Walchand College of Engineering, Sangli

Computer Science & Engineering

Third Year

Course: Design and Analysis of Algorithm Lab

Lab course coordinator: Ms N L Mudegol

Batch: T6

Week 2 Assignment

Sorting Algorithm

Q.1You are an IT company's manager. Based on their performance over the last N working days, you must rate your employee. You are given an array of N integers called workload, where workload[i] represents the number of hours an employee worked on an ith day. The employee must be evaluated using the following criteria: • Rating = the maximum number of consecutive working days when the employee has worked more than 6 hours. You are given an integer N where N represents the number of working days. You are given an integer array workload where workload[i] represents the number of hours an employee worked on an ith day. Task Determine the employee rating.

Approach:

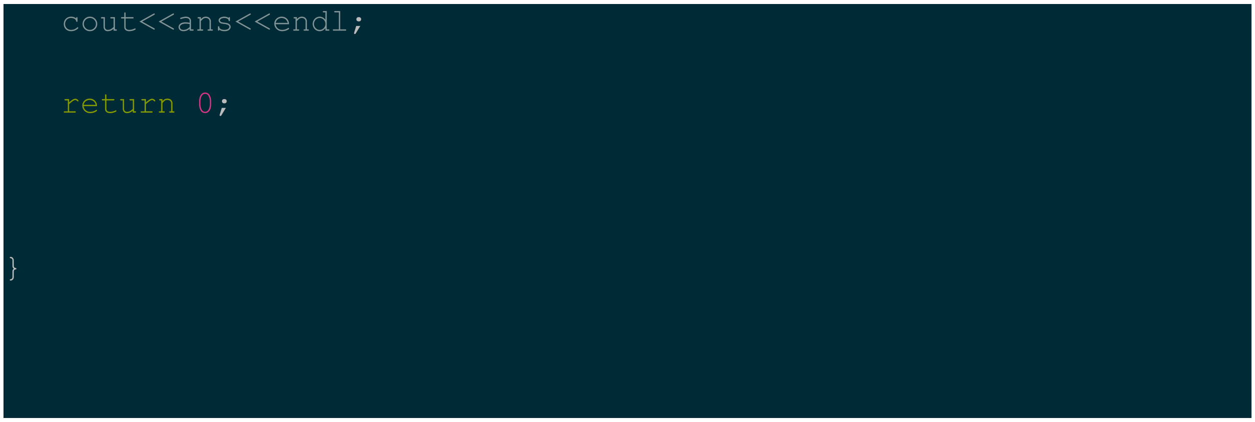
- Mark all days with more than 6 hours of working as 1 otherwise 0 (make a binary array)

- Maintain a sum variable which get you the number of consecutive 1s

- compare it with your answer variable and update the maximum

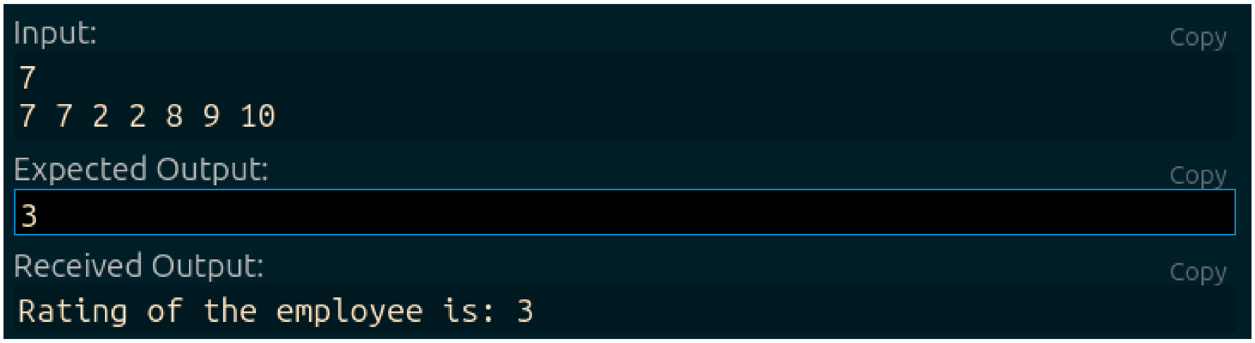
```
#include <bits/stdc++.h>

using namespace std;

int main(){
   int n;
   cout << "Enter the number of days: ";
   cin>>n;
   vector<int>v(n);
   cout << "Enter the working hour per day: ";
   for(int i=0;i<n;i++){
       cin>>v[i];
   }
   int ans=0;
   vector<int>bin;
```

```
for(int i=0;i<n;i++){
    if(v[i]>6){
        bin.push_back(1);
    }
    else{
        bin.push_back(0);
    }
}
int sum = 0;

for(int i=0;i<n;i++){
    if(bin[i]==1){
        sum++;
    }
    else{
        ans = max(ans,sum);
        sum=0;
    }
}
ans = max(ans,sum);
cout<<"Rating of the employee is: ";
```



```
    cout<<ans<<endl;

    return 0;


}
```

