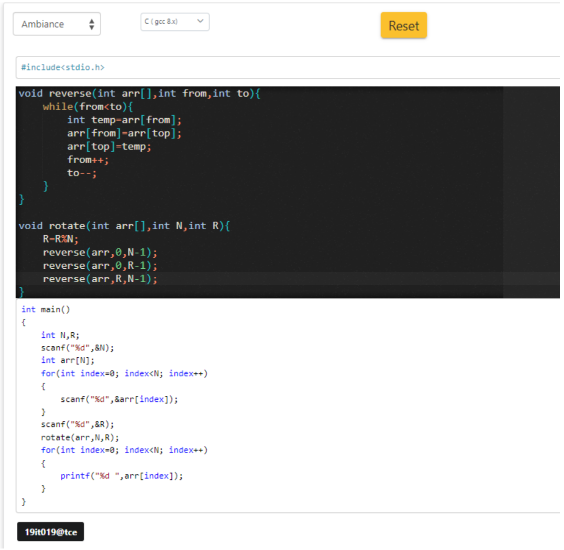
Output:
80 90 100 10 20 30 40 50 60 70

Explanation:
Here R = **3**
After the **first** right-rotation, the integers in the array become 100 10 20 30 40 50 60 70 80 90
After the **second** right-rotation, the integers in the array become 90 100 10 20 30 40 50 60 70 80
After the **third** right-rotation, the integers in the array become 80 90 100 10 20 30 40 50 60 70
Hence the output is 80 90 100 10 20 30 40 50 60 70

Example Input/Output 2:
Input:
5
45 78 12 98 56
10004

Output:
78 12 98 56 45

Max Execution Time Limit: 100 millisecs



```c
#include<stdio.h>
void reverse(int arr[],int from,int to){
    while(from<to){
        int temp=arr[from];
        arr[from]=arr[top];
        arr[top]=temp;
        from++;
        to--;
    }
}

void rotate(int arr[],int N,int R){
    R=R%N;
    reverse(arr,0,N-1);
    reverse(arr,0,R-1);
    reverse(arr,R,N-1);
}
int main()
{
    int N,R;
    scanf("%d",&N);
    int arr[N];
    for(int index=0; index<N; index++)
    {
        scanf("%d",&arr[index]);
    }
    scanf("%d",&R);
    rotate(arr,N,R);
    for(int index=0; index<N; index++)
    {
        printf("%d ",arr[index]);
    }
}
```

19it019@tce

Code:

```c
void reverse(int arr[],int from,int to){
    while(from<to){
        int temp=arr[from];
        arr[from]=arr[to];
        arr[to]=temp;
        from++;
        to--;
    }
}

void rotate(int arr[],int N,int R){
    R=R%N;
    reverse(arr,0,N-1);
    reverse(arr,0,R-1);
    reverse(arr,R,N-1);
}
```

Extras:

