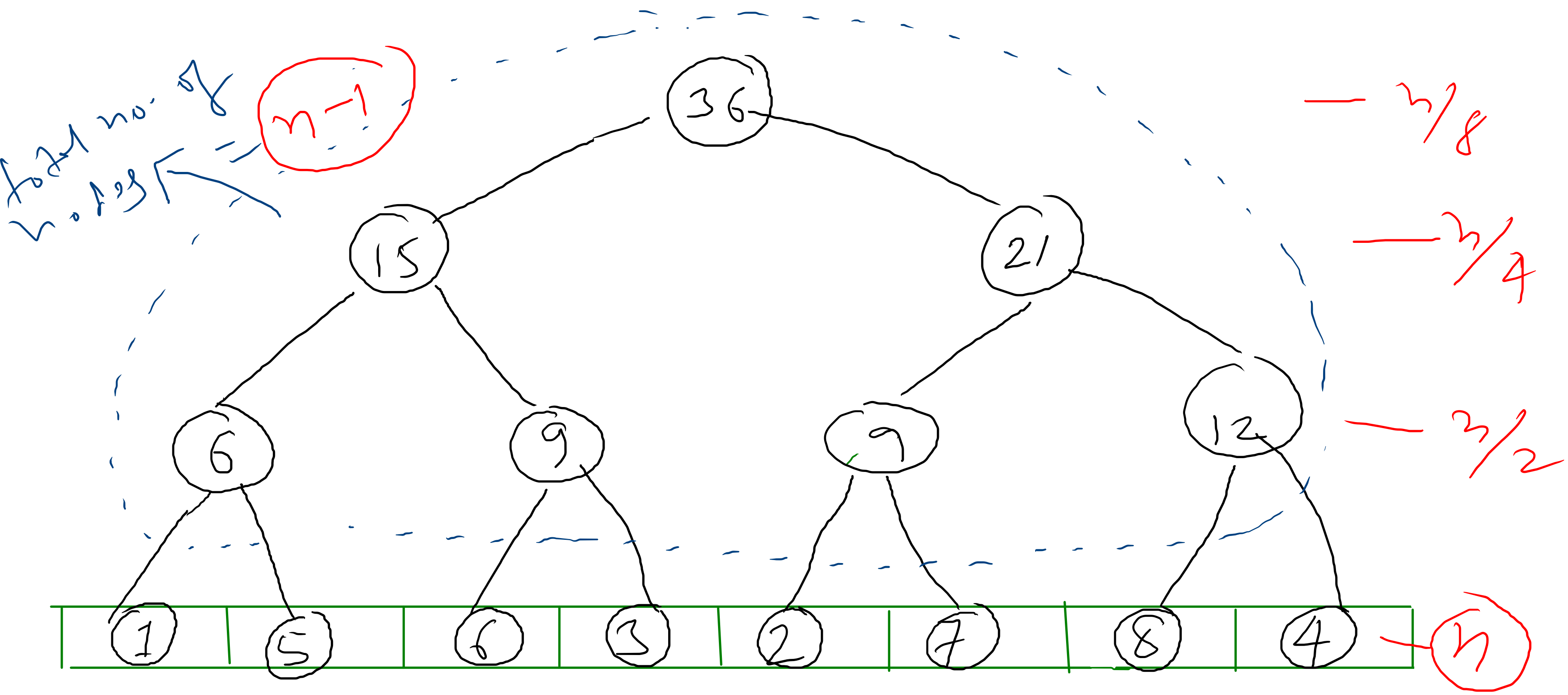
Segment tree : Each node of a segment tree represents some operation on a particular segement of a given array.



Question : How many nodes will be there in segement tree if input array size is n?
Solution : 2n -1
proof : n + n/2 + n/4 +n/8 + .. = n[1 +1/2+1/4+1/8+...] = n[1 + 1] = 2n

If size of array is power of 2, then the tree formed is FULL TREE.
In general, for sake of easy implementation we transform the input array as power of 2 with some dummy elements.

Post 2^k transformation, size of array:
size_of_array_post_power_2_transfomation <=2*(original_size)
size <=2n

Segment tree Array based implementation

Strategy 1: Implementation with binary tree node

This requires storing a lot of redundant information.

