→ Every number → represented as power of 2. eg → $14 = 2^3 + 2^2 + 2^1$, $5 = 2^2 + 2^0$

⇓ fenwick ⇓

Tree works on this concept

→ Say we want getSum(13) → ie. find sum of all elements from 0-13 index (total 14 elements)

⇓

$$\text{getSum}(13) = \sum_{i=0}^{7} arr[i] + \sum_{i=8}^{11} arr[i] + \sum_{i=12}^{13} arr[i]$$

2^3 elements | 2^2 elements | 2^1 elements

∵ 14 was $2^3 + 2^2 + 2^1$

⇒ ∴ Binary Indexed Tree → makes an array → each element in the array → stores sum of ranges → where size of ranges are in powers of 2.

→ Total terms $\leq \log(N)$ → ∵ $\log(N)$ = no. of bits in the binary representation.

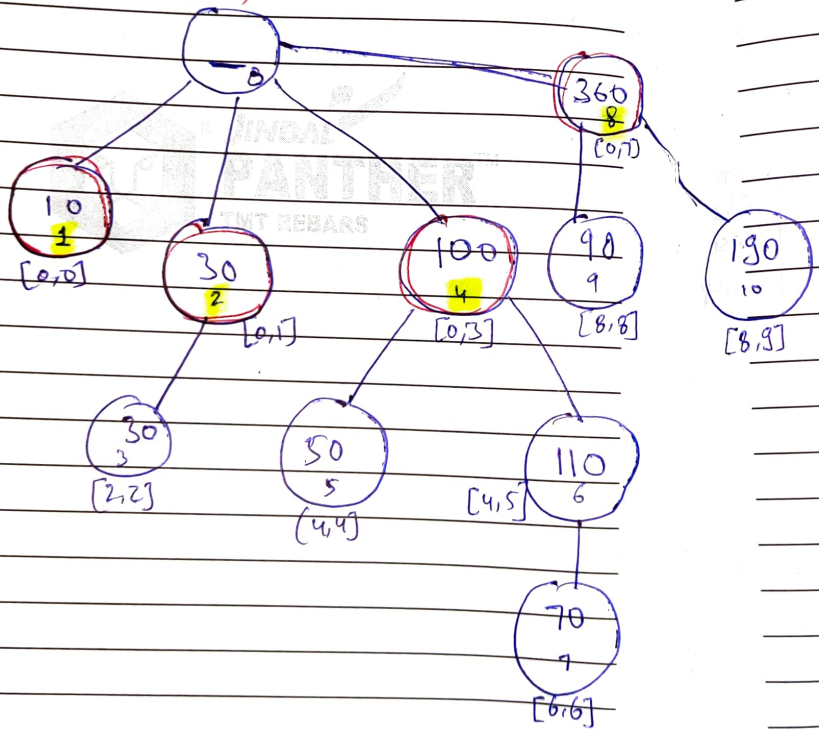
Date:

→ How it looks like:

arr[] = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
index: 0 1 2 3 4 5 6 7 8 9

BinIndTree[] = [—, 10, 30, 30, 100, 50, 110, 70, 360, 90, 190]
index: 0 1 2 3 4 5 6 7 8 9 10
[0,0] [0,1] [2,2] [0,3] [4,4] [4,5] [6,6] [0,7] [8,8] [8,9]

dummy node (never addressed)



⇒ Say we want getSum(5) → $\sum_{i=0}^{3} arr[i] + \sum_{i=4}^{5} arr[i]$

= BinTree[4] + BinTree[6]

= 100 + 110 = 210

each node represents $\sum_{x=i}^{j} arr[x]$ where [i,j] range is a power of 2.

Date:

→ How is it constructed?

→ Consider prev. example: we had 10 elements in the array.

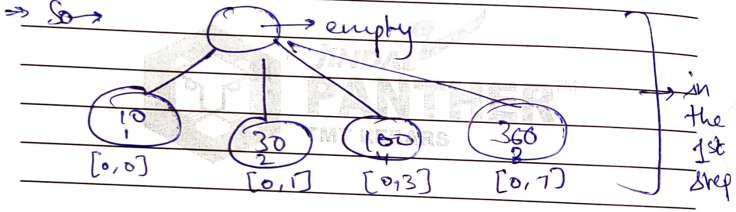
∴ 1 2 3 4 5 6 7 8 9 10

powers of 2.

↓

each of them represents range [0, i-1] & prefix sum

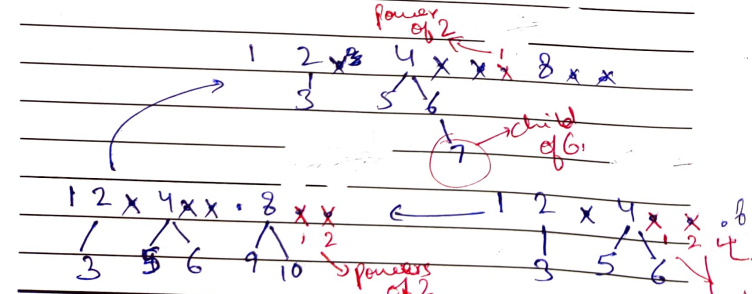
→ So →



∴ 1 2 4 8 → they are done

consider remaining indices as subarrays

↓ in next iteration



1 2 X 4 . . . 8 . .

3 → 3 comes as child of 2

∵ (2° = 1) ∴ could be represented as power of 2.

3 is NOT power of 2

→ can be represented as powers of 2.