Name: Valluru Varshini

Class: CSE – F

Reg no: 230701369

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.

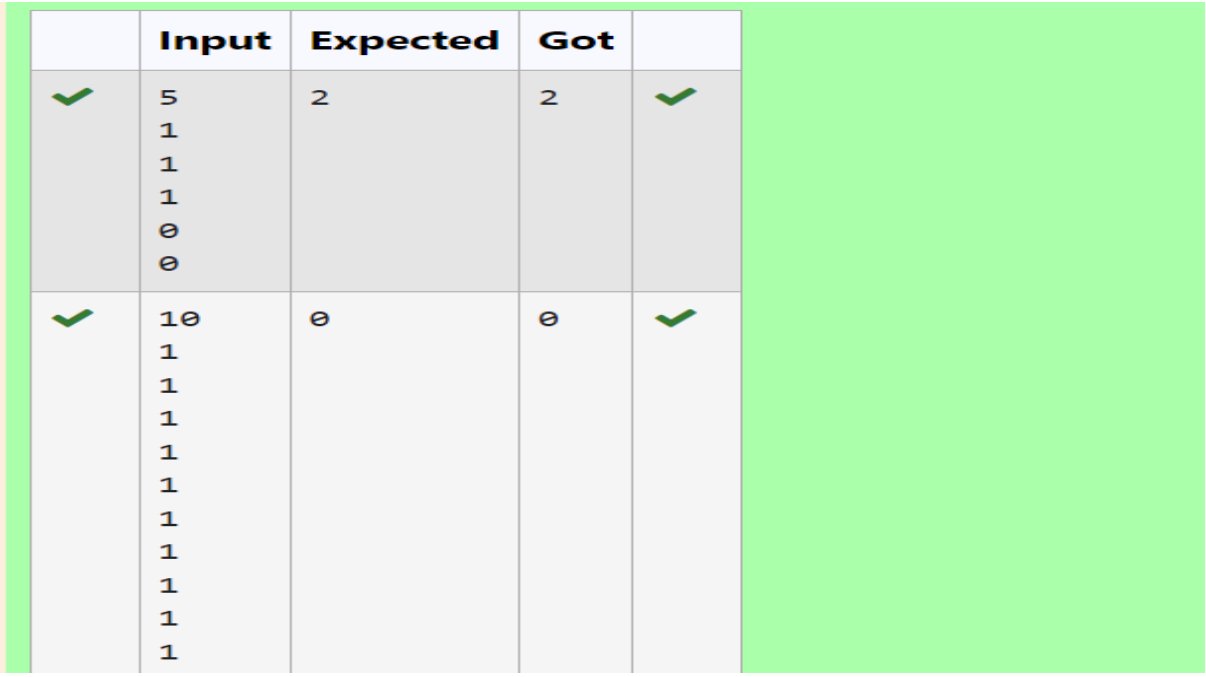
PROGRAM:

```
#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>


int divide(int arr[],int low,int high)
```

```
{
    if(arr[high]==1)
    {
        return 0;
    }
    if(arr[low]==0)
    {
        return high-low+1;
    }
    int mid=(low+high)/2;
    int left=divide(arr,low,mid);
    int right=divide(arr,mid+1,high);
    return left+right;
}
int main()
{
    int size;
    scanf("%d",&size);
    int arr[size];
    for(int i=0;i<size;i++)
```

```
    {
        scanf("%d",&arr[i]);
    }
    int count=divide(arr,0,size-1);
    printf("%d\n",count);
}
```

OUTPUT:

	Input	Expected	Got	
✔	3 3 2 3	3	3	✔

Passed all tests! ✔

RESULT: 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);
```

}

	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 merge(int arr[], int left, int mid, int right) {

    int i, j, k;
```

```
int n1 = mid - left + 1;
int n2 = right - mid;
int leftArr[n1], rightArr[n2];
for (i = 0; i < n1; i++)
    leftArr[i] = arr[left + i];
for (j = 0; j < n2; j++)
    rightArr[j] = arr[mid + 1 + j];
i = 0;
j = 0;
k = left;
while (i < n1 && j < n2) {
    if (leftArr[i] <= rightArr[j]) {
        arr[k] = leftArr[i];
        i++;
    }
    else {
        arr[k] = rightArr[j];
        j++;
    }
    k++;
```

```
    }
    while (i < n1) {
        arr[k] = leftArr[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
```

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

```
            }
        }
    }
    if(c==0)
    {
        printf("No");
    }
}
```

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>
```

```
void quicksort(int arr[],int first,int last){
   int i, j, pivot, temp;
   if(first<last){
      pivot=first;
      i=first;
      j=last;
      while(i<j){
         while(arr[i]<=arr[pivot]&&i<last)
         i++;
         while(arr[j]>arr[pivot])
         j--;
         if(i<j){
            temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
         }
      }
      temp=arr[pivot];
      arr[pivot]=arr[j];
      arr[j]=temp;
```

```
      quicksort(arr,first,j-1);
      quicksort(arr,j+1,last);
   }
}
int main()
{
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    quicksort(arr,0,n-1);
    for(int i=0;i<n;i++)
    {
        printf("%d ",arr[i]);
    }
}
```

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.