4/07/22

* Divide and Conquer Algorithm:

It works recursively. breaking down a given problem P. into two or more sub-problems (P1, P2 ... etc) of the same or related type, until these becomes simple enough to be solved directly. Then solutions to these sub problems (S1, S2, ... etc). are combined to get a solution of orgnal problem P.

A typical divide and conquer algorithm solves a given problem P using following steps.

Step 1: Divide:

- Breaks (or) divides the given problem P. into sub problem $P_1, P_2, \ldots$ etc of same type.

Step 2: Conquer:

- Recursively solve the sub problem $P_1, P_2, \ldots$ etc

Step 3: Combine:

- Combines the solutions of sub-problems S1, S2. etc to get solution of original problem P.

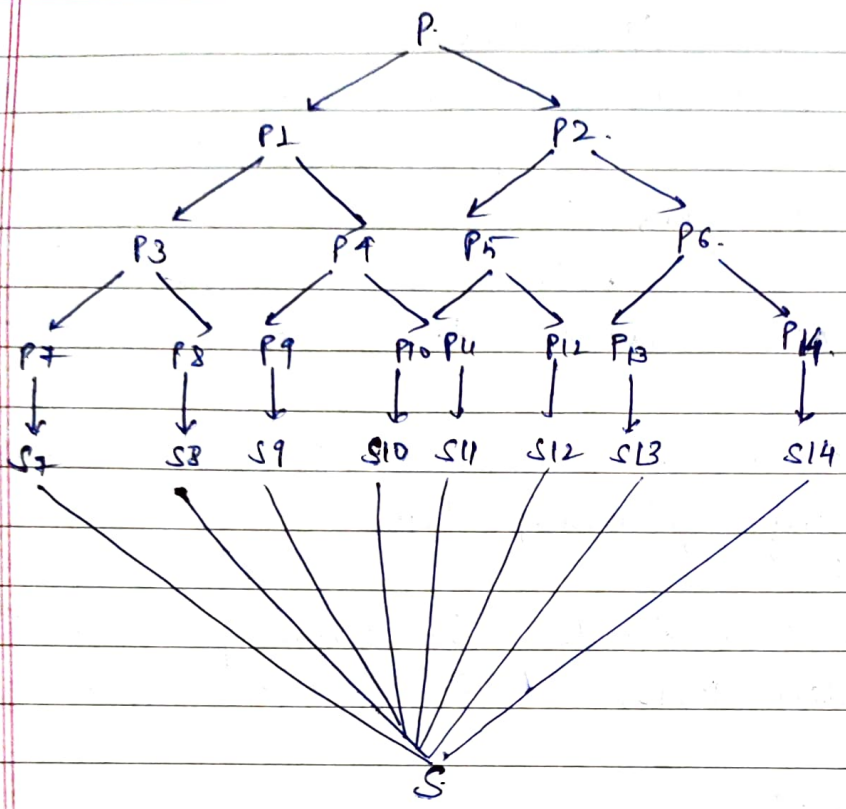
* Idea behind Divide and Conquer Algorithm:

Given a problem P of size $n = 2^k$.

Algorithm DAC(P):

- If 'n' is small, solve it.
- else.
  - divide P into 2 (or) more sub-problems $P_1, P_2, \ldots$ etc
  - DAC(P_1)
  - DAC(P_2).
  - :
- combine solutions of $S_1, S_2 \ldots$ etc of sub problems $P_1, P_2 \ldots$ etc

Problem



$n = 2^k$

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

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

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

★ Time Required to Solve Problem P:

P P1 P2

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

Where,

$f(n)$ = is the additional cost of combining solutions of sub-problems.

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

In general,

$$\boxed{T(n) = bT(n/2) + f(n)}$$

It is called recurrence relation of divide and conquer algorithm.

★ Examples:

(1) Solve $T(n) = 2T(n/2) + n$, using iteration (or) substitution method taking $n = 2^k$.

Sol:- Let $T(n) = 2T(n/2) + n$ ——(1).

$T(n) = 2\{2T(n/4) + n/2\} + n.$

$T(n) = 4T(n/4) + 2n.$

$T(n) = 4\{2T(n/8) + n/4\} + 2n.$

$T(n) = 8T(n/8) + 3n.$

$T(n) = nT(n/n) + kn.$

$T(n) = nT(1) + kn.$

$T(1)$ is negligibly small.

$\therefore \boxed{T(n) = kn}$

Put $n = 2^k.$

$\log n = k \log 2$

$k = \frac{\log n}{\log 2}$

$\boxed{k = \log_2 n}$

$\therefore \boxed{T(n) = n \log_2 n}$

$\therefore$ Complexcity $O(n \log_2 n)$

(2) In an algorithm, we divide large problem into 3-equal parts. and discard two of them, in a constant time, what is the complexicity of the algorithm for the size $n = 3^k$.

Sol^n: Let $T(n) = T(n/3) + c$ ——— (1)

$T(n) = T(n/9) + c + c.$

$T(n) = T(n/9) + 2c.$

$T(n) = T(n/27) + c + 2c$

$T(n) = T(n/27) + 3c$

$T(n) = T(n/n) + kc.$

$T(n) = T(1) + kc$

$T(1)$ is negligibly small

$\therefore \boxed{T(n) = kc}$

Put $n = 3^k$.

