Designing Of Algorithms

Divide And Conquer

Introduction

Strategy of Divide and Conquer :

- **Divide the big problem into some sub-problems.**
- **Solve the sub-problems (conquer) using Recursion until that particular sub-problem is solved.**
- **Combine the sub-problem solution so that we will be able to get the final solution of the problem.**

Main Point : If the problem is big, divide that problem otherwise not.

Abstract Algorithm of Divide and Conquer :

i -> starting element of an array

j -> ending element of an array

```
def DAC(a,i,j):
    if(small(a,i,j)):
        return (solution(a,i,j))          # O(1)
    else:
        m = Divide(a,i,j)                 #f1(n)
        b = DAC(a,i,m)                    #T(n/2)
        c = DAC(a,m+1,j)                  #T(n/2)
        return (combine(b,c))             #f2(n)
```

DAC - Finding of Time Complexity :

$T(n) = O(1)$; if n is small

$f1(n) + T(n/2) + T(n/2) + f2(n)$; if n is large

So, overall time complexity is

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

$f(n) \Rightarrow$ Divide + Combine

This is known as the Recurrence Relation.

So, for different problems we have different Recurrence

Relations. for example :

In QuickSort, Recurrence Relation is

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

In Strassen's Matrix Multiplication,

$T(n) = 8T(n/2) + n^2$

Here, 8 is the number of subproblems

$T(n/2)$ represents size of subproblem

n^2 is the Divide + Combine function

Applications of Divide and Conquer :

There are so many applications of Divide and Conquer for example :

- **Finding of Power of an Element**
- **Binary Search**
- **Merge Sort**

- **Quick Sort**
- **Selection Procedure**
- **Finding of inversions**
- **Finding of Maxima and Minima in the given array of elements**
- **Strassen's Matrix Multiplication and so on**