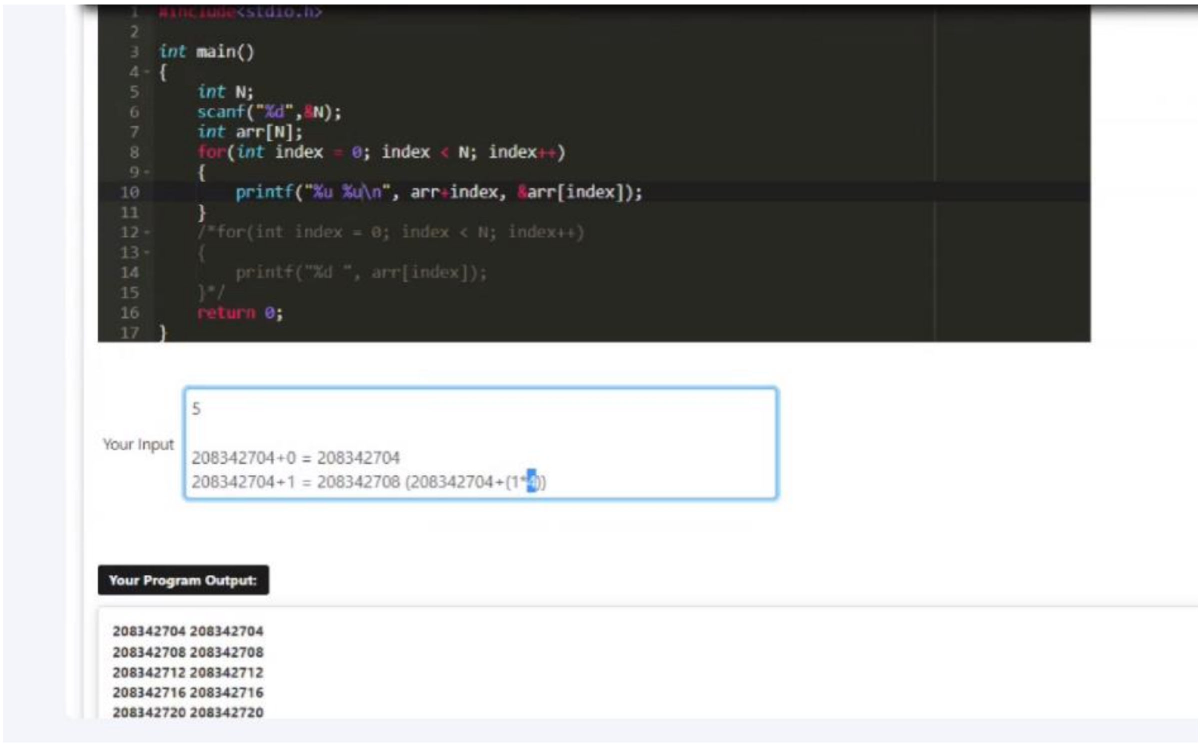
C POINTERS:

The size of the pointer varies on the machine (32bit or 64bit)

The size of the pointer in 64 bit machine is 8 bytes, the size does not change with data types (int, char, etc..)

We can also get the input of array element as below:

Pointer addition done below:



```c
#include<stdio.h>

int main()
{
    int N;
    scanf("%d",&N);
    int arr[N];
    for(int index = 0; index < N; index++)
    {
        printf("%u %u\n", arr+index, &arr[index]);
    }
    /*for(int index = 0; index < N; index++)
    {
        printf("%d ", arr[index]);
    }*/
    return 0;
}
```

