UNIT 2 : DIVIDE AND CONQUER APPROACH

PAGE NO.:
DATE : / /

* Divide and Conquer Approach :-

• General method :-

- Divide and conquer is an algorithmic pattern.
- In algorithmic methods, the design is to take a dispute on a huge input, break the input into minor pieces, decide the problem on each of the small pieces and then merge the piecewise solution into a global solution.
- This mechanism of solving problem is called as the divide and conquer strategy.

* Divide and conquer algorithm consists of a dispute using following three steps :-

1] Divide :-

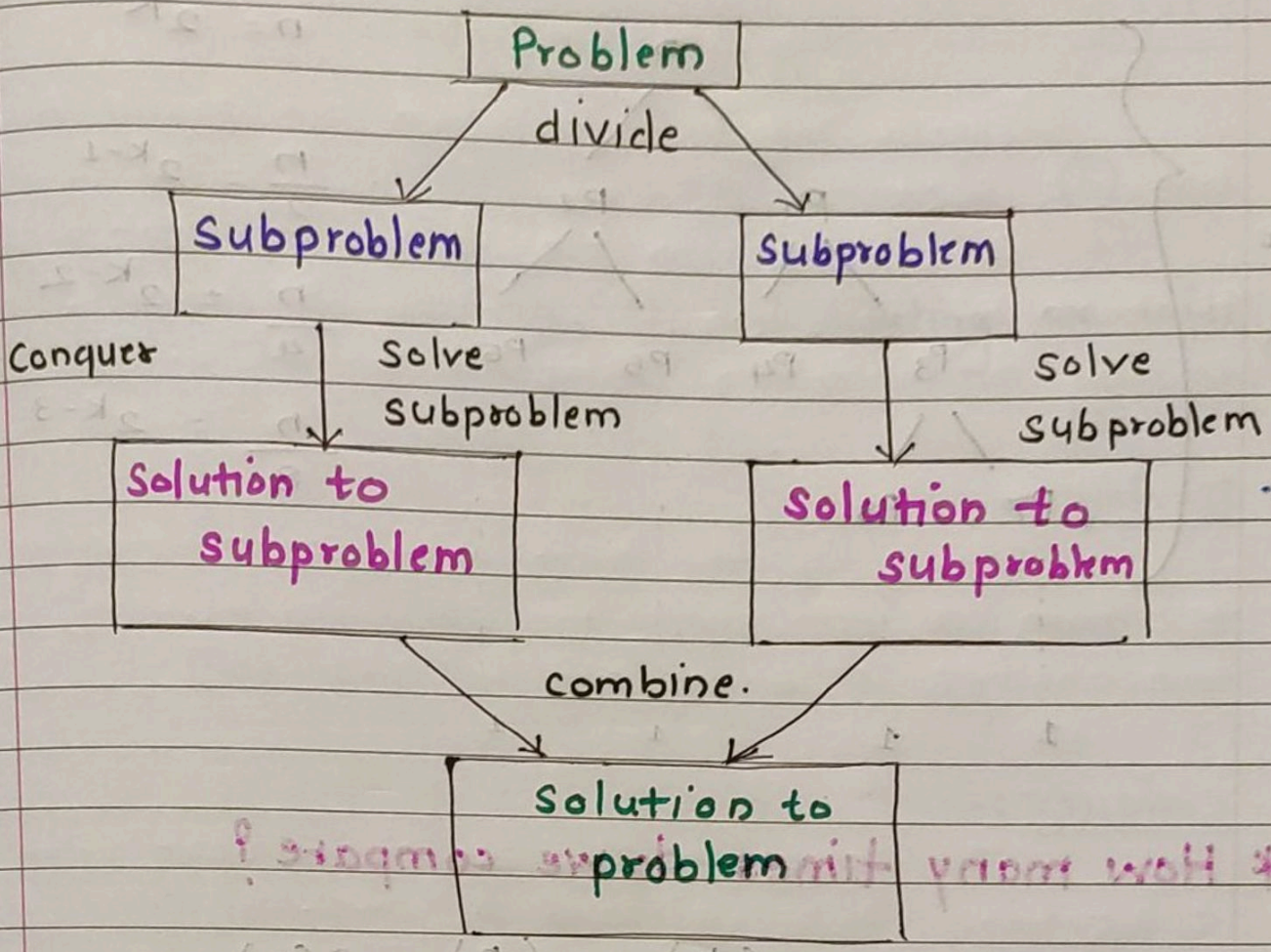
Divide the original problem into a set of subproblems.

2] Conquer :-

Solve every problem individually, recursively.

