Design analysis and algorithm

Lab 7

Divide And Conquer _binary and max_vote_navie

Name: Naomi George

Slot: L29

Reg No: 19bce7572

Course code: CSE3004

Max_Votes:

```python
n = int(input())
seq = [int(i) for i in input().split()]


def divide_func(seq, l, r):
    if l+1==r:
        return seq[l]
    elif l+2==r:
        return seq[l]
    m = (l+r)//2
    left = divide_func(seq, l, m)
    right = divide_func(seq, m, r)

    c1, c2 = 0, 0
    for i in seq[l:r]:
        if i == left:
            c1+=1
        elif i == right:
            c2+=1
    if c1>(r-l)//2 and left != -1:
        return left
    elif c2>(r-l)//2 and right != -1:
        return right
    else:
        return -1

print(int(divide_func(seq, 0, n) != -1))
```

Output:

```
5
1 2 3 4 5
0
```

```
5
2 3 9 2 2
1
```

Analysis:

Divide & Conquer - max voting :

for maximum voting,
If we divide the input array into 2 halves & recursively find the majority element of the left & right halves, & recursively ~~find the majority element of~~ we can determine the global majority element in linear time.

We define a helper function which takes left & right end of the current array as a parameter. The initial value of $l = 0$ & $r = n-1$.

Base Case : This is a trivial case of single element array when both the left & right ends are equal. we return this single element as the majority element i.e if $(l == r)$, return $X[l]$.

Divide : Divide the array into 2 equal halves by calculating the mid-value, i.e, $mid = (r - l)/2 + l$

Conquer : recursively calculate the majority element of the left & right halves & store them.

Combine : If the majority element of left & right halves are equal, then it must be the global maj. element & we return the value or else one of them must be equal to global majority element.

Now for the complexity analysis :

the problem is of size n by recursively solving the 2 smaller sub problem of size $n/2$ & combine the solution of these 2 smaller problems by counting the frequency by performing 2 linear scan of size n.

Complexity of divide part $= O(1)$

" " conquer part $= 2T(n/2)$

" " combine part $= O(n) + O(n)$

Overall time complexity $T(n) = O(1) + 2T(n/2) + O(n)$

$= 2T(n/2) + O(n)$

& $T(n) = O(n \log n)$ for worst case

Binary Search:

```python
seq = [int(i) for i in input().split()]
search_seq = [int(i) for i in input().split()]
n = seq[0]
seq = seq[1:]

def binary_search(seq, elt, r):
    l = 0
    while l<=r:
        m = (l+r)//2
        if elt > seq[m]:
            l = m + 1
        elif elt < seq[m]:
            r = m - 1
        else:
            return m
    return -1

soln = list()
for i in search_seq[1:]:
    ans = binary_search(seq, i, n-1)
    soln.append(ans)
print(' '.join([str(i) for i in soln]))
```

Output:

```
5 1 5 8 12 13
5 8 1 23 1 11
2 0 -1 0 -1
```

Analysis:

Divide & Conquer - Binary Search:

For a collection of elements 'n', we need 'm' number of splits to get our target.

Thus a double in our collection 2n results in m+1 splits.

A mathematically function that represents such change in logarithms,

To see this clearly, if you have a collection of elements which are to the power of two that is 1, 2, 4, 8, 16. The result of using the logarithm function to base ~~to~~ two (2) is equal to m. Thus m+1 is the maximum no. of splits we would require to find our target.

To ~~et~~ calculate the time complexity of binary search we can apply our knowledge seen. Therefore, in the worst case binary search worst time complexity is

$O(\log n)$