Your Input

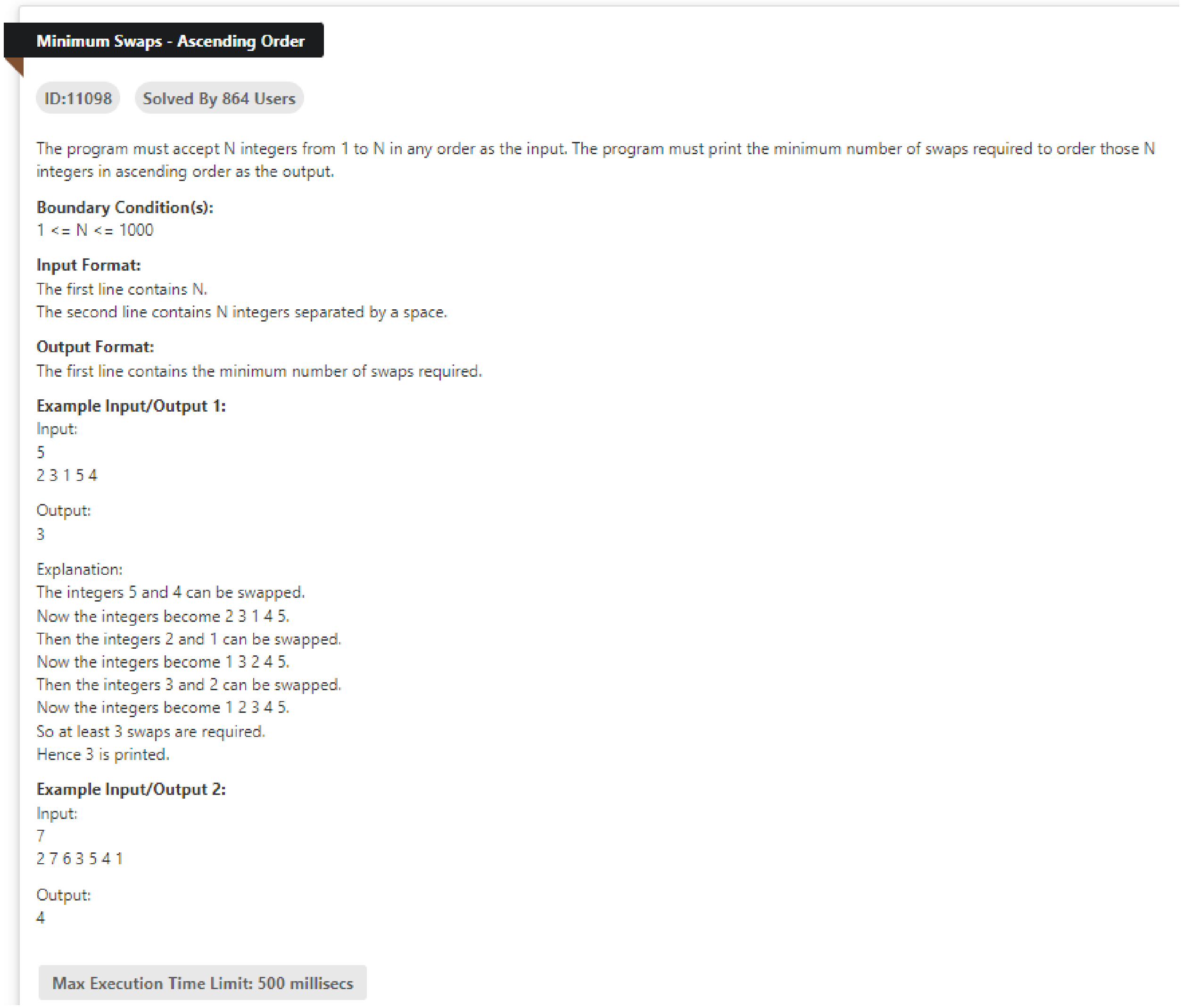
```
5

208342704+0 = 208342704
208342704+1 = 208342708 (208342704+(1*4))
```

Your Program Output:

```
208342704 208342704
208342708 208342708
208342712 208342712
208342716 208342716
208342720 208342720
```

SESSION 3:

Problem 1:



Minimum Swaps - Ascending Order

ID:11098 Solved By 864 Users

The program must accept N integers from 1 to N in any order as the input. The program must print the minimum number of swaps required to order those N integers in ascending order as the output.

Boundary Condition(s):
1 <= N <= 1000

Input Format:
The first line contains N.
The second line contains N integers separated by a space.

Output Format:
The first line contains the minimum number of swaps required.

Example Input/Output 1:
Input:
```
5
2 3 1 5 4
```
Output:
```
3
```
Explanation:
The integers 5 and 4 can be swapped.
Now the integers become 2 3 1 4 5.
Then the integers 2 and 1 can be swapped.
Now the integers become 1 3 2 4 5.
Then the integers 3 and 2 can be swapped.
Now the integers become 1 2 3 4 5.
So at least 3 swaps are required.
Hence 3 is printed.