Output :



```
Input:
7
7 7 2 2 8 9 10
Expected Output:
3
Received Output:
Rating of the employee is: 3
```

Q.2 You have N boxes numbered 1 through N and K candies numbered 1 through K. You put the candies in the boxes in the following order: • first candy in the first box, • second candy in the second box, • .... • .... • so up to N-th candy in the Nth box, • the next candy in (N - 1)-th box, • the next candy in (N - 2)-th box • .... • .... • and so on up to the first box, • then the next candy in the second box • ...... and so on until there is no candy left. So you put the candies in the boxes in the following order: Find the index of the box where you put the K-th candy.

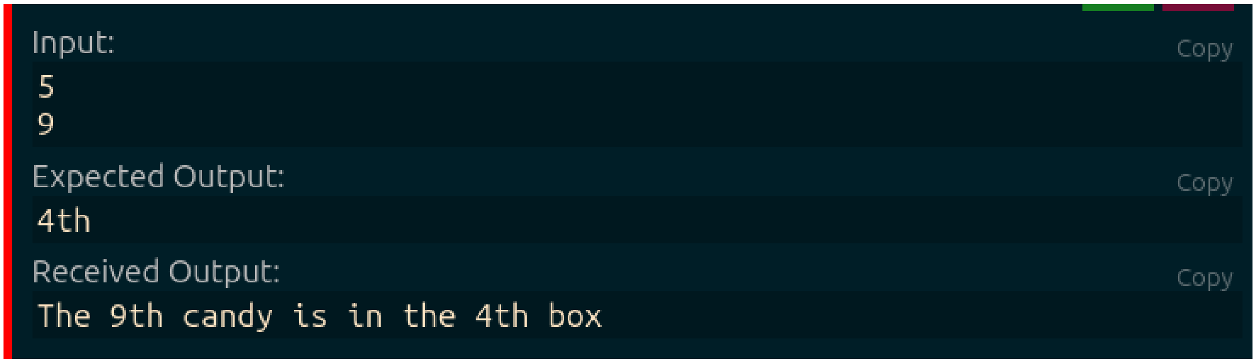
Approach: Keep adding the value of each array index and increment the index and decreament the value of k until it reaches 0.

If index goes out of bound modulo it by the size of the array .

Return the index when the k becomes 0 , that is your answer.

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

using namespace std;

int main(){
   int n;
   cout << "Enter the number of boxes: ";
   cin>>n;
   int k;
   cout << "Enter the number of candies: ";
   cin>>k;
   int a[n];
   int idx = 0;
   while(k--){
       a[idx]++;
       idx = (idx+1)%n;

   }
   cout<<"The Kth candy is in the "<<idx<<"th box"<<endl;
   return 0;

}
```

Output :



