<u>**NAME:**</u> Venkateswar L

<u>**ROLL NUMBER:**</u> 230701376

<u>**SECTION:**</u> CSE-F

Design and Analysis Of Algorithms
CS23331

WEEK 4: DIVIDE AND CONQUER

PROGRAM 1:

AIM: Given an array of 1s and 0s this has all 1s first followed by all 0s. Aim is to find the number of 0s. Write a program using Divide and Conquer to Count the number of zeroes in the given array.

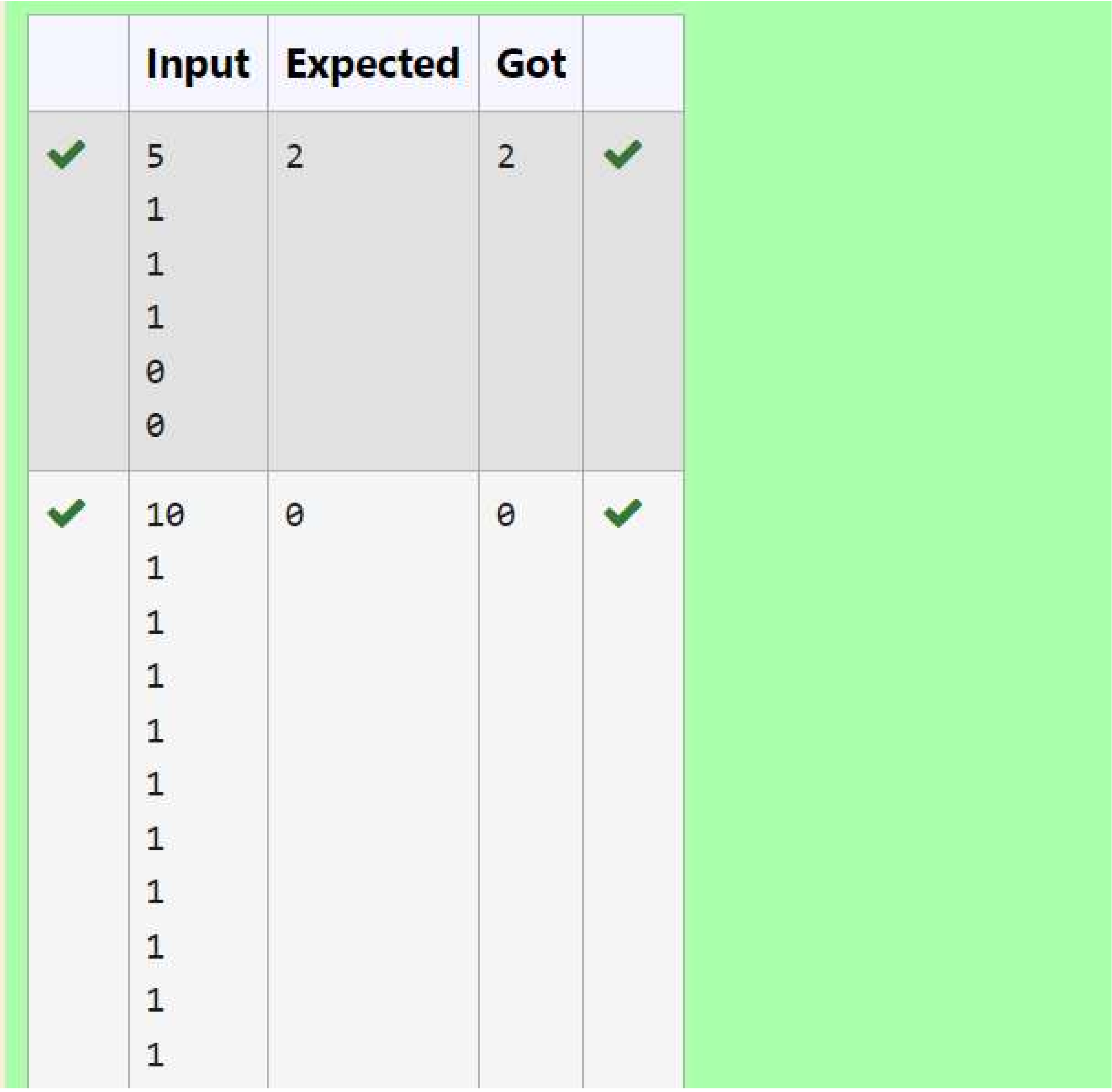
ALGORITHM:

Step 1: Input the size of the array and the elements.
Step 2: Define the recursive divide function to find the first occurrence of 1.
Step 3: Call the divide function and compute the result.
Step 4: Output the result.