Example Input/Output 2:
Input:
```
7
2 7 6 3 5 4 1
```
Output:
```
4
```

Max Execution Time Limit: 500 millisecs

Stimulations:

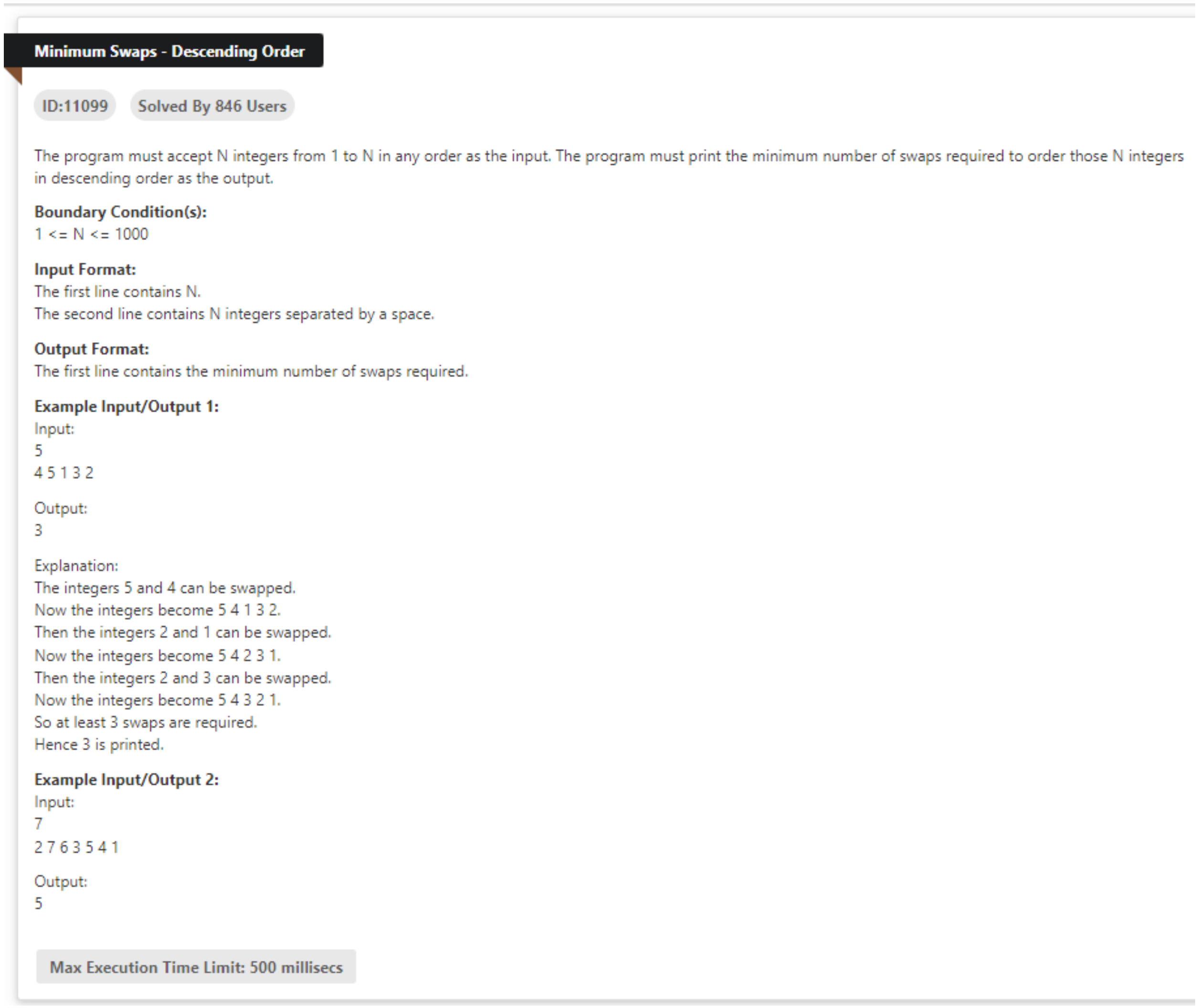
https://cscircles.cemc.uwaterloo.ca/java_visualize/#code=import+java.util.*%3B%0Apublic+class+minimumSwapsAscendingOrder+%7B%0A%0A++++public+static+void+main(String%5B%5D+args)+%7B%0A++++int+N%3D+5%3B%0A++++int+arr%5B%5D+%3D+%7B0,2,3,1,5,4%7D%3B%0A++++boolean+visited%5B%5D+%3D+new+boolean%5BN%2B1%5D%3B%0A++++int+totalSwap%3D0%3B%0A++++for(int+index%3D1%3Bindex%3C%3DN%3Bindex%2B%2B)%7B%0A++++++++if(visited%5Barr%5Bindex%5D%5D)%7B++++//if+already+visited%0A++++++++++++continue%3B%0A++++++++%7D%0A++++++++if(arr%5Bindex%5D%3D%3Dindex)%7B+++++//if+2%3D%3D2+change+to+true+and+continue++%0A++++++++++++visited%5Barr%5Bindex%5D%5D%3Dtrue%3B%0A++++++++++++continue%3B%0A++++++++%7D%0A++++++

