Ex:4 **Date**:29.08.24

Name: Veronica Regina Paul **Register number**:230701377

Divide and Conquer

Number of Zeros in a Given Array

AIM

To find the number of zeros in a sorted array of 1s followed by 0s using the Divide and Conquer approach.

ALGORITHM

1. Divide the Array
   - Define a recursive function count(a[], l, r) to divide the array into two halves.
   - If l > r, return 0 (base case for an empty array).
2. Check Boundaries
   - If both a[l] and a[r] are 1, return 0 since there are no zeros in the range.
   - If both a[l] and a[r] are 0, return the count of elements in the range (r - l + 1).
3. Recursive Division
   - Find the midpoint m = (l + r) / 2.
   - Recursively count zeros in the left half (count(a, l, m)) and in the right half (count(a, m + 1, r)).
4. Combine Results
   - Return the sum of zeros from the left and right halves.
5. Output
   - The result of count(arr, 0, n - 1) gives the total count of zeros.

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 count(int a[], int l, int r) {
   if (l > r){
     return 0;
   }
   if (a[l] == 1 && a[r] == 1) {
      return 0;
   }
   if (a[l] == 0 && a[r] == 0){
     return r - l + 1;
   }
   int m = (l + r) / 2;
   int left = count(a, l, m);
   int right = count(a, m + 1, r);
   return left + right;
}
int main() {
   int n;
   scanf("%d", &n);
   int arr[n];
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }
   int zeroCount = count(arr, 0, n - 1);
   printf("%d\n", zeroCount);
   return 0;
```

}

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	

2-Majority Element

AIM

To find the majority element in an array using a Divide and Conquer approach. The majority element is defined as the element that appears more than ⌊n / 2⌋ times in the array.

ALGORITHM

1. Initialize Variables
   - cnt = 0: A global counter to count occurrences of a candidate element.
   - arr[]: Array of integers with size n.
2. Recursive Count Function
   - Define a recursive function Cnt(a[], l, r, key):
     - Find the midpoint m = l + (r - l) / 2.
     - If a[m] is equal to the key, increment the cnt value.
     - If not, recursively count the occurrences of key in the left half Cnt(a, l, m, key) and the right half Cnt(a, m+1, r, key).
3. Main Function
   - Take the input for the array size n.
   - Input the array arr[].
   - Select the first element arr[0] as the candidate majority element k.
   - Call Cnt(arr, 0, n, k) to count the occurrences of k in the array.
   - If the count of k is greater than n / 2, print k as the majority element.
   - If the count is not sufficient, iterate through the array to find the correct majority element.
4. Output
   - Print the majority element if it exists.

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`
- $1 <= n <= 5 * 10^4$
- $-2^{31} <= nums[i] <= 2^{31} - 1$

For example:

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

PROGRAM

```
#include <stdio.h>

int cnt=0;

int Cnt(int a[], int l, int r, int key)

{

   int m = l + (r - l) / 2;

   if (a[m] == key)

      cnt++;

   else

   {

      Cnt(a, l, m, key);

      Cnt(a, m + 1, r, key);

   }

   return cnt;

}
```

```
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 (Cnt(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	✔

3-Finding Floor Value

AIM

To find the **floor value** of a given number x in a sorted array using the divide and conquer approach. The floor value of x is the largest element in the array that is smaller than or equal to x.

ALGORITHM

1. Input the Array:
   - First, read the size of the array n.
   - Then, read the n integers into an array arr[].
   - Read the value x, for which we need to find the floor value.
2. Recursive Function (Floor):
   - Define a recursive function Floor(a[], l, r, c, ip) where:
     - a[]: Array containing the elements.
     - l: Left index of the array.
     - r: Right index of the array.
     - c: Current floor value (initialized to a[0]).
     - ip: The value x for which we are finding the floor.
   - Find the middle element mid of the array.
   - If the element at mid is smaller than or equal to x, update c to a[mid] (since it is a valid floor candidate) and recursively search the left half for a larger floor value.
   - If the element at mid is larger than x, recursively search the left half of the array.
   - If the element at mid is smaller than or equal to x, continue searching the right half of the array.
   - The base case will return the current value of c, which is the floor.
3. Return the Floor Value:
   - After completing the recursion, return the floor value.