```
Input:
5
9
Expected Output:
4th
Received Output:
The 9th candy is in the 4th box
```

Q.3 Implement and Explain Tower of Hanoi algorithm.

The Tower of Hanoi is a classic problem that involves moving a set of disks from one peg to another, using a third peg as an auxiliary. The challenge is to move all the disks from the source peg to the destination peg following these rules:

1. Only one disk can be moved at a time.
2. A disk can only be placed on top of a larger disk.
3. All disks must be moved from the source peg to the destination peg.

Problem Setup

- **Source Peg (A)**
- **Auxiliary Peg (B)**
- **Destination Peg (C)**
- **n:** Number of disks

Recursive Solution

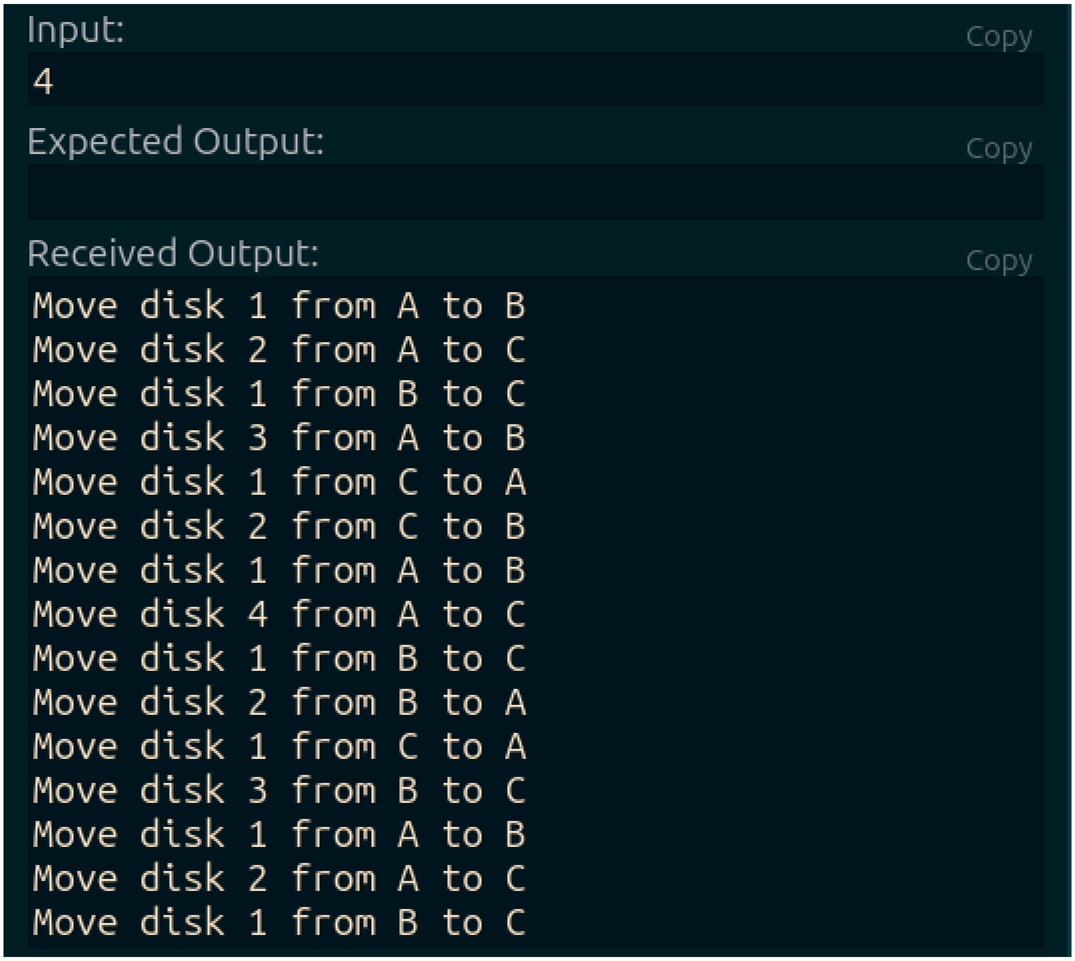
The problem can be solved using a recursive approach. The key idea is to break down the problem into smaller subproblems:

1. Move the top `n-1` disks from the source peg to the auxiliary peg.
2. Move the `nth` disk from the source peg to the destination peg.
3. Move the `n-1` disks from the auxiliary peg to the destination peg.

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

// Function to move disks
void moveDisk(int n, char source, char destination, char
auxiliary) {
   if (n == 1) {
       // Base case: move the only disk from source to
destination
       cout << "Move disk 1 from " << source << " to " <<
destination << endl;
       return;
   }

   // Move n-1 disks from source to auxiliary
   moveDisk(n - 1, source, auxiliary, destination);
```

```
    // Move the nth disk from source to destination
    cout << "Move disk " << n << " from " << source << " to " <<
destination << endl;

    // Move the n-1 disks from auxiliary to destination
    moveDisk(n - 1, auxiliary, destination, source);
}

int main() {
    int n; // Number of disks
    cout << "Enter the number of disks: ";
    cin >> n;

    // Call the moveDisk function
    moveDisk(n, 'A', 'C', 'B');
    return 0;
}
```

