Ex. No: 4 **Date: 03.09.24**

Register No.: 230701386 **Name: R YASHVINTHINI**

Divide and Conquer

4.a. Number of Zeros in a Given Array

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.

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. Initialize Count
2. Create a function findCount that takes an array a, and two integers l and r representing the left and right indices of the array segment to be processed.
3. check if the element at index l is 0. If true, increment count by the number of elements in the segment (r - l + 1).
4. If the element at index l is not 0 and l is less than r, calculate the middle index m as (l + r) / 2. Recursively call findCount for the left segment (l to m) and the right segment (m + 1 to r).
5. read the size of the array n from the user.
6. Read Array Elements
7. Call the findCount function with the array a, and the indices 0 and n - 1.
8. Print count.

Program:

```
#include<stdio.h>

int count=0;

void findCount(int a[],int l,int r){
```

```
    if(a[l]==0){
        count+=(r-l+1);
    }else{
        if(l<r){
            int m=(l+r)/2;
            findCount(a,l,m);
            findCount(a,m+1,r);
        }
    }
}
int main(){
    int n;
    scanf("%d",&n);
    int a[n];
    for(int i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    findCount(a,0,n-1);
    printf("%d",count);
}
```

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	✔

4.b. Majority Element

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.

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

Algorithm:

function Count(a[], l, r, key):

mid = l + (r - l) / 2

if a[mid] == key:

increment count

if l < mid:

Count(a, l, mid, key)

if mid + 1 < r:

Count(a, mid + 1, r, key)

return count

function main():

n = input() // Read the size of the array

arr[] = input() // Read the array of size n

k = arr[0] // Set the first element as the key

if Count(arr, 0, n, k) > n / 2:

print(k) // If the count of k exceeds n/2, print it as the majority element

else:

for i = 0 to n / 2 - 1:

```
            if arr[i] != k:
                print(k)  // If the first half contains an element different from k, print k
                break
```

Program:

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

}

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;
          }
    }

}
```

Output:

	Input	Expected	Got	
✔	3 3 2 3	3	3	✔

4.c. Finding Floor Value

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. 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:

```
function find(a[], l, r, key):
    if l <= r:
        if a[l] <= key:
            print a[l]  // Print elements less than or equal to the key
        // Recursive case: divide the array into two parts and continue searching
        mid = (l + r) / 2
        find(a, l, mid, key)  // Search left half
        find(a, mid + 1, r, key)  // Search right half

function main():
    n = input()  // Read the size of the array
    a[] = input()  // Read the array of size n
    k = input()  // Read the key
    find(a, 0, n-1, k)  // Call the recursive function to print values <= key
```

Program:

```
#include<stdio.h>
void find(int a[],int l,int r,int key)
{
```

```
    if(a[l]<=key)
    {
        printf("%d",a[l]);
    }
    else
    {
        if(l>r){
            int mid=(l+r)/2;
            find(a,l,mid+1,key);
            find(a,mid,r,key);
        }
    }
}
int main()
{
    int n;
    scanf("%d",&n);
    int a[n];

    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    int k;
    scanf("%d",&k);
    find(a,n-1,0,k);
```

}

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	✔

4.d. Two Elements Sum to X

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
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:

```
function sum(a[], l, r, s):
    if l < r:
        mid = (l + r) / 2  // Find the middle index
        if a[mid] + a[r] == s:  // Check if the sum of a[mid] and a[r] equals s
            s1 = a[mid]
            s2 = a[r]
            return 1  // Pair found
        else:
            return sum(a, l, r - 1, s)  // Continue searching in the subarray

function main():
    n = input()  // Read the size of the array
    a[] = input()  // Read the array of size n
    x = input()  // Read the target sum x
```

```
result = sum(a, 0, n - 1, x)  // Call the sum function
if result == 0:
    print("No")  // If no pair is found, print "No"
else:
    print(s1)
    print(s2)  // Print the two numbers found that add up to x
```

Program:

```
#include<stdio.h>
int s1=0,s2=0;
int sum(int a[],int l,int r,int s)
{
    if(l<r)
    {
        int mid=(l+r)/2;
        if(a[mid]+a[r]==s)
        {
            s1=a[mid];
            s2=a[r];
            return 1;
        }
        sum(a,l,r-1,s);

    }
    return 0;
```

```
}
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 y=sum(a,0,n-1,x);
    if (y==0)
    printf("%s","No");
    else
    {
        printf("%d\n%d",s1,s2);
    }

}
```

Output:

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

4.e. Implementation of Quick Sort

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 **Algorithm:**

```
function quickSort(a[], l, r):
    if l < r:
        pivotIndex = partition(a, l, r)  // Partition the array and get the pivot index
        quickSort(a, l, pivotIndex - 1)  // Recursively sort the left subarray
        quickSort(a, pivotIndex + 1, r)  // Recursively sort the right subarray

function partition(a[], l, r):
    pivot = a[r]  // Select the pivot (using the last element)
```

```
    i = l - 1  // Pointer for the smaller element
    for j = l to r - 1:  // Iterate through the array
        if a[j] <= pivot:  // If current element is smaller than or equal to the pivot
            i = i + 1
            swap(a[i], a[j])  // Swap elements
    swap(a[i + 1], a[r])  // Move the pivot to the correct position
    return i + 1  // Return the pivot index

function main():
    n = input()  // Read the size of the array
    a[] = input()  // Read the array of size n
    quickSort(a, 0, n - 1)  // Call the quickSort function to sort the array
    print(a[])  // Print the sorted array
```

Program:

```
 #include<stdio.h>
void quick(int a[],int l,int r)
{
    if(l<r)
    {

        int p=(l+r)/2;
        int i=l;
        int j=r;

        while(i<j)
        {
```

```
        while(a[p]>=a[i] )
        {
            i++;
        }

        while(a[p]<a[j] )
        {
            j--;
        }
        if(i<=j)
        {
            int temp=a[i];
            a[i]=a[j];
            a[j]=temp;
        }int temp=a[j];
    a[j]=a[p];
    a[p]=temp;
    quick(a,l+1,r);



  }
}
```

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