Strategy 2: Implementation with plain array using BFS indexing

Let's keep the root vertex at index 0, and its two child vertices at indices 1 and 2.

If tree is rooted at index 0 and parent is at index i then formulae for child indices :

Left child index : 2i +1

Right child index : 2i +2

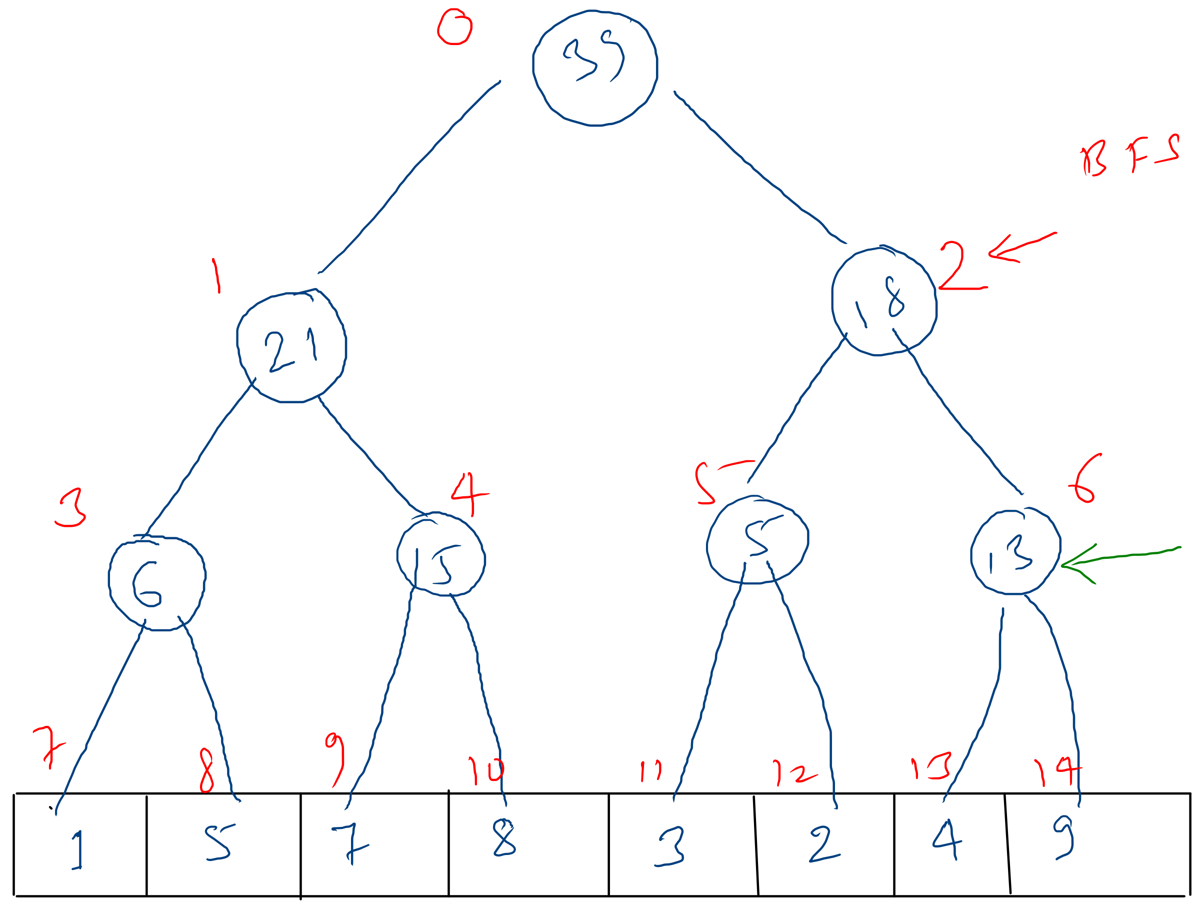
We need to store at most 4n vertices. It might be less, but for convenience we always allocate an array of size 4n. There will be some elements in the sum array, that will not correspond to any vertices in the actual tree, but this doesn't complicate the implementation.

So, we store the Segment Tree simply as an array tree[] with a size of four times the input size n.

Why do we need the tree size as 4n ?