$\log n = k \log 3$.

$k = \frac{\log n}{\log 3}$

$\boxed{k = \log_3 n}$.

$\therefore \boxed{T(n) = c \log_3 n}$

$\therefore$ Complexity $O(c \log_3 n)$.

* Merge sort Algorithm:

Merge sort keeps on dividing a given list of numbers into equal halves if possible until it can no more be divided. Then mergesort combines the smaller sorted list keeping new list sorted.

Algorithm:

Step 1: If there is only one element in the list it is sorted, return.

Step 2: Divide the list recursively into two halves until it can no more be divided.

Step 3: Merge the smaller lists into new list in a sorted order.

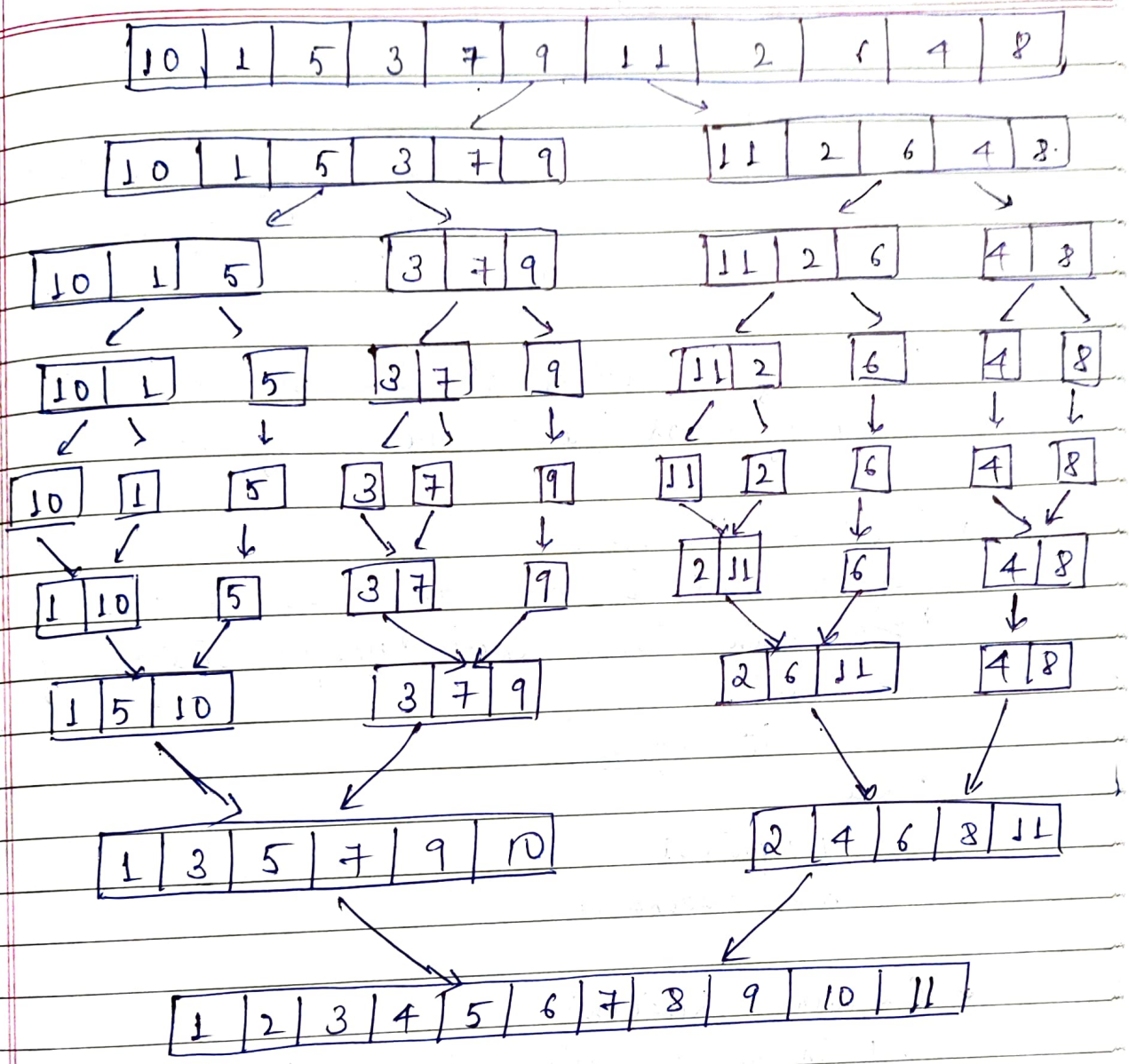
* Examples.

① Consider an unsorted list of numbers and sort it by merge sort algorithm.

10, 1, 5, 3, 7, 9, 11, 2, 6, 4, 8.

Sol: Given nos;

10, 1, 5, 3, 7, 9, 11, 2, 6, 4, 8.


10 1 5 3 7 9 11 2 6 4 8
10 1 5 3 7 9
11 2 6 4 8
10 1 5
3 7 9
11 2 6
4 8
10 1
5
3 7
9
11 2
6
4
8
10
1
5
3
7
9
11
2
6
4
8
1 10
5
3 7
9
2 11
6
4 8
1 5 10
3 7 9
2 6 11
4 8
1 3 5 7 9 10
2 4 6 8 11
1 2 3 4 5 6 7 8 9 10 11