DIVIDE AND CONQUER

PROBLEM-1

AIM: 1-NUMBER OF ZEROES IN A GIVEN ARRAY

ALGORITHM:

1. Read integer `n` and array `arr[]`.

2. Define a recursive function to find the first `1` using binary search.

3. If the first element is `0`, return `0`. If the last element is `1`, return `n`.

4. Recurse to find the first `1` by checking midpoints.

5. Print `n - ZeroStart`, the number of `1`s in the array.

PROBLEM:

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.

Input Format

First Line Contains Integer m – Size of array

Next m lines Contains m numbers – Elements of an array

Output Format

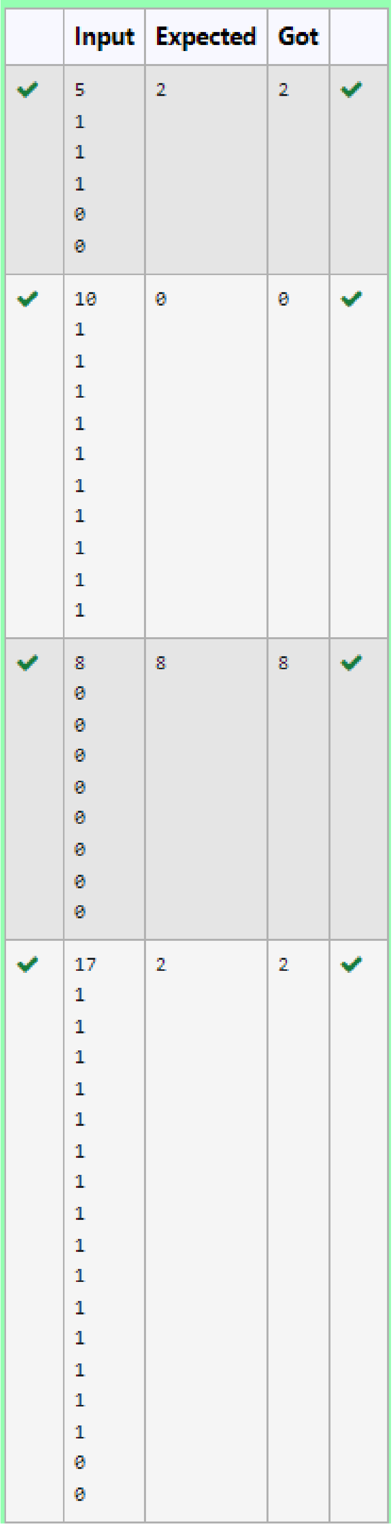
First Line Contains Integer – Number of zeroes present in the given array.

PROGRAM:

```
#include <stdio.h>

int function(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 function(a,0,mid);
    else if (a[mid]==0)
        return mid;
    else
        return function(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 ZeroStart=function(arr,0,n);
   printf("%d",n-ZeroStart);
}
```

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	✓
✓	8 0 0 0 0 0 0 0 0	8	8	✓
✓	17 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0	2	2	✓

PROBLEM-2

AIM: 2-MAJORITY ELEMENT

ALGORITHM:

1. Read integer `d` and array `arr[]` of size `d`.

2. Define a recursive function `y()` to count occurrences of a given number `c` in the array using binary search.

3. In the main function, set `f` as the first element of the array.

4. If the count of `f` is more than half the size of the array, print `f`.

5. Otherwise, print `f` only if it differs from another element in the array.

PROBLEM:

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.

Example 1:

Input: nums = [3,2,3]

Output: 3

Example 2:

Input: nums = [2,2,1,1,1,2,2]

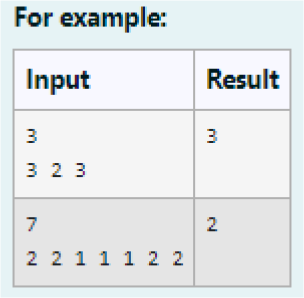
Output: 2

Constraints:

n == nums.length

1 <= n <= 5 * 104

-2^31 <= nums[i] <= 2^31 – 1



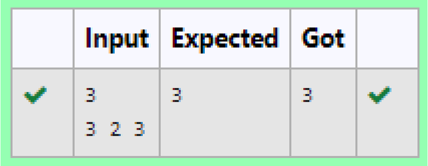
For example:

Input	Result
3 3 2 3	3
7 2 2 1 1 1 2 2	2

PROGRAM:

```
#include <stdio.h>

int x = 0;

int y(int arr[], int left, int right, int c) {

    if (left > right) {

        return 0;

    }

    int mid = left + (right - left) / 2;

    if (arr[mid] == c) {

        x++;

        y(arr, left, mid - 1, c);

        y(arr, mid + 1, right, c);

    } else if (arr[mid] < c) {

        y(arr, mid + 1, right, c);

    } else {

        y(arr, left, mid - 1, c);

    }

    return x;

}

int main() {

    int d;

    scanf("%d", &d);

    int arr[d];
    for (int i = 0; i < d; i++) {
        scanf("%d", &arr[i]);
    }
    int f = arr[0];
    if (y(arr, 0, d, f) > d / 2) {
        printf("%d", f);
    } else {
        for (int i = 0; i < d; i++) {
            if (arr[i] != f) {
                printf("%d", f);
                break;
            }
        }
    }
    return 0;
}
```

OUTPUT:



	Input	Expected	Got	
✓	3 3 2 3	3	3	✓

PROBLEM-3

AIM: 3-FINDING FLOOR VALUE

ALGORITHM:

1. Read the integer `n` (size of the array) and the array `arr[]` of size `n`.

2. Read the integer `x` (the target value for which the floor is to be found).

3. Define a recursive function `Floor()` to find the largest element less than or equal to `x` using binary search.

4. In the function, calculate the middle index `mid`.

5. If the element at `mid` is greater than `present` and less than `x`, update `present`.

6. Recursively search the left or right half of the array based on the comparison of the `mid` element with `x`.

7. Print the value of `present`, which is the floor value of `x`.

PROBLEM:

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.

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains Integer x – Value for x

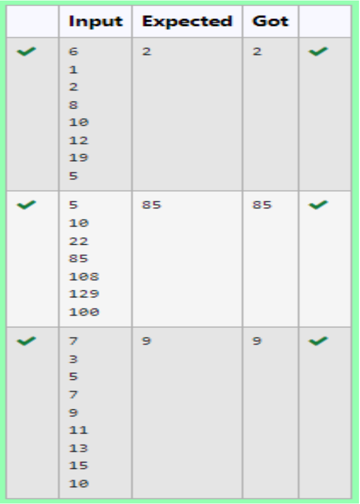
Output Format

First Line Contains Integer – Floor value for x

PROGRAM:

```
#include <stdio.h>

int Floor(int arr[], int left, int right, int present, int x) {

    int mid = left + (right - left) / 2;

    if ((arr[mid] > present) && (arr[mid] < x)) {
        present = arr[mid];
        return present;
    } else {
        return Floor(arr, left, mid, present, x);
        return Floor(arr, mid + 1, right, present, x);
    }
    return present;
}
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);
    printf("%d", Floor(arr, 0, n, arr[0], x));
    return 0;
}
```

OUTPUT:



	Input	Expected	Got	
✓	6 1 2 8 10 12 19 5	2	2	✓
✓	5 10 22 85 108 129 100	85	85	✓
✓	7 3 5 7 9 11 13 15 10	9	9	✓

PROBLEM-4:

AIM: TWO ELEMENTS SUM TO X

ALGORITHM:

1. Read integers `n` (size of array) and `s` (target sum).

2. Read the array `a[]` of size `n`.

3. Define a recursive function `SumS()` to find two elements in the array that sum to `s` using binary search.

4. In the function, calculate the middle index `mid`.

5. If the sum of `arr[mid]` and `arr[r]` equals `s`, print the two elements.

6. If the sum is greater than `s`, recursively search the left half of the array.

7. If the sum is less than `s`, recursively search the right half of the array. If no pair is found, print "No".

PROBLEM:

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

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains Integer x – Sum Value

Output Format

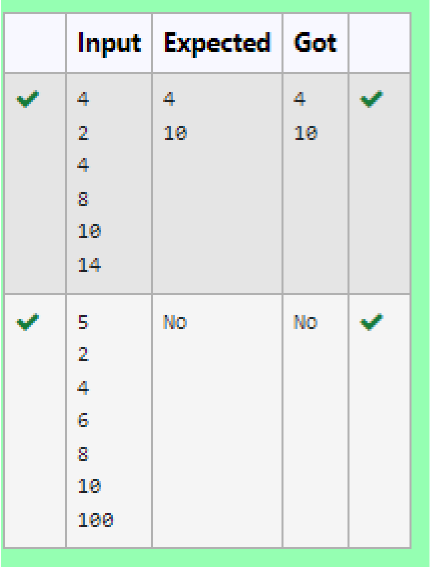
First Line Contains Integer – Element1

Second Line Contains Integer – Element2 (Element 1 and Elements 2 together sums to value “x”)

PROGRAM:

```
#include <stdio.h>

void SumS(int arr[],int x,int l,int r)

{
    if (l<r)
    {
        int mid=(l+r)/2;
        if (arr[r]+arr[mid]==x)
            printf("%d\n%d ",arr[mid],arr[r]);
        else if(arr[mid]+arr[r]>x)
            SumS(arr,x,mid,r-1);
        else if(arr[mid]+arr[r]<x)
            SumS(arr,x,l+1,mid);
    }
    else
    printf("No");
}
int main()
{
        int n,s;
        scanf("%d",&n);
        int a[n];
        for (int i=0;i<n;i++)
            scanf("%d",&a[i]);
        scanf("%d",&s);
        SumS(a,s,0,n-1);
        return 0;
}
```

OUTPUT:



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

PROBLEM-5:

AIM: IMPLEMENTATION OF QUICK SORT

ALGORITHM:

1. Read integer `n` (size of the array) and the array `a[]` of size `n`.

2. Define the `Partition()` function to perform partitioning of the array around a pivot element.

3. In the partitioning step, move elements smaller than the pivot to the left and larger elements to the right.

4. Define the `QuickSort()` function to recursively sort the array using the `Partition()` function.

5. In `QuickSort()`, call the `Partition()` function to get the pivot index, then recursively sort the left and right subarrays.

6. Call `QuickSort()` to sort the entire array.

7. Print the sorted array.

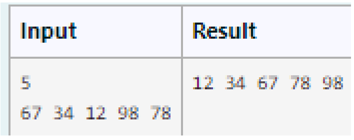
PROBLEM:

Write a Program to Implement the Quick Sort Algorithm

Input Format:

The first line contains the no of elements in the list-n
The next n lines contain the elements.

Output:
Sorted list of elements

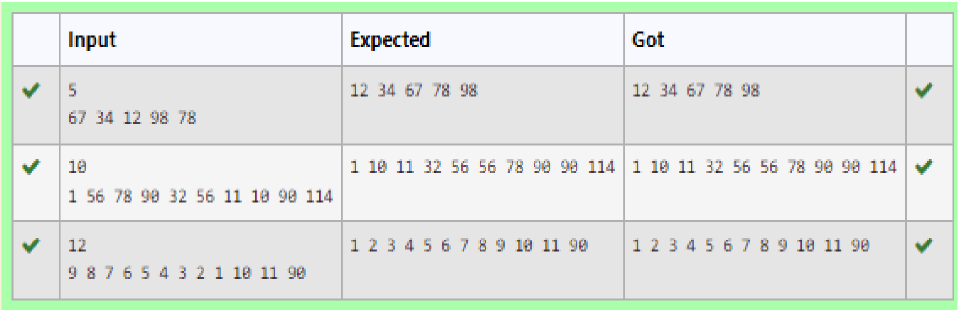


Input	Result
5 67 34 12 98 78	12 34 67 78 98

PROGRAM:

```
#include <stdio.h>
int Partition(int arr[],int l,int r)
{
    int pivot=arr[r];
    int temp;
    int j=l-1;
    for (int i=l;i<=r;i++)
    {
        if (pivot>arr[i])
        {
            j++;
            temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
        }
    }
    int t = arr[j+1];
    arr[j+1] = arr[r];
    arr[r] = t;
    return (j+1);
}
void QuickSort(int arr[],int l,int r)
{
    if (l<r){
        int p=Partition(arr,l,r);
        QuickSort(arr,l,p-1);
        QuickSort(arr,p+1,r);
    }
}
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]);
}
```

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	✓