Output :



```
4
```

```
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
```

Q4.

There is a frog initially placed at the origin of the coordinate plane. In exactly 1 second, the frog can either move up 1 unit, move right 1 unit, or stay still. In other words, from position (x, y), the frog can spend 1 second to move to:

- $(x + 1, y)$
- $(x, y + 1)$
- (x, y)

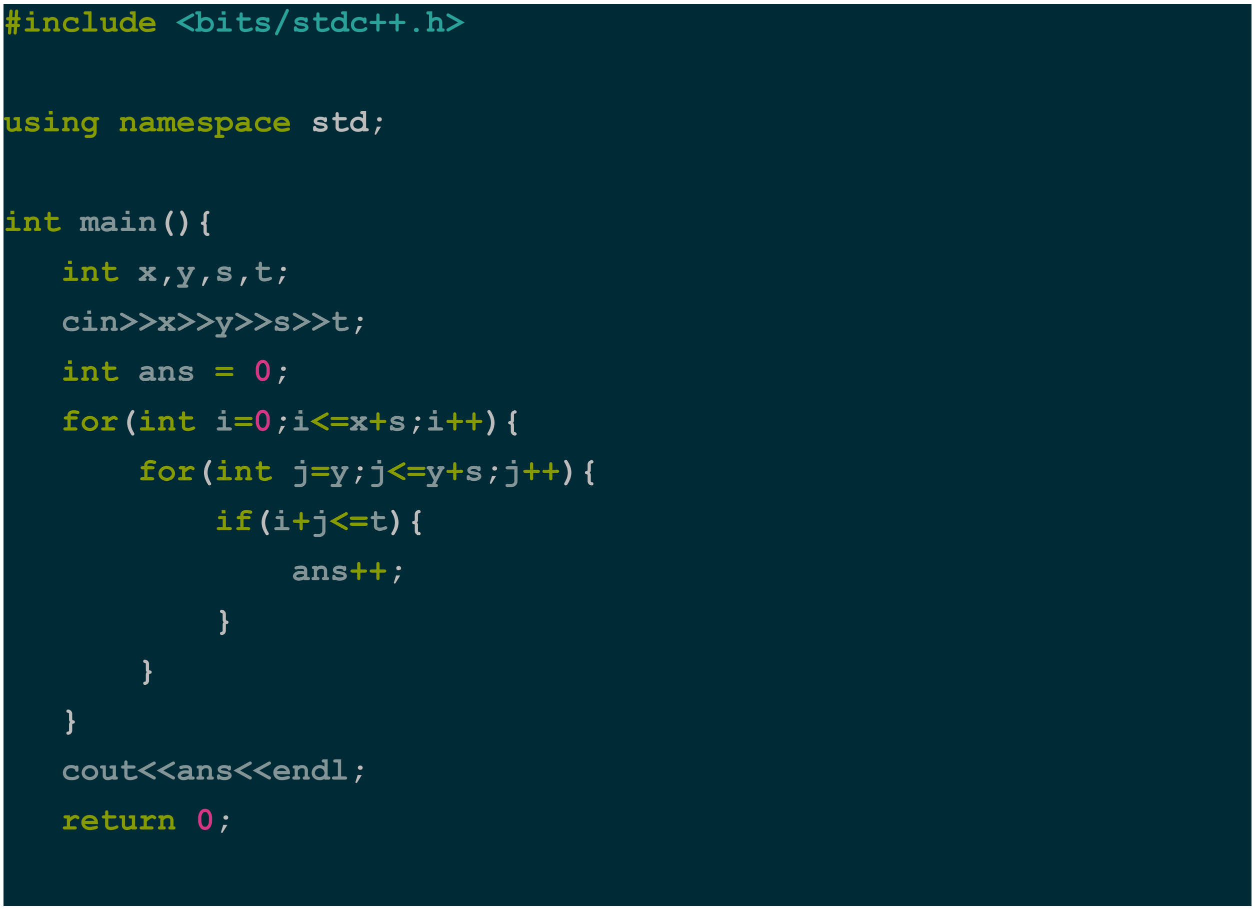
After T seconds, a villager who sees the frog reports that the frog lies on or inside a square of side-length s with coordinates (X, Y), $(X + s, Y)$, $(X, Y + s)$, $(X + s, Y + s)$. Calculate how many points with integer coordinates on or inside this square could be the frog's position after exactly T seconds