Solution : To get the n as power of 2 we need at most 2n elements. If we have 2n elements at leaf node level then to construct the tree for these 2n leaf nodes we need additional (2n -1) nodes. Thus, in total we need 4n nodes.

Array based tree implmentation using BFS indexing



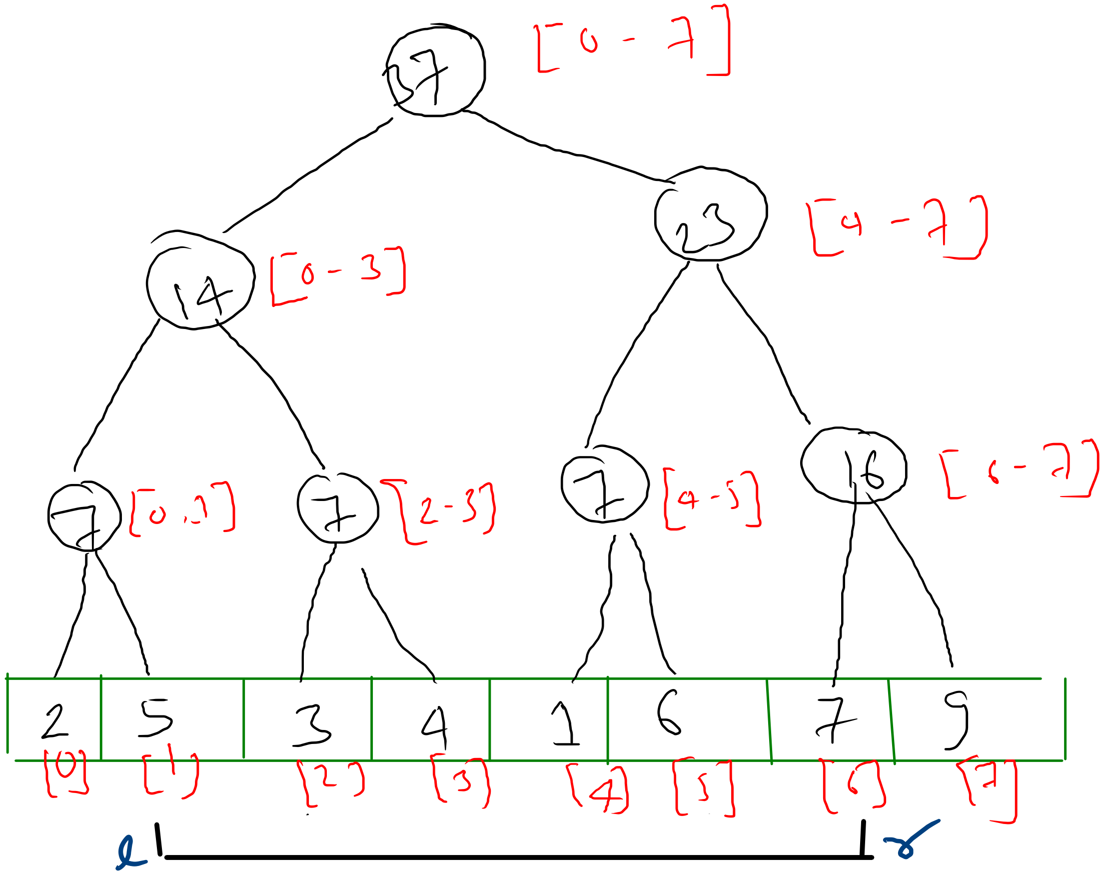
BFS index

$i = 6$

left_child_index = $2 \times 6 + 1$
= 13

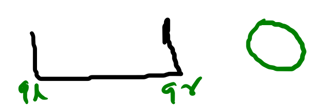
right_child_index = $2 \times 6 + 2$
= 14

Time complexity of range query on segment tree.
Eg. find the sum in array from index l =1 to index r= 6 ?

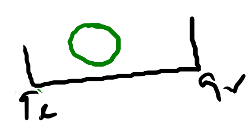


There can be three cases while traversing the tree to find the sum for given query range ?