PROGRAM:

```
#include <stdio.h>
int divide(int [],int,int);
int divide(int a[],int left,int right)
{
   int mid=0;
   mid=left+(right-left)/2;
   if (a[0]==0)
      return 0;
   else if (a[right-1]==1)
      return right;
   if ((a[mid]==0) && (a[mid-1]==0))
      return divide(a,0,mid);
   else if (a[mid]==0)
      return mid;
   else
      return divide(a,mid+1,right);
}

int main()
{
   int n;
   scanf("%d",&n);
   int arr[n];
   for (int i=0;i<n;i++)
   {
        scanf("%d",&arr[i]);
    }
   int zero=divide(arr,0,n);
   printf("%d",n-zero);

}
```

OUTPUT:



	Input	Expected	Got	
✓	5 1 1 1 0 0	2	2	✓
✓	10 1 1 1 1 1 1 1 1 1 1	0	0	✓

RESULT: Thus the program is executed successfully.

PROGRAM 2:

AIM: Given an array nums of size n, return *the majority element*.
The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

ALGORITHM:

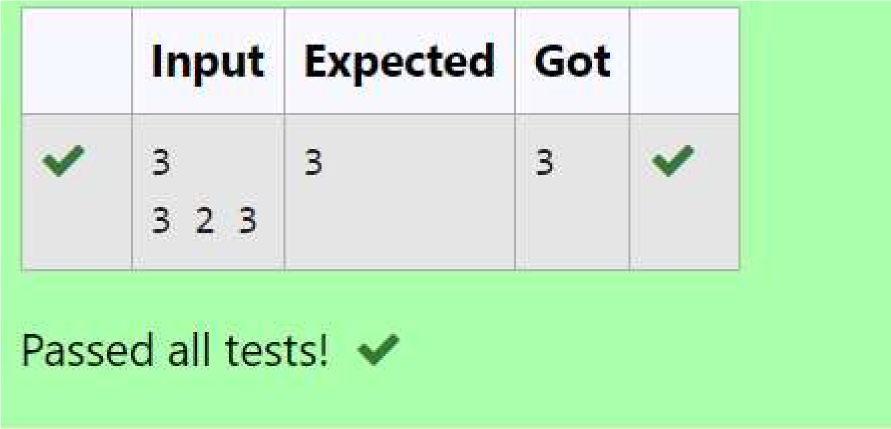
Step 1: Input the size of the array and its elements.
Step 2: Define the recursive function Count to count occurrences of a specific element (key).
Step 3: Find the majority element and check if its count exceeds half the array size.
Step 4: Handle edge cases where k is not the majority element.
Step 5: Display the output

PROGRAM:

```
#include<stdio.h>

#include <stdio.h>
int mid=0,c=0;
int Count(int [],int,int,int);
int Count(int a[],int left,int right,int key)
{
   int mid=left+(right-left)/2;
   if (a[mid]==key)
      c++;
   else
   {
      Count(a,left,mid,key);
      Count(a,mid+1,right,key);
   }
   return c;

}

int main()
{
   int n;
   scanf("%d",&n);
   int arr[n];
   for (int i=0;i<n;i++)
       scanf("%d",&arr[i]);
   int k=arr[0];
   if (Count(arr,0,n,k)>n/2)
```

```
          printf("%d",k);
    else
    {
        for (int i=0;i<n/2;i++)
           if (arr[i]!=k)
            {
                printf("%d",k);
                break;
            }
    }

}
```

<u>OUTPUT:</u>



	Input	Expected	Got	
✓	3 3 2 3	3	3	✓

Passed all tests! ✓

<u>RESULT:</u> Thus the program is executed successfully.

PROGRAM 3:

AIM: Given a sorted array and a value x, the floor of x is the largest element in array smaller than or equal to x. Write divide and conquer algorithm to find floor of x.