++int+edges%3D0,cycleIndex%3Dindex%3B%0A++++++++while(!visited%5Barr%5BcycleIndex%5D%5D)%7B%0A++++++++++++visited%5Barr%5BcycleIndex%5D%5D%3Dtrue%3B%0A++++++++++++edges%2B%2B%3B%0A++++++++++++cycleIndex%3Darr%5BcycleIndex%5D%3B%0A++++++++%7D%0A++++++++totalSwap%2B%3Dedges-1%3B%0A++++%7D%0A++++System.out.println(totalSwap)%3B%0A%09%7D+%0A%7D&mode=display&curInstr=62

Code:

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

    public static void main(String[] args) {
        //Your Code Here
    Scanner in = new Scanner(System.in);
    int N= in.nextInt();
    int arr[] = new int[N+1];
    for(int index=1;index<=N;index++){
        arr[index]=in.nextInt();
    }
    boolean visited[] = new boolean[N+1];
    int totalSwap=0;
    for(int index=1;index<=N;index++){
        if(visited[arr[index]]){    //if already visited
             continue;
        }
        if(arr[index]==index){     //if 2==2 change to true and continue
            visited[arr[index]]=true;
            continue;
        }
        int edges=0,cycleIndex=index;
        while(!visited[arr[cycleIndex]]){
            visited[arr[cycleIndex]]=true;
            edges++;
            cycleIndex=arr[cycleIndex];
        }
        totalSwap+=edges-1;
    }
    System.out.println(totalSwap);
    }
}
```

Problem 2:



Minimum Swaps - Descending Order

ID:11099 Solved By 846 Users

The program must accept N integers from 1 to N in any order as the input. The program must print the minimum number of swaps required to order those N integers in descending order as the output.

Boundary Condition(s):
1 <= N <= 1000

Input Format:
The first line contains N.
The second line contains N integers separated by a space.

Output Format:
The first line contains the minimum number of swaps required.

Example Input/Output 1:
Input:
5
4 5 1 3 2

Output:
3

Explanation:
The integers 5 and 4 can be swapped.
Now the integers become 5 4 1 3 2.
Then the integers 2 and 1 can be swapped.
Now the integers become 5 4 2 3 1.
Then the integers 2 and 3 can be swapped.
Now the integers become 5 4 3 2 1.
So at least 3 swaps are required.
Hence 3 is printed.

Example Input/Output 2:
Input:
7
2 7 6 3 5 4 1

Output:
5

Max Execution Time Limit: 500 millisecs

Code:

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

    public static void main(String[] args) {
        //Your Code Here
        Scanner in = new Scanner(System.in);
        int N=in.nextInt();
        int arr[] = new int[N+1];
        for(int index=N;index>=1;index--){ //getting the matrix in reverse order
            arr[index]=in.nextInt();
        }
        boolean visited[] = new boolean[N+1];
        int totalswaps=0;
        for(int index=1;index<=N;index++){
            if(visited[arr[index]]){
                continue;
            }
            if(arr[index]==index){
                visited[arr[index]]=true;
                continue;
            }
            int edges=0,cycleIndex=index;
            while(!visited[arr[cycleIndex]]){
                visited[arr[cycleIndex]]=true;
                edges++;
                cycleIndex=arr[cycleIndex];
            }
            totalswaps+=edges-1;
        }
        System.out.println(totalswaps);
    }
}
```