SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

19ITB201 – DESIGN AND ANALYSIS OF ALGORITHMS

II YEAR IV SEM

UNIT-II-BRUTE FORCE AND DIVIDE AND CONQUER

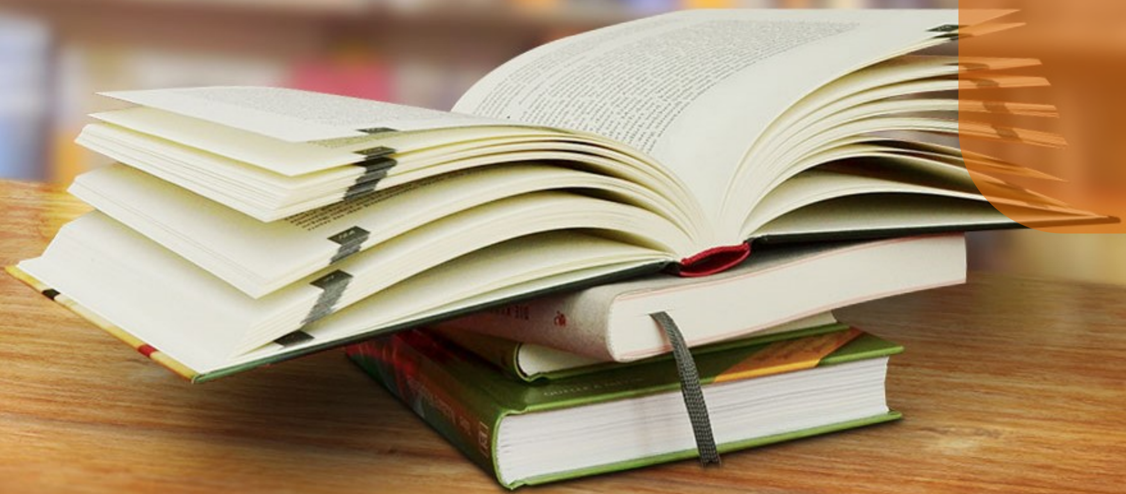
TOPIC: Divide and Conquer –Merge Sort

Prepared by
T.Shanmugapriya,AP/IT

DIVIDE AND CONQUER- MERGE SORT

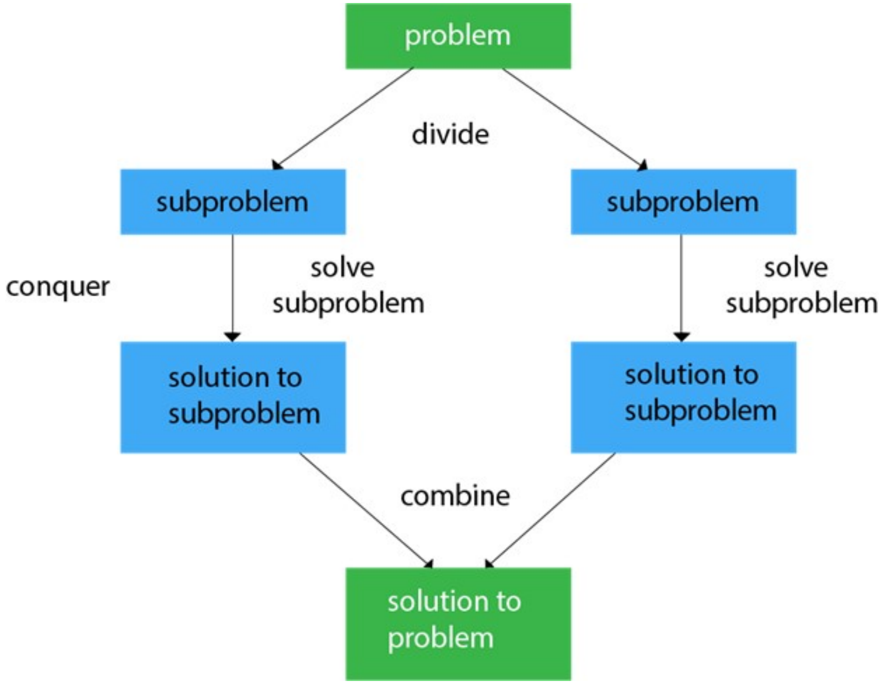
Subject :Design and Analysis of Algorithm