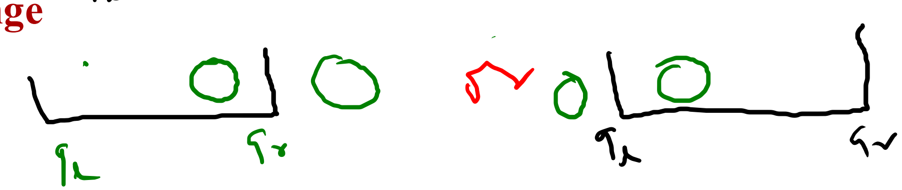
Case 1. encountered node lies completely outside the query range



Case2. encountered node lies completely inside the query range



Case3. range of encountered node is greater than the query range
means need to traverse both left and right child



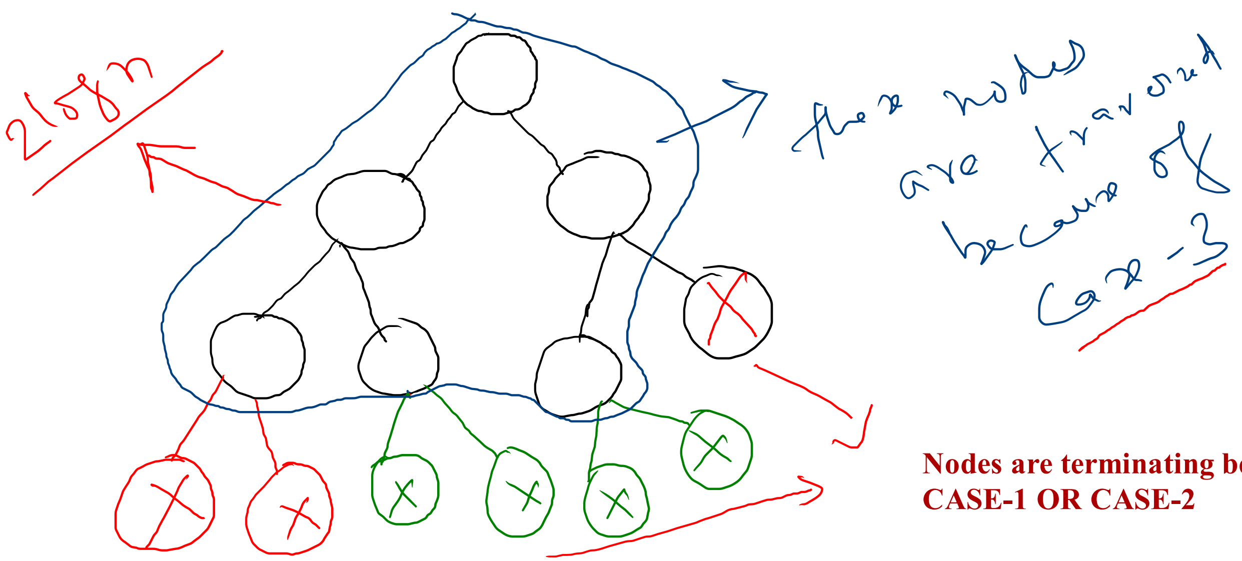
Time complexity of range query on segment tree.
Eg. find the sum in array from index l =2 to index r= 5 ?

CASE-1 and CASE-2 are recursion terminator.
CASE-3 keeps the recursion continued..
Means, we are recursively traversing the tree only in CASE-3. That is visiting the left subtree and right subtree together. Traversing left-subtree and right-subtree is required to reach at 'l' index and 'r' index of query.

To reach to 'l' index we need to traverse logn nodes.
To reach to 'r' index we need to traverse logn nodes.

Total number of nodes traversed because of CASE-3 = 2logn

Post CASE-3 traversal recursion will terminate because of either CASE-1 or CASE-2.



Nodes are terminating because of either CASE-1 OR CASE-2

Nodes terminated because of CASE-1 OR CASE-2 can be treated as leaf nodes of traversed tree for the given query.
This query bound traversed tree is just the part of whole tree. This doesnot represent the whole tree traversal.

As we know that total number of leaf-nodes are same as the number of internal nodes. Since count of internal nodes(CASE-3) are 2logn, So, the count of leaf-nodes(CASE-1 OR CASE-2) will be 2logn.

Thus, total node traversed by recursion to answer the range query is 4logn.