DIVIDE AND CONQUER

1.

Problem Statement

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.

Algorithm:

1. Start
2. Input integer size
3. Input array arr[size]
4. Define function div(arr, low, high)
5. If arr[high] == 1, return 0
6. If arr[low] == 0, return high - low + 1
7. Calculate mid = (low + high) / 2
8. Recursively call div(arr, low, mid) and div(arr, mid + 1, high)
9. Return the sum of left and right results
10. store the result in count
11. Output value of count
12. End

Program:

```
#include<stdio.h>
int div(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=div(arr,low,mid);
   int right=div(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=div(arr,0,size-1);
   printf("%d\n",count);
}
```

	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 0 0	2	2	✔

Passed all tests! ✔

2.

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 * 10^4`
- `-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

Algorithm:

1. Start
2. Input integer size
3. Input array arr[size]
4. Define function divide(a, low, high, s)
5. If low == high, return a[low]
6. Calculate mid = (low + high) / 2
7. Recursively call divide(a, low, mid, s) and divide(a, mid + 1, high, s)
8. If left > (s / 2), return left, else return right
9. Call divide(arr, 0, size - 1, size) and store the result in majority
10. Output value of majority
11. End

Program:

```
#include<stdio.h>
int divide(int a[],int low,int high,int s)
{
   if(low==high)
```

```
    {
        return a[low];
    }
    int mid=(low+high)/2;
    int left=divide(a,low,mid,s);
    int right=divide(a,mid+1,high,s);
    if(left>(s/2))
        return left;
    else
        return right;
}

int main()
{
    int size;
    scanf("%d",&size);
    int arr[size];
    for(int i=0;i<size;i++)
    {
        scanf("%d",&arr[i]);
    }
    int l=0,h=size-1;
    int majority=divide(arr,l,h,size);
    printf("%d",majority);
}
```

	Input	Expected	Got	
✔	3 3 2 3	3	3	✔

Passed all tests! ✔

3.

Problem Statement:
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

Algorithm:

1. Start
2. Input integer size
3. Input array arr[size]
4. Input integer a
5. Calculate mid = (0 + (size - 1)) / 2
6. Call find(arr, 0, mid, a) and store the result in left
7. Call find(arr, mid + 1, size - 1, a) and store the result in right
8. If left > right, output left, else output right
9. Define function find(arr, low, high, x)
10. Initialize element = 0
11. For i = low to high:
12. Return element
13. End

Program:

```
#include<stdio.h>
int find(int arr[],int high,int low,int x);
int main(){
   int size;
   scanf("%d",&size);
   int arr[size];
   for(int i=0;i<size;i++){
      scanf("%d",&arr[i]);
   }
   int a;
   scanf("%d",&a);
   int mid=(0+(size-1)/2);
   int left=find(arr,0,mid,a);
   int right=find(arr,mid+1,size-1,a);
   if(left>right)
   {
      printf("%d",left);
```

```
    }
    else
    {
       printf("%d",right);
    }
}
    int find(int arr[],int low,int high,int x){
    int element=0;
    for(int i=0;i<high;i++){
       if(arr[i]<=x){
          if(arr[i]>element){
             element=arr[i];
          }
       }
    }
    return element;
}
```

	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	✔

Passed all tests! ✔

4.

Problem Statement:
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")

Algorithm:

14. Start
15. Input integer n
16. Input array arr[n]
17. Input integer x
18. Call find_pair(arr, 0, n-1, x)
19. Define function find_pair(arr, left, right, x)
20. While left < right:
21. Calculate current_sum = arr[left] + arr[right]
22. If current_sum == x, print arr[left] and arr[right]
23. If current_sum < x, increment left
24. Else, decrement right
25. If no pair found, print "No"
26. End

Program:

```
#include <stdio.h>

void find_pair(int arr[], int left, int right, int x) {
   while (left < right) {
      int current_sum = arr[left] + arr[right];

      if (current_sum == x) {
         printf("%d\n", arr[left]);
         printf("%d\n", arr[right]);
         return;
      } else if (current_sum < x) {
         left++;
      } else {
         right--;
      }
   }
```

```
    printf("No\n");
}

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

    find_pair(arr, 0, n - 1, x);

    return 0;
}
```

	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! ✔

5.

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

For example:

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

Algorithm:

1. Start
2. Input integer n
3. Input array a[n]
4. Define function part(a, l, h)
5. Set pivot = a[l], initialize i = l, j = h
6. While i < j:
7. Increment i until a[i] > pivot
8. Call Quicksort(a, 0, n-1)
9. Output sorted array a[]
10. End

Program:

```
#include <stdio.h>
int part(int a[],int l,int h){
int pivot=a[l];
int i=l,j=h,t;
while(i<j){
```

```
  while(a[i]<=pivot && i<h){
     i++;
  }
  while(a[j]>pivot && j>l){
     j--;
  }
  if(i<j){
     t=a[i];
     a[i]=a[j];
     a[j]=t;
  }
}
t=a[l];
a[l]=a[j];
a[j]=t;
return j;
}
void Quicksort(int a[],int l,int h){
if(l<h){
  int par=part(a,l,h);
  Quicksort(a,l,par-1);
  Quicksort(a,par+1,h);
}
}
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]);
}
}
```

	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! ✔