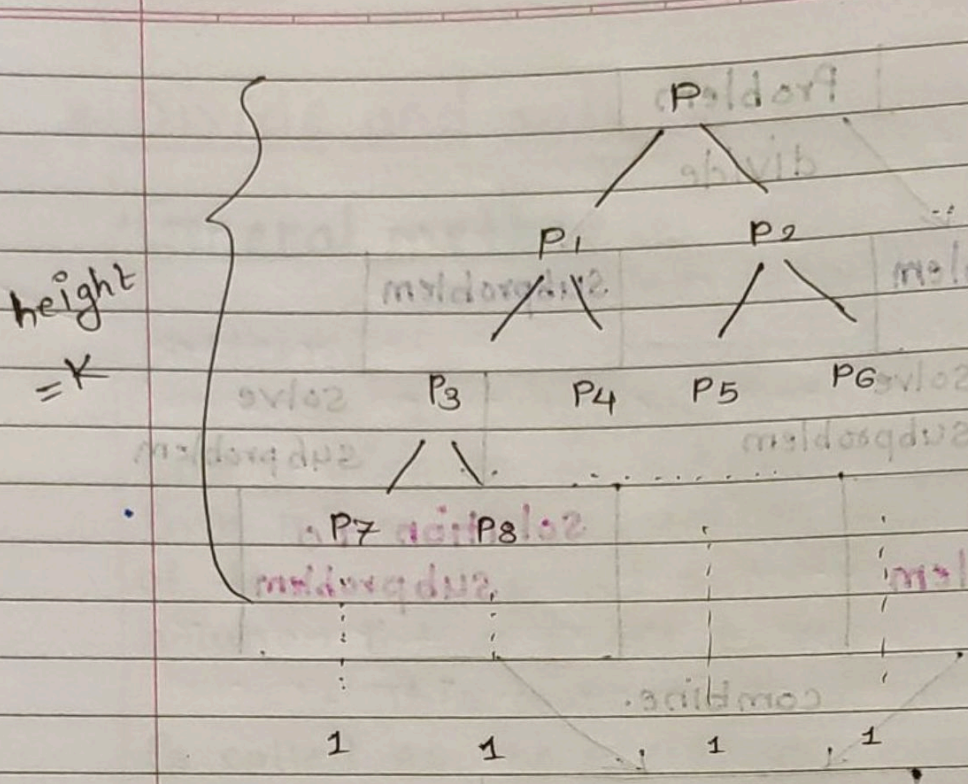
3] Combine :-

Put together the solutions of the subproblems to get the solution to the whole problem.



* **Algorithm :-** (Given a problem P of size $n = 2^k$)

```
1. Algorithm DAandC(P) {
2. If n is small, solve
3. else {
4. divide P into sub-problems P1, P2 .... Pk
5. Apply DAandC to each of these sub-problems.
6. combine all the solutions.
7. }
8. }
```



$n = 2^k$

$\frac{n}{2} = 2^{k-1}$

$\frac{n}{4} = 2^{k-2}$

$\frac{n}{8} = 2^{k-3}$

* How many times do we compare?

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + f(n)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + f(n) \quad \rightarrow (1)$$

So we solve this equation by using substitution method.

Replace $n \rightarrow n/2$ in eqn (1)

∴ Equation (1) becomes

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n/2}{2}\right) + \frac{n}{2}$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2} \quad — \quad (2)$$

∴ Put value of $T\left(\frac{n}{2}\right)$ i.e. eqn (2) in eqn (1)

$$\therefore T(n) = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

PAGE NO.:
DATE: / /

$$= 4T\left(\frac{n}{4}\right) + n + n$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 2n \quad \text{---} \quad (3)$$

Similary if we go on substituting we will get

$$T(n) = 8T\left(\frac{n}{8}\right) + 3n$$

$$\vdots$$

$$T(n) = n \cdot T\left(\frac{n}{n}\right) + \text{---} \; n$$

$$= n \cdot T(1) + K \cdot n$$

$$\boxed{\therefore T(n) = K \cdot n}$$

Therefore putting value of K we get

We have $n = 2^K$

Therefore $K = \log_2 n$

$$\therefore T(n) = \log n \cdot n$$

$$\therefore T(n) = O(n \log n)$$

PAGE NO.:
DATE: / /

* Merge sort using divide & conquer approach:-

- Suppose we had to sort an array A using merge sort.
- A subproblem would be to sort a sub section of this array starting at index p and ending at index r, denoted as A[p..r]

- **Divide :-**

If q is the half-way point between p and r, then we can split the subarray A[p..r] into two arrays A[p..q] and A[q+1..r]

- **Conquer :-**

- In conquer step we try to sort both the subarrays A[p..q] and A[q+1..r].
- If we haven't yet reached the base case, we again divide both these subarrays and try to sort them.

- **Combine :-**

- When the conquer step reaches the base step and we get two sorted subarrays A[p..q] & A[q+1..r] for array A[p..r], we combine the results by creating a sorted array A[p..r] from two sorted subarrays A[p..q] and A[q+1..r].

* How the merge function works?

- Our task is to merge two subarrays A[p..q] and A[q+1..r] to create a sorted array A[p..r] So the inputs to the function are A, p, q and r.

PAGE NO.:
DATE: / /

- The merge function works as follows:-

(1) Create copies of the subarrays $L \rightarrow A[p..q]$ and $M \rightarrow A[q+1..r]$

(2) Create three pointers i, j and k

i) i maintains current index of L, starting at 1.

ii) j maintains current index of M, starting at 1

iii) K maintains the current index of $A[p..q]$ starting at p.

(3) Until we reach the end of either L or M pick the larger among the element L and M and place them in the correct position at $A[p..q]$

(4) When we run out of elements in either L or M, pick up the remaining element and put in $A[p..q]$.