Introduction

Let us consider a scenario where we are given an array A and we have to perform two operations

- Calculate the sum of the first x elements
- Modify an element present in the array at some index idx to some value y.

One of the solutions could be to run a loop for the first x elements and calculate the sum in O(n) time (where n is the number of elements in the array), and update operation can be done as A[idx] = y in O(1) time.

Another solution could be to keep another prefix sum array, where we store the sum of the first i elements at i-th index. Then calculating the sum would take O(1) time, but the update operation will take O(n) time as we will need to update all the indices greater than i if we update index i .

Now the third solution can be using a segment tree that takes O(log n) time for both operations. **An alternative solution is to use a Fenwick or a binary indexed tree which is easier to implement and takes less space as compared to a segment tree.** In addition to this, the time complexity for both operations still remains O(log n).

Binary Indexed Tree(BIT) is an array of size n+1, where each cell in the array stores the sum of some values in the input array. We will see below how.

Let's consider an input array A as [1,3,5,11,7,4,6,7]. Now since BIT[] has n+1 elements, we don't store anything at index 0.

indexes	0	1	2	3	4	5	6	7
values	1	3	5	11	7	4	6	9

Input array A

Each integer can be represented as a sum of powers of two, for example, we will start with index 1, where index 1 can be written as 2^0, so index 1 of BIT[] will store the sum of values from index 0 to 0, represented as (0,0). Similarly, we represent each index of the BIT[] array as the sum of powers of two.

Index Of BIT[]	Index can be represented as	Parent Index	Range to store the sum
1	2^0	0	(0,0)
2	2^1	0	(0,1)
3	$2^1 + 2^0$	2	(2,2)
4	2^2	0	(0,3)
5	$2^2 + 2^0$	4	(4,4)
6	$2^2 + 2^1$	4	(4,5)
7	$2^2 + 2^1 + 2^0$	6	(6,6)
8	2^3	0	(0,7)

Parent Index: It can be calculated from the column 'Index can be represented as', by just taking the sum of all terms and leaving the last term. The representation which contains only one term has a parent index as 0. This means that in the binary representation of the index, we have to remove the last set bit.

Therefore to calculate parent index

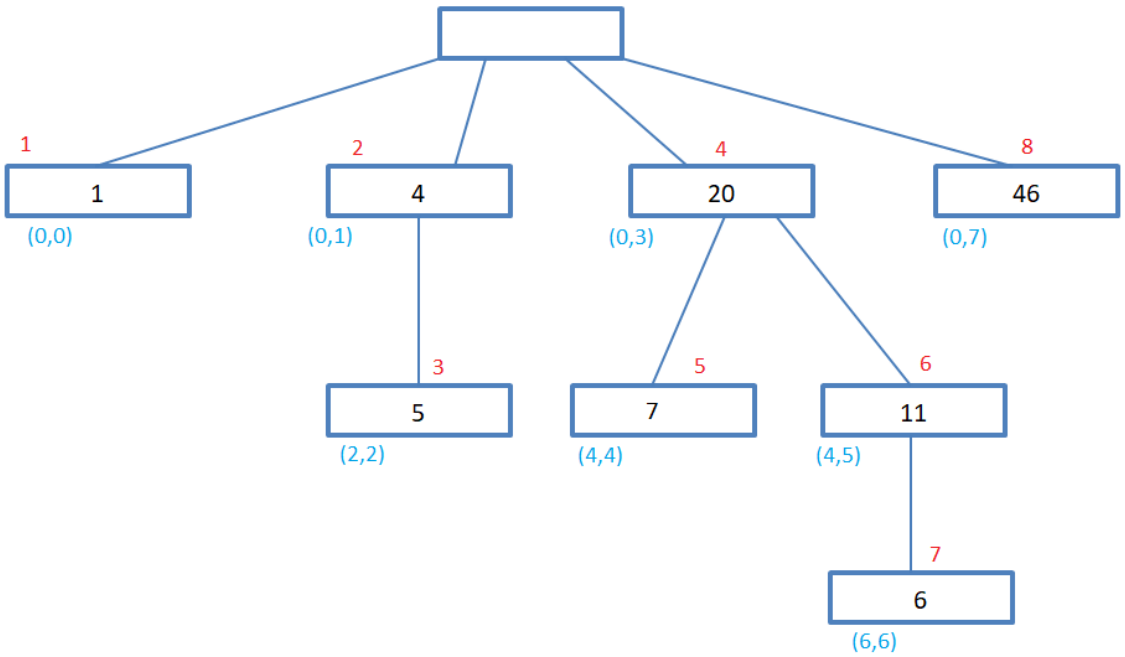
parentIndex = idx - (idx & (-idx)).

Example: 7 can be represented as 111 and -7 is represented as 001 in binary.

Now 7 & (-7) = 001_2 = 1 in decimal .

Now parentIndex = 7 - 1 = 6

Range to store the sum: It can be calculated as (parent Index, Index of BIT[] - 1).



The child node BIT[x] of the node BIT[y] stores the sum of the elements between y(inclusive) and x(exclusive): arr[y,...,x).

So let's take an example of index 6, the value at index 6 = sum of elements of input array from index 4 to 5, where 4 is the parent index and 5 is the current element index - 1 i.e. 6 - 1 = 5.

Similarly index 4 of BIT[] stores the sum of elements in the input array from index 0(parent index) to current index - 1, i.e. 3.