Input Format:

The first and only line of input contains four space-separated integers: X, Y, s, and T.

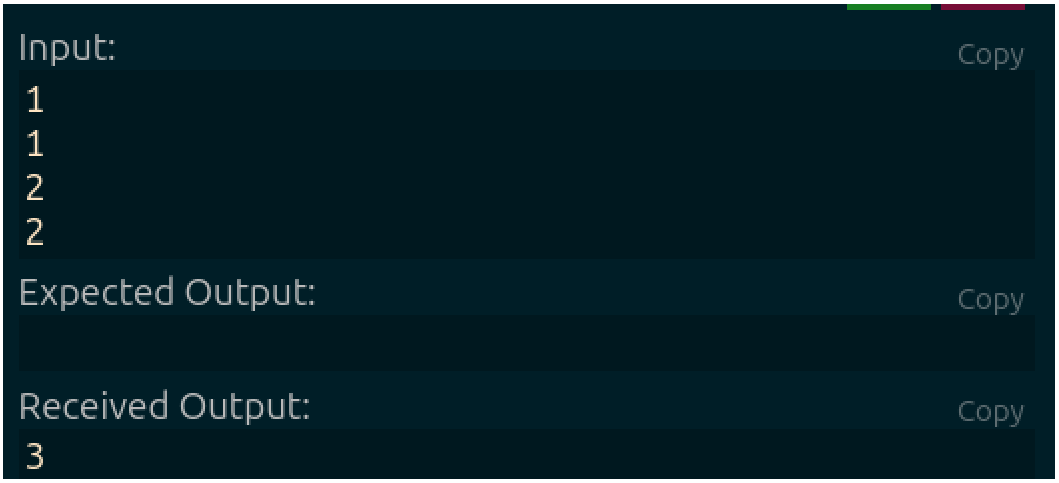
Output Format:

Print the number of points with integer coordinates that could be the frog's position after T seconds.



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

using namespace std;

int main(){
   int x,y,s,t;
   cin>>x>>y>>s>>t;
   int ans = 0;
   for(int i=0;i<=x+s;i++){
       for(int j=y;j<=y+s;j++){
           if(i+j<=t){
               ans++;
           }
       }
   }
   cout<<ans<<endl;
   return 0;
}
```

Output :



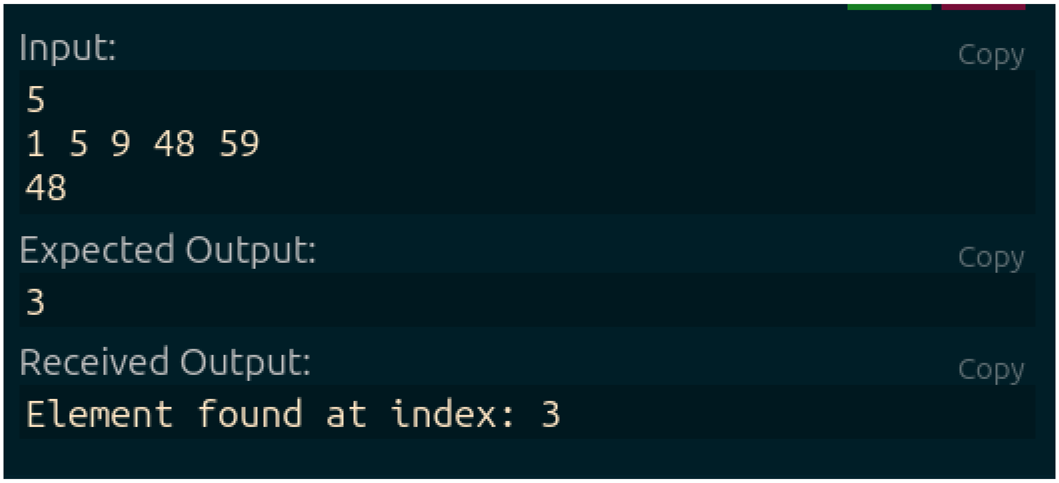
```
Input:
1
1
2
2
Expected Output:

Received Output:
3
```

Q.5 Implement linear search Algorithm.

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

using namespace std;

int main(){
   int n;
   cin>>n;
   vector<int>v(n);
   for(auto &x:v){
       cin>>x;
   }
   int find_element;
   cin>>find_element;

   for(int i=0;i<n;i++){
       if(v[i]==find_element){
           cout<<"Element found at index: ";
```

```
            cout<<i;
            return 0;
        }
    }

    return 0;

}
```

Output :

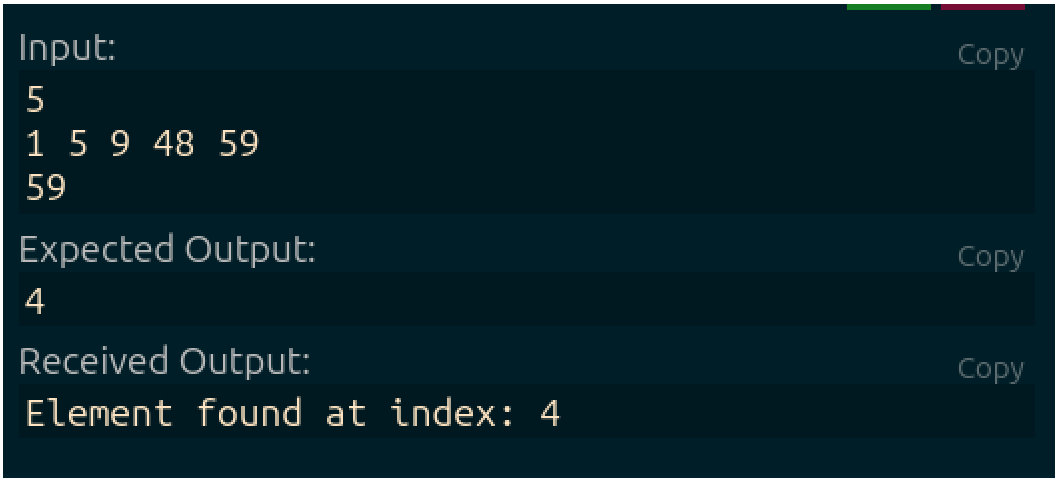


```
Input:
5
1 5 9 48 59
48
Expected Output:
3
Received Output:
Element found at index: 3
```

Q.6 Implement Binary Search algorithm.

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

using namespace std;

int main(){
    // cout<<"Enter the number of elements in the sorted array:
";
    int n;
    cin>>n;
    vector<int>v(n);
    // cout<<"Enter the elements of the sorted array: ";
    for(int i=0;i<n;i++){
        cin>>v[i];
    }
    int find_element;
    // cout<<"Enter the element to be found: ";
    cin>>find_element;
    int left = 0;
    int right = n-1;

    while(left<=right){
        int mid = (left+right)/2;
        if(v[mid]==find_element){
            cout<<"Element found at index: ";
            cout<<mid<<endl;
            return 0;
        }
        else if(v[mid]<find_element){
            left = mid+1;
        }
        else{
            right = mid-1;
        }
    }
    return 0;

}
```

Output :



Input:

```
5
1 5 9 48 59
59
```

Expected Output:

```
4
```

Received Output:

```
Element found at index: 4
```