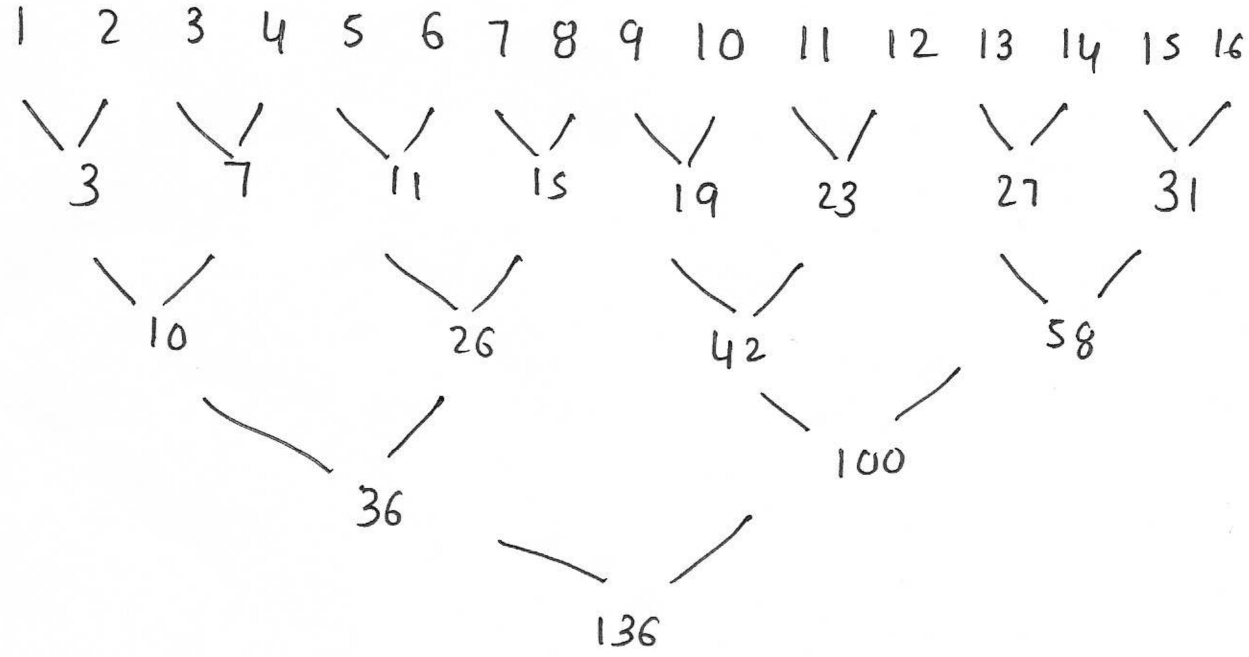
Problem 2: Parallel prefix algorithm

Assume we have an input array of size 'N' and that we have access to N processors and one for each element in the input array.

On observing an example input array as follows



Computing the odd-even pair sum's is the first step in the parallel prefix algorithm

Our output would look like

1 $\underline{3}$ 6 $\underline{10}$ 15 $\underline{21}$ 28 $\underline{36}$ 45 $\underline{55}$ 66 $\underline{78}$ 91 $\underline{105}$ 120 $\underline{13}$

As you can see, the even positions in the output array ~~are~~ can be obtained by recursively taking levels from the tree shown above.

For example to compute Output(2) = 3 we take the sub-tree, ① ② and obtain ③

to compute output(4) = 10 = sub trees

$$\begin{tikzcd}[column sep=small] 1 \arrow[dr, dash] & & 2 \arrow[dl, dash] & 3 \arrow[dr, dash] & & 4 \arrow[dl, dash] & \\ & 3 & & & 7 & = & 10 \end{tikzcd}$$

to compute output(6) = 21 = ③ ⑦ ⑪ => ㉑

∴ we can conclude that you can fill in the even entries of the algorithm by recursing on arrays of size $n/2$ of pairwise sums (go down the tree completely to finish the step)

Step 3: To fill in the odd entries in the output, we just have add the input in the position and the even element before it.

For example to get '6' at position '3', add input at position (3) which is '3' with the previous even element (3), the sum is '6'.

To summarize:

1) compute pairwise sum on adjacent processors till you have the tree sort of structure

2) To compute the even entries of the output, keep recursing on arrays of size $n/2$ of these pair sums

3) To compute the odd entries, get the element in input array and add it to the even positioned element in output.

Therefore, we can conclude that parallel prefix algorithm is recursive.

P.S: This would apply to the case where we only have 'P' processors where $P << N$,

we would have 'P' processors with (N/P) elements each,

compute prefix algorithm on the 'p' processors and then a scan on the results of the processors (P) as a whole.