Application of Divide and Conquer

Binary Search

Binary Search :

input : An array of n elements and what element "x" we want to search in an array

output : Position of an element x if it is found and if it is not present in the array then our function will return -1

Implementation of Binary Search :

0	**1**	**2**	**3**	**4**
2	**4**	**6**	**8**	**10**

i = 0

j = 4

x => That we want to search in an array = 8

mid = (0 + 4)/2 = 2

a[mid] = a[2] = 6 == 8 -> false

a[mid] = a[2] = 6 < 8 -> true

BinarySearch(a,mid+1,j,x);

i = 3

j = 4

mid = (3+4)/2 = 7/2 = 4

a[mid] = a[4] = 10 == 8 -> false

a[mid] = a[4] = 10 > 8

BinarySearch(a,i,mid-1,x);

i = 3

j = 3

i == j (3 == 3)

a[i] = a[3] = 8 == 8 - true

Return -> 3

```
BinarySearch(a,i,j,x){

if(i == j){                                       // small problem

        if(a[i] == x){

                return i;                         O(1)

        }

return -1;

}

// Big problem -> we apply Divide and Conquer Strategy

while(i < j){

int mid = (i + j)/2;                              O(1)

if(a[mid] == x){

        return mid;                               O(1)

}

if(a[mid] < x){

BinarySearch(a,mid+1,j,x);                        T(n/2)

}

else if(a[mid] > x){

BinarySearch(a,i,mid-1,x);                        T(n/2)

}
```

```
}
return -1;
}
```

Recurrence Relation :

T(n) = T(n/2) + c -> Binary Search Algorithm

a = 1

b = 2

n^(log_b a) = n^(log_2 1) = n^0 = 1

f(n) = c

Which one is greater?

Both are equal i.e. constant

Overall time complexity is : O(f(n) logn) => O(c logn) => O(logn)

Discussion about Best case, worst case and average case time complexity

Best case : O(1)

Worst case : O(logn)

Average case : O(logn)