Segment Tree | Set 1 (Sum of given range)

Let us consider the following problem to understand Segment Trees.

We have an array arr[0 . . . n-1]. We should be able to  
**1** Find the sum of elements from index l to r where 0 <= l <= r <= n-1

**2** Change value of a specified element of the array to a new value x. We need to do arr[i] = x where 0 <= i <= n-1.

[**Recommended: Please solve it on “*PRACTICE* ” first, before moving on to the solution.**](https://practice.geeksforgeeks.org/problems/count-multiples-of-five/0)

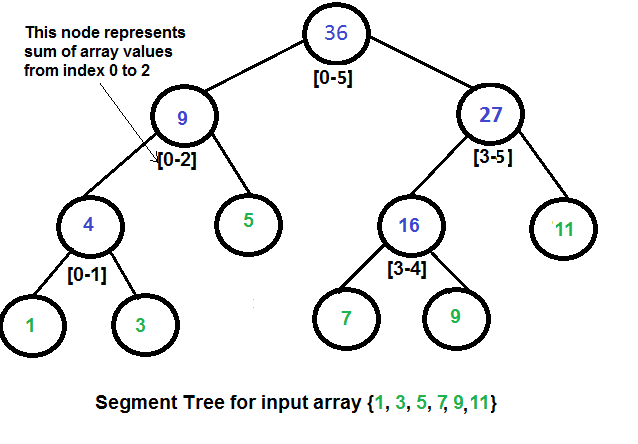
A **simple solution** is to run a loop from l to r and calculate the sum of elements in the given range. To update a value, simply do arr[i] = x. The first operation takes O(n) time and the second operation takes O(1) time.

**Another solution** is to create another array and store sum from start to i at the ith index in this array. The sum of a given range can now be calculated in O(1) time, but update operation takes O(n) time now. This works well if the number of query operations is large and very few updates.

What if the number of query and updates are equal? **Can we perform both the operations in O(log n) time once given the array?** We can use a Segment Tree to do both operations in O(Logn) time.

**Representation of Segment trees**  
**1.** Leaf Nodes are the elements of the input array.  
**2.** Each internal node represents some merging of the leaf nodes. The merging may be different for different problems. For this problem, merging is sum of leaves under a node.

An array representation of tree is used to represent Segment Trees. For each node at index i, the left child is at index 2\*i+1, right child at 2\*i+2 and the parent is at .



**How does above segment tree look in memory?**  
Like Heap, the segment tree is also represented as an array. The difference here is, it is not a complete binary tree. It is rather a full binary tree (every node has 0 or 2 children) and all levels are filled except possibly the last level. Unlike Heap, the last level may have gaps between nodes. Below are the values in the segment tree array for the above diagram.

*Below is memory representation of segment tree for input array {1, 3, 5, 7, 9, 11}  
st[] = {36, 9, 27, 4, 5, 16, 11, 1, 3, DUMMY, DUMMY, 7, 9, DUMMY, DUMMY}*

The dummy values are never accessed and have no use. This is some wastage of space due to simple array representation. We may optimize this wastage using some clever implementations, but code for sum and update becomes more complex.

**Construction of Segment Tree from given array**  
We start with a segment arr[0 . . . n-1]. and every time we divide the current segment into two halves(if it has not yet become a segment of length 1), and then call the same procedure on both halves, and for each such segment, we store the sum in the corresponding node.  
All levels of the constructed segment tree will be completely filled except the last level. Also, the tree will be a [Full Binary Tree](https://www.geeksforgeeks.org/binary-tree-set-3-types-of-binary-tree/) because we always divide segments in two halves at every level. Since the constructed tree is always a full binary tree with n leaves, there will be n-1 internal nodes. So the total number of nodes will be 2\*n – 1. Note that this does not include dummy nodes.

**What is the total size of the array representing segment tree?**  
If n is a power of 2, then there are no dummy nodes. So the size of the segment tree is 2n-1 (n leaf nodes and n-1) internal nodes. If n is not a power of 2, then the size of the tree will be 2\*x – 1 where x is the smallest power of 2 greater than n. For example, when n = 10, then size of array representing segment tree is 2\*16-1 = 31.  
An alternate explanation for size is based on heignt. Height of the segment tree will be . Since the tree is represented using array and relation between parent and child indexes must be maintained, size of memory allocated for segment tree will be .

**Query for Sum of given range**  
Once the tree is constructed, how to get the sum using the constructed segment tree. The following is the algorithm to get the sum of elements.

int getSum(node, l, r)

{

if the range of the node is within l and r

return value in the node

else if the range of the node is completely outside l and r

return 0

else

return getSum(node's left child, l, r) +

getSum(node's right child, l, r)

}

**Update a value**  
Like tree construction and query operations, the update can also be done recursively. We are given an index which needs to be updated. Let *diff*be the value to be added. We start from the root of the segment tree and add *diff*to all nodes which have given index in their range. If a node doesn’t have a given index in its range, we don’t make any changes to that node.