Unit :V



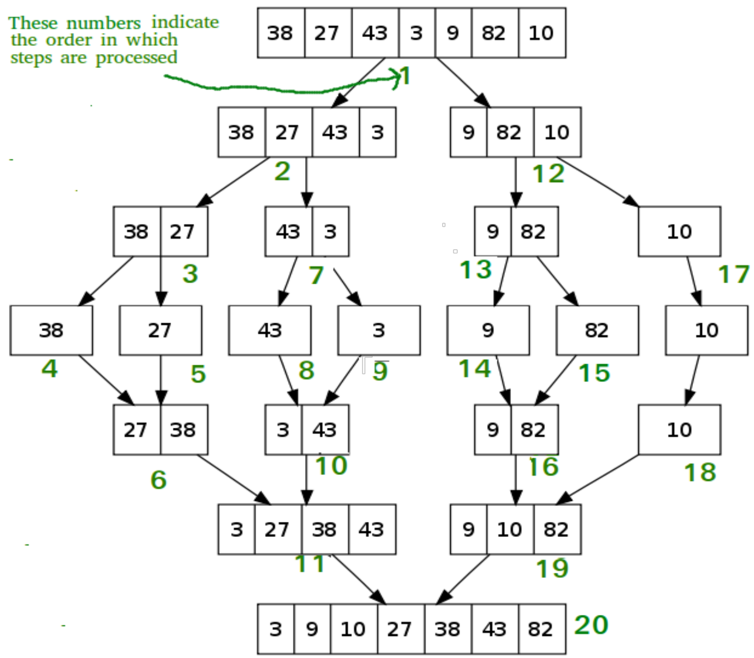
Identify the problem

Problem



problem
divide
subproblem
subproblem
conquer
solve
subproblem
solve
subproblem
solution to
subproblem
solution to
subproblem
combine
solution to
problem

Problem Example

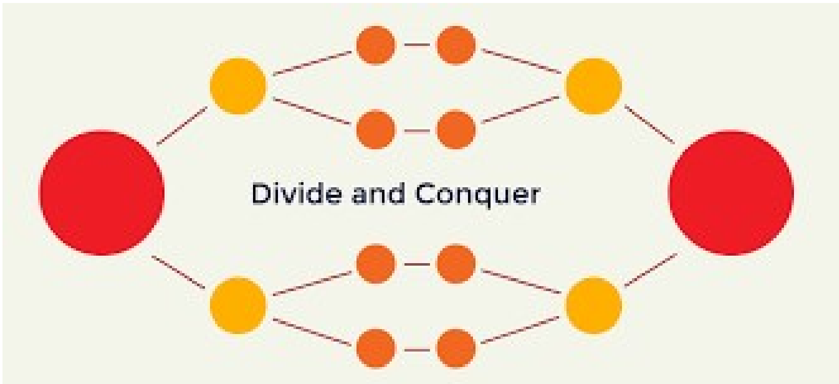


General Method

Divide and conquer algorithm consists of two parts:

Divide :Divide the problem into a number of sub problems. The sub proble ms are solved recursively.

Conquer :The solution to the original problem is then formed from the sol utions to the sub problems (patching together the answers).



Control Abstraction of Divide and Conquer

DANDC (P)

{

if SMALL (P) then return S (p); else

{

divide p into smaller instances p_1, p_2, …. P_k, $k\ ^{3}\ 1$; apply DANDC to each of these sub pro blems;

return (COMBINE (DANDC (p_1) , DANDC (p_2),…., DANDC (p_k));

}

}

If the sizes of the two sub problems are approximately equal then the computing time of DANDC is:

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ 2\,T(n/2) + f(n) & \text{otherwise} \end{cases}$$

Where, T (n) is the time for DANDC on ‘n’ inputs

g (n) is the time to complete the answer directly for small inputs and f (n) is the time for Divide and Combine

Applications

Sorting

Searching

Min max

Large Interger multiplication

$$^{+}3 \times {}^{-}8 \times {}^{-}2 = \left({}^{+}3 \times {}^{-}8\right) \times {}^{-}2$$
$$= \left({}^{-}24\right) \times {}^{-}2$$
$$= {}^{+}48$$

Merge sort

- Merge sort algorithm is a classic example of divide and conquer. To sort an array, recursively, sort its left and right halves separately and then merge them.

- The time complexity of merge mort in the *best case, worst case* an d *average case* is O(n log n) and the number of comparisons used i s nearly optimal.