PROBLEM

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

PROGRAM

```
#include <stdio.h>
int Floor(int a[],int l,int r,int c,int ip)
{
   int mid=l+(r-l)/2;
   if ((a[mid]>c) && (a[mid]<ip))
   {
      c=a[mid];
      return c;
   }
   else
   {
      return Floor(a,l,mid,c,ip);
      return Floor(a,mid+1,r,c,ip);
   }
   return c;
}
int main()
{
   int n,val;
   scanf("%d",&n);
   int arr[n];
   for (int i=0;i<n;i++)
      scanf("%d",&arr[i]);
   scanf("%d",&val);
```

```
    printf("%d",Floor(arr,0,n,arr[0],val));
    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	✔

4-Two Elements sum to x

AIM

The aim of this program is to find two elements in a sorted array whose sum equals a given value x using the divide and conquer strategy. If such a pair is found, the program prints the two elements; otherwise, it prints "No".

ALGORITHM

1. Input the Array and Target Value:
   - Read the integer n which represents the size of the array.
   - Read the n integers into an array arr[].
   - Read the integer x, the target sum for which we need to find two elements whose sum equals x.
2. Recursive Divide and Conquer Function (SumS):
   - Initialize two pointers, l (low) at the beginning (0) of the array and h (high) at the last index (n-1).
   - Check if the sum of the elements at indices l and h equals x:
     - If arr[l] + arr[h] == x, print arr[l] and arr[h] as the pair.
     - If the sum is greater than x, move the high pointer h one step left (h - 1) to reduce the sum.
     - If the sum is less than x, move the low pointer l one step right (l + 1) to increase the sum.
   - Continue this process until the pointers meet or cross each other.
   - If no valid pair is found (when l crosses h), print "No".
3. End Program:
   - The program stops after printing the result of the pair or "No".

ALGORITHM

PROBLEM

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”)

PROGRAM

```
#include <stdio.h>
void SumS(int arr[],int x,int l,int h)
{
  if (l<h)
  {
    int mid=(l+h)/2;
    if (arr[h]+arr[mid]==x)
      printf("%d\n%d ",arr[mid],arr[h]);
    else if(arr[mid]+arr[h]>x)
      SumS(arr,x,mid,h-1);
    else if(arr[mid]+arr[h]<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	✓

5-Implementation of Quick Sort

AIM

The aim of this program is to implement the Quick Sort algorithm to sort a given list of integers in ascending order.

ALGORITHM

1. Input the List:
   - Read the integer n, representing the number of elements in the array.
   - Read the n integers into an array arr[].
2. Partition Function:
   - Choose the last element as the pivot.
   - Initialize two pointers: i and j.
     - i is initially set to l - 1 (just before the leftmost element).
     - j starts from l and iterates through the array.
   - Compare each element with the pivot. If an element is smaller than the pivot, increment j and swap the elements at indices i and j.
   - After iterating through the array, swap the pivot with the element at index j + 1. This places the pivot in its correct sorted position.
   - Return the index of the pivot.
3. Quick Sort Function:
   - Recursively apply the Quick Sort on the left and right subarrays:
     - Left subarray: From the starting index l to the pivot index p - 1.
     - Right subarray: From the pivot index p + 1 to the ending index r.
   - Base case: When l >= r, no further sorting is needed.
4. Output the Sorted List:
   - After the Quick Sort is complete, print the sorted array.

Pseudocode

1. Input:
   - Read n and the array arr[].

2. Function Partition(arr[], l, r):
    - Select pivot arr[r].
    - Initialize j = l - 1.
    - Iterate i from l to r:
        - If arr[i] < pivot, swap arr[i] and arr[j + 1], and increment j.
    - Swap arr[j + 1] and arr[r].
    - Return j + 1.
3. Function QuickSort(arr[], l, r):
    - If l < r, call Partition(arr[], l, r) to get the pivot index p.
    - Recursively call QuickSort(arr[], l, p - 1) and QuickSort(arr[], p + 1, r).
4. Output:
    - Print the sorted array after Quick Sort.

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

For example:

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

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	✔