ALGORITHM:

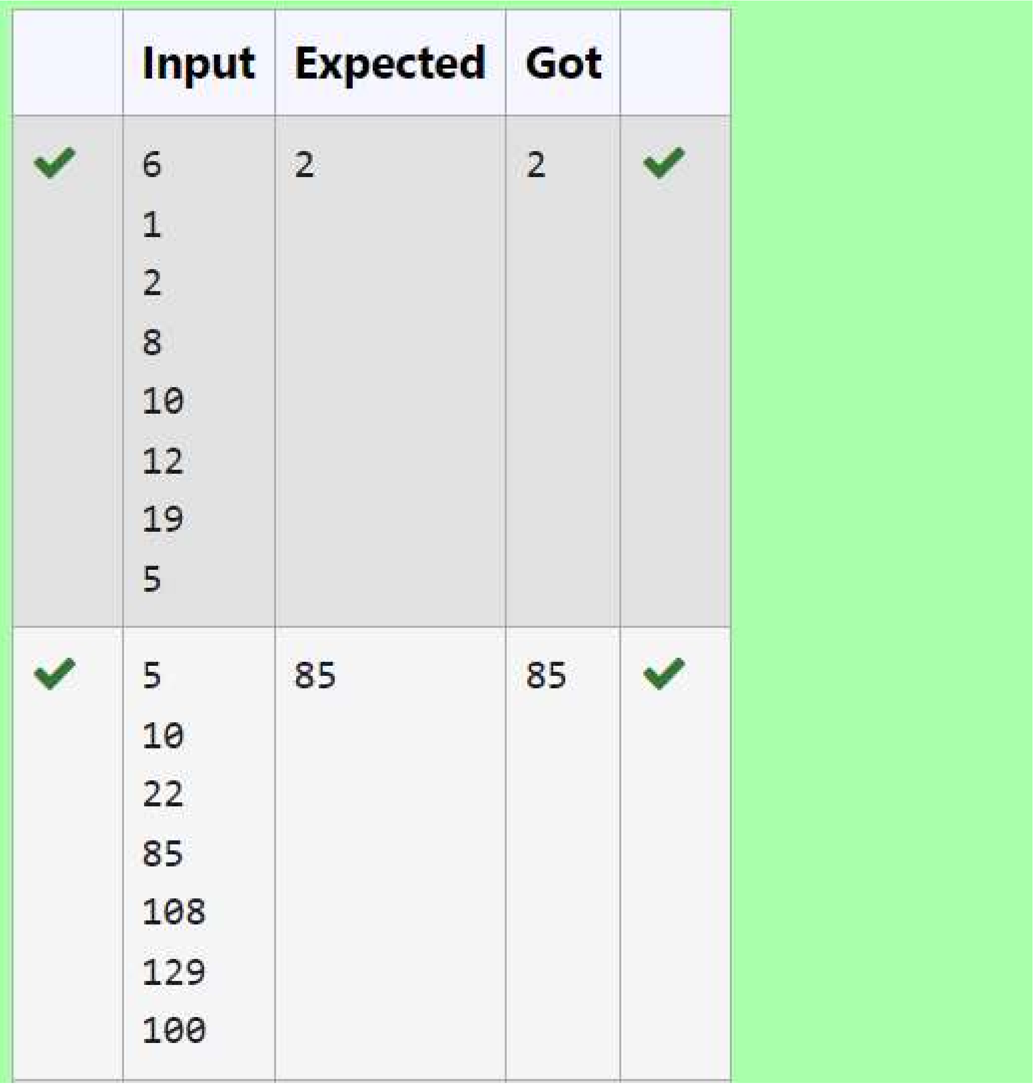
Step 1: Input the size of the array and its elements.
Step 2: Define the search function to find the largest element smaller than or equal to x.
Step 3: Call the search function and get the result.
Step 4: Output the result.

PROGRAM:

```
#include<stdio.h>

int search(int arr[], int n, int x)
{
   if (x>=arr[n-1])
      return n-1;
   if (x<arr[0])
      return -1;

   for (int i=1;i<n;i++)
      if (arr[i]>x)
         return arr[i-1];

   return -1;
}

int main()
{
   int n;
   scanf("%d",&n);
   int a[n];
   for (int i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   int x;
   scanf("%d",&x);
   int res=search(a, n, x);
   if (res!=-1)
```

```
    printf("%d",res);
}
```

OUTPUT:



	Input	Expected	Got	
✓	6 1 2 8 10 12 19 5	2	2	✓
✓	5 10 22 85 108 129 100	85	85	✓

RESULT: Thus the program is executed successfully.