Algorithm

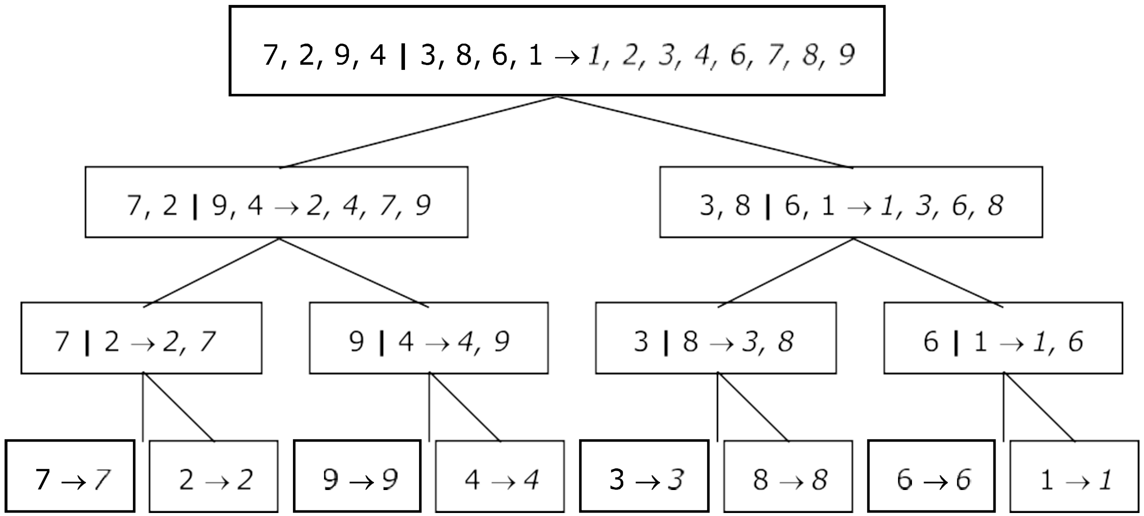
Algorithm MERGESORT (low, high)

```
// a (low : high) is a global array to be sorted.
{

i ptr

if (low < high)
{
mid := (low  + high)/2      //finds where to split the set
MERGESORT(low,  mid)//sort one subset
MERGESORT(mid+1, high) //sort the other subset
MERGE(low, mid, high)  // combine the results
}}
```

```
Algorithm MERGE (low, mid, high)
// a (low : high) is a global array containing two sorted subsets
// in a (low : mid) and in a (mid + 1 : high).
// The objective is to merge these sorted sets into single sorted
// set residing in a (low : high). An auxiliary array B is used.
{
        h :=low; i := low; j:= mid + 1;
        while ((h ≤ mid) and (J ≤ high)) do
        {
                if (a[h] ≤ a[j]) then
                {
                        b[i] := a[h]; h := h + 1;
                }
                else
                {
                        b[i] :=a[j]; j := j + 1;
                }

                    i := i + 1;
            }
            if (h > mid) then
                    for k := j to high do
                    {
                            b[i] := a[k]; i := i + 1;
                    }
            else
                    for k := h to mid do
                    {
                            b[i] := a[K]; i := i + l;
                    }
            for k := low to high do
                    a[k] := b[k];
    }
```

Example

For example let us select the following 8 entries 7, 2, 9, 4, 3, 8, 6, 1 to illustrate merge sort algorithm:



Analysis of Merge Sort

We will assume that 'n' is a power of 2, so that we always split into even halves, so we solve for the case $n = 2^k$.

For n = 1, the time to merge sort is constant, which we will be denote by 1. Otherwise, the time to merge sort 'n' numbers is equal to the time to do two recursive merge sorts of size n/2, plus the time to merge, which is linear. The equation says this exactly:

$T(1) = 1$

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

This is a standard recurrence relation, which can be solved several ways. We will solve by substituting recurrence relation continually on the right–hand side.

We have, $T(n) = 2T(n/2) + n$

Since we can substitute n/2 into this main equation

$$2\,T(n/2) = 2\,(2\,(T(n/4)) + n/2)$$
$$= 4\,T(n/4) + n$$

We have,

$$T(n/2) = 2\,T(n/4) + n$$
$$T(n) = 4\,T(n/4) + 2n$$

Again, by substituting n/4 into the main equation, we see that

$$4T\,(n/4) = 4\,(2T(n/8)) + n/4$$
$$= 8\,T(n/8) + n$$

So we have,

$$T(n/4) = 2\,T(n/8) + n$$
$$T(n) = 8\,T(n/8) + 3n$$

Continuing in this manner, we obtain:

$$T(n) \quad = \quad 2^k\, T(n/2^k) + K.\, n$$

As $n = 2^k$, $K = \log_2 n$, substituting this in the above equation

$$T(n) = 2^{\log_2 n} \; T\left(\frac{2^k}{2}\right) - \; \log_2 n \,.\, n$$

$$= n\, T(1) + n \log n$$
$$= n \log n + n$$

Representing this in O notation:

$T(n) =$ **O(n log n)**


Thank you!