

DIVIDE AND CONQUER

QUESTION 4.A

AIM:

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 :

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize array a of size n

Step 4: For each index i from 0 to $n-1$, input $a[i]$

Step 5: Call the function `countz(a, 0, n - 1)` and store its result in `count`

Step 6: Print the value of `count`

Step 7: Stop

Function `countz(a[], l, r)`:

Step 1: If $l > r$, return 0

Step 2: Calculate `mid` as $l + (r - l) / 2$

Step 3: Initialize `count` to 0

Step 4: If `a[mid] == 0`, set `count = 1`

Step 5: Return `count + countz(a, l, mid - 1) + countz(a, mid + 1, r)`

PROGRAM :

```

#include <stdio.h>

int countz(int a[],int l,int r);

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

    return 0;
}

int countz(int a[],int l,int r)
{
    if (l>r)
    {
        return 0;
    }
    int mid=l+(r-l)/2;
    int count=0;
    if (a[mid]==0)
    {
        count=1;
    }
    return count + countz(a, l, mid - 1) + countz(a, mid + 1, r);
}

```

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 :

The above program is executed successfully .

QUESTION 4.B

AIM :

Given an array `nums` of size `n`, return *the majority element*.

The majority element is the element that appears more than $\lfloor n / 2 \rfloor$ 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`
- `-231 <= nums[i] <= 231 - 1`

ALGORITHM :

Step 1: Start

Step 2: Input the integer `n`

Step 3: Initialize array `a` of size `n`

Step 4: For each index `i` from 0 to `n-1`, input `a[i]`

Step 5: Call the function `majority(a, 0, n - 1)` and store its result in `majoele`

Step 6: If `majoele` is not -1, print `majoele`; otherwise, print "No Majority Element"

Step 7: Stop

Function `majority(a[], l, r)`:

Step 1: If `l == r`, return `a[l]`

Step 2: Calculate `mid` as $(l + r) / 2$

Step 3: Call `majority(a, l, mid)` and store its result in `leftmajo`

Step 4: Call `majority(a, mid + 1, r)` and store its result in `rightmajo`

Step 5: Initialize `lc` and `rc` to 0

Step 6: For each index `i` from `l` to `r`, if `a[i] == leftmajo`, increment `lc`; if `a[i] == rightmajo`, increment `rc`

Step 7: If `lc > (r - l + 1) / 2`, return `leftmajo`

Step 8: If `rc > (r - l + 1) / 2`, return `rightmajo`

Step 9: Return -1

PROGRAM :

```

#include <stdio.h>

int majority(int a[], int l, int r)
{
    if (l == r)
    {
        return a[l];
    }
    int mid = (l + r) / 2;
    int leftmajo = majority(a, l, mid);
    int rightmajo = majority(a, mid + 1, r);

    int lc = 0, rc = 0;
    for (int i = l; i <= r; i++)
    {
        if (a[i] == leftmajo) lc++;
        if (a[i] == rightmajo) rc++;
    }
    if (lc > (r - l + 1) / 2)
    {
        return leftmajo;
    }
    if (rc > (r - l + 1) / 2)
    {
        return rightmajo;
    }
    return -1;
}

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

    int majoele = majority(a, 0, n - 1);

    if (majoele != -1)
    {
        printf("%d\n",majoele);
    }
    else
    {
        printf("No Majority Element\n");
    }
}

```

OUTPUT :

	Input	Expected	Got	
✓	3 3 2 3	3	3	✓

Passed all tests! ✓

RESULT :

The above program is executed successfully.

QUESTION 4.C

AIM :

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 :

Step 1: Start
Step 2: Input the integer n
Step 3: Initialize array a of size n
Step 4: For each index i from 0 to $n-1$, input $a[i]$
Step 5: Input integer k
Step 6: Call `findfloor(a, 0, n - 1, k)`
Step 7: Stop

Function `findfloor(a[], l, r, key)`:

Step 1: If $a[r] \leq \text{key}$, print $a[r]$ and return
Step 2: If $l < r$, do Steps 3 and 4
Step 3: Calculate mid as $(l + r) / 2$
Step 4: Call `findfloor(a, mid + 1, r, key)`
Step 5: Call `findfloor(a, l, mid, key)`

PROGRAM :

```

#include<stdio.h>
int search(int[],int,int,int);
int search(int arr[],int x,int left,int right)
{
    int mid=left+(right-left)/2;
    if(arr[mid]<=x)
    {
        int max = arr[mid];
        for(int i=0;i<mid;i++){
            if(arr[i]>=max)
                max=arr[i];
        }
        return max;
    }
    else if(arr[mid]>x)
    {
        return search(arr,x,left,mid);
    }
    else
        return search(arr,x,mid+1,right);
}

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

```

OUTPUT:

	Input	Expected	Got	
✓	6	2	2	✓
	1			
	2			
	8			
	10			
	12			
	19			
	5			

RESULT:

The above program is executed successfully.

QUESTION 4.B

AIM :

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 :

Step 1: Start

Step 2: Input the integer `n`

Step 3: Initialize array `arr` of size `n`

Step 4: For each index `i` from 0 to `n-1`, input `arr[i]`

Step 5: Input integer `x`

Step 6: Call `findPair(arr, 0, n - 1, x)`

Step 7: Stop

Function `findPair(arr[], left, right, x)`:

Step 1: If `left >= right`, print "No" and return

Step 2: Calculate sum as `arr[left] + arr[right]`

Step 3: If `sum == x`, print `arr[left]` and `arr[right]`, and return

Step 4: If `sum < x`, call `findPair(arr, left + 1, right, x)`

Step 5: Otherwise, call `findPair(arr, left, right - 1, x)`

PROGRAM :


```

#include<stdio.h>
void twosum(int arr[],int left,int right,int x){
    if (left >= right){
        printf("No");
        return;
    }
    int sum=arr[left]+arr[right];
    if (sum==x){
        printf("%d\n",arr[left]);
        printf("%d\n",arr[right]);
    }
    else if(sum<x){
        twosum(arr,left+1,right,x);
    }
    else{
        twosum(arr,left,right-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);
    twosum(arr,0,n-1,x);
    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	✓

Passed all tests! ✓

RESULT:

The above program is executed successfully.

QUESTION 4.E

AIM:

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 :

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize array arr of size n

Step 4: For each index i from 0 to n-1, input arr[i]

Step 5: Call quickSort(arr, 0, n - 1)

Step 6: For each index i from 0 to n-1, print arr[i]

Step 7: Stop

Function quickSort(arr[], left, right):

Step 1: If left < right, do Steps 2 to 7

Step 2: Set pivot to (left + right) / 2

Step 3: Initialize i to left and j to right

Step 4: While i < j, do Steps 5.1 to 5.4

Step 5.1: While arr[pivot] >= arr[i], increment i

Step 5.2: While arr[pivot] < arr[j], decrement j

Step 5.3: If i <= j, swap arr[i] and arr[j]

Step 6: Swap arr[j] and arr[pivot]

Step 7: Call quickSort(arr, left + 1, right)

PROGRAM :

```

#include<stdio.h>
void quicksort(int arr[],int left,int right){
    if(left<right){
        int j=right;
        int i=left;
        int pivot=left;
        while(i<j){
            while(arr[i]<=arr[pivot]){
                i++;
            }
            while(arr[j]>arr[pivot]){
                j--;
            }
            if(i<j){
                int temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
            }
        }
        int temp=arr[j];
        arr[j]=arr[pivot];
        arr[pivot]=temp;
        quicksort(arr,left,j-1);
        quicksort(arr,j+1,right);
    }
}

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:

The above program is executed successfully .