PROGRAM 4:

AIM: Given a sorted array of integers say arr[] and a number x. Write a recursive program using divide and conquer strategy to check if there exist two elements in the array whose sum = x. If there exist such two elements then return the numbers, otherwise print as “No”.
Note: Write a Divide and Conquer Solution

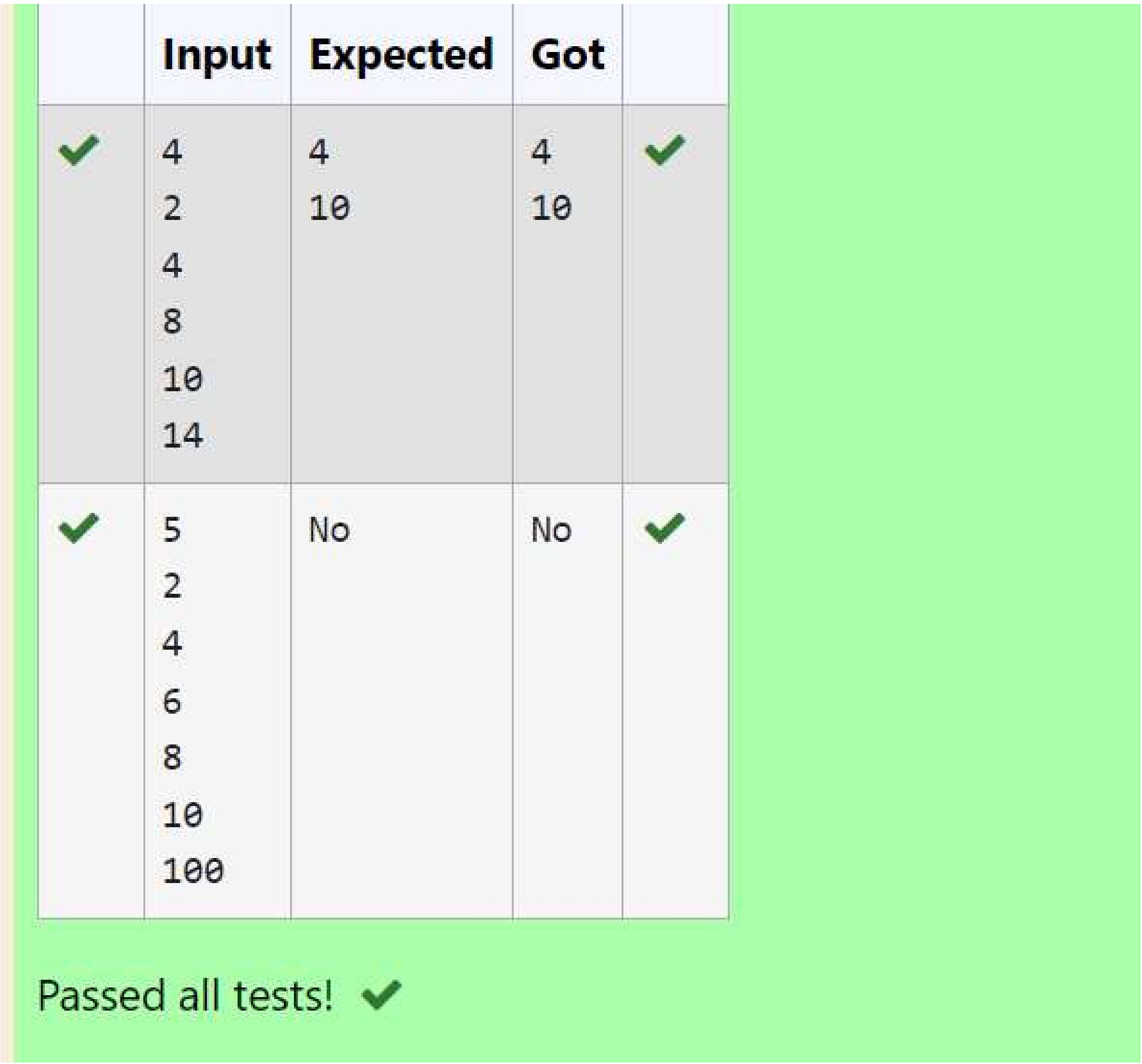
ALGORITHM:

Step 1: Input the size of the array and its elements.
Step 2: Define the `sum` function to find two elements whose sum equals `x`.
Step 3: Call the `sum` function to find the pair.
Step 4: Output the result.

PROGRAM:

```
#include<stdio.h>
void sum(int a[], int l, int r, int x)
{
    if (l>=r)
    {
        printf("No\n");
        return;
    }

    int ts=a[l]+a[r];

    if (ts==x)
    {
        printf("%d\n",a[l]);
        printf("%d\n",a[r]);
    }
    else if (ts<x)
    {
        sum(a,l+1,r,x);
    }
    else
    {
        sum(a,l,r-1,x);
    }
}

int main()
{
    int n,x;
    scanf("%d",&n);
    int arr[n];
    for (int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    scanf("%d",&x);
    sum(arr,0,n-1,x);
}
```

OUTPUT:



	Input	Expected	Got	
✓	4 2 4 8 10 14	4 10	4 10	✓
✓	5 2 4 6 8 10 100	No	No	✓

Passed all tests! ✓

RESULT: Thus the program is executed successfully.

PROGRAM 5:

AIM: Write a Program to Implement the Quick Sort Algorithm

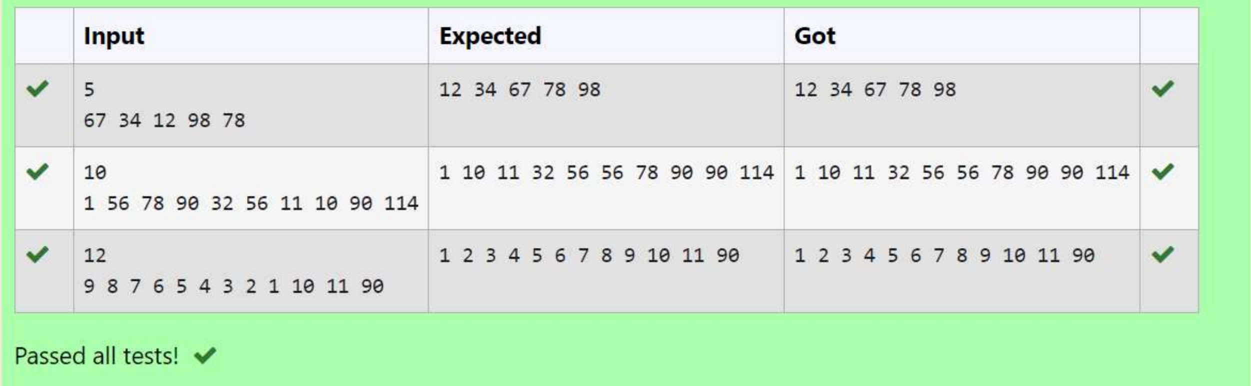
ALGORITHM:

Step 1: Input the array size and elements.
Step 2: Define the swap function.
Step 3: Define the partition function.
Step 4: Define the quicksort function.
Step 5: Call the quicksort function in the main() function.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>

void swap(int *p1, int *p2)
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

int partition(int a[], int low, int high)
{
    int p = a[high];
    int i = low - 1;
    for (int j = low; j < high; j++)
    {
        if (a[j] < p)
        {
            i++;
            swap(&a[i], &a[j]);
        }
    }
    swap(&a[i + 1], &a[high]);
    return (i + 1);
}

void quicksort(int a[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(a, low, high);
        quicksort(a, low, pi - 1);
        quicksort(a, pi + 1, high);
    }
}

int main()
{
    int n;
    scanf("%d", &n);
    int a[n];
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    quicksort(a, 0, n - 1);
    for (int i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    printf("\n");
    return 0;
}
```

OUTPUT:



	Input	Expected	Got	
✓	5 67 34 12 98 78	12 34 67 78 98	12 34 67 78 98	✓
✓	10 1 56 78 90 32 56 11 10 90 114	1 10 11 32 56 56 78 90 90 114	1 10 11 32 56 56 78 90 90 114	✓
✓	12 9 8 7 6 5 4 3 2 1 10 11 90	1 2 3 4 5 6 7 8 9 10 11 90	1 2 3 4 5 6 7 8 9 10 11 90	✓

Passed all tests! ✓

RESULT: Thus the program is executed successfully.