**Range Sum Queries**

**1** To find the sum of the first i elements.   
**2**Modify the value of a specified element of the array arr[i] = x where 0 <= i <= n-1.

A **simple solution** is to run a loop from 0 to i-1 and calculate the sum of the elements.

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 simple solution is to create an extra array and store the sum of the first i-th elements at the

i-th index in this new array.

The sum of a given range can now be calculated in O(1) time, but the update operation takes O(n) time now.

This works well if there are a large number of query operations but a very few numbers of update operations.

**Range Sum**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 | 2 | 0 | 6 | 5 | -1 | 2 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

The First Line Represents the Array elements and the second row represents the index of the array

**PreFixSum**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 | 5 | 5 | 11 | 16 | 15 | 17 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Problems With the above approach

If the array has many updates it takes O(n) time

Prefixsum doesn’t scale well when there are many updates

**Problems –**

Sligthly Complicated

Inverse Case O(n) Space

Worst Case O(n)

**Could we perform both the query and update operations in O (log n) time?**

One efficient solution is to use [Segment Tree](https://www.geeksforgeeks.org/segment-tree-set-1-sum-of-given-range/) that performs both operations in O(Logn) time.

*An alternative solution is Binary Indexed Tree, which also achieves O(Logn) time complexity for both operations.*

*Compared with Segment Tree, Binary Indexed Tree requires less space and is easier to implement.*

**Representation**   
Binary Indexed Tree is represented as an array.

Let the array be BITree[].

Each node of the Binary Indexed Tree stores the sum of some elements of the input array.

The size of the Binary Indexed Tree is equal to the size of the input array, denoted as n.

**Benefits of FenWick Tree (BIT)**

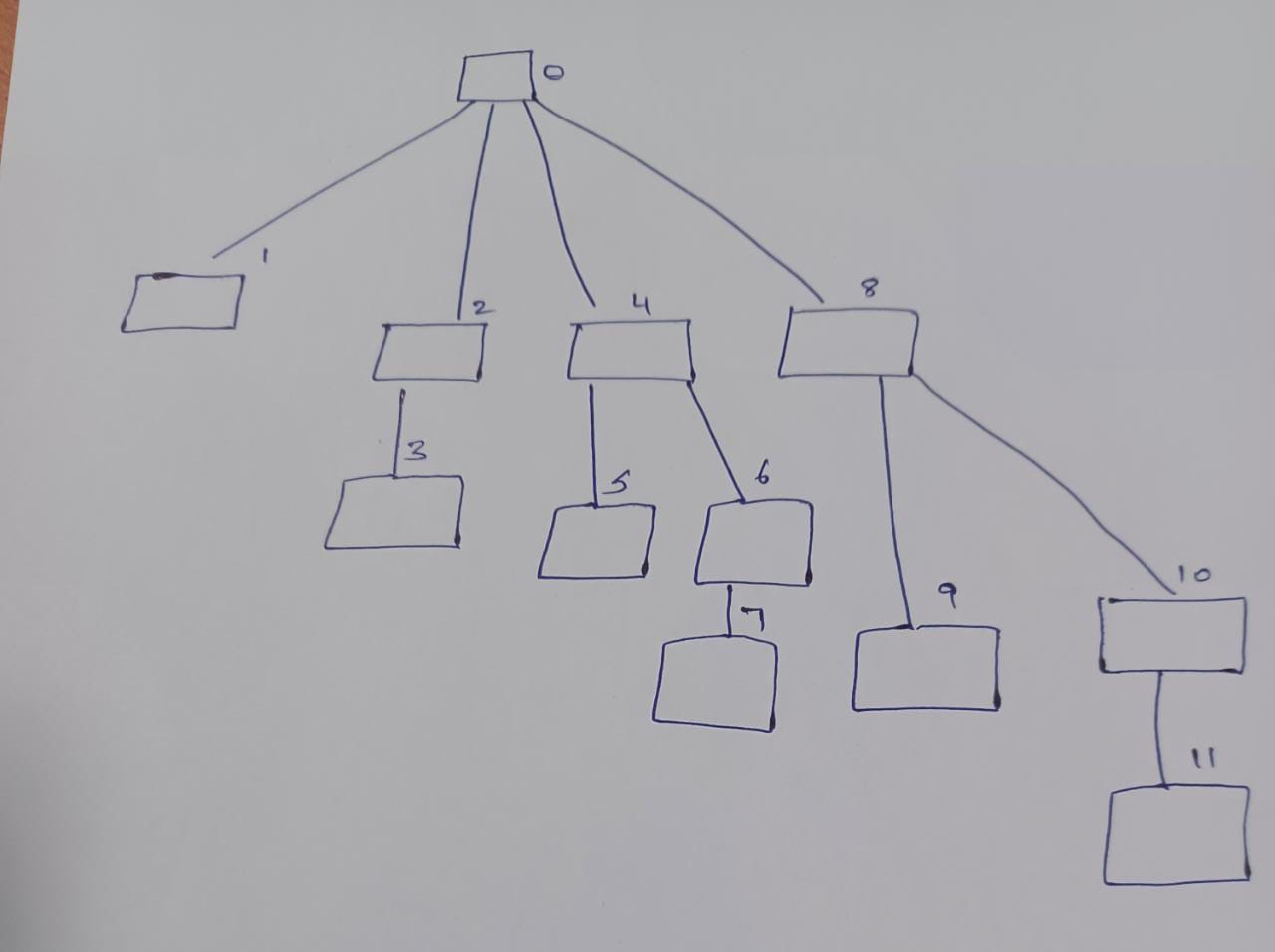
**Both Update and PreFix Sum required O(logN) time complexity  
Easy to Implement**

**Requires Very Less Space**

**FenWick Tree**

**Consider the following Array**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 2 | -1 | 6 | 5 | 4 | -3 | 3 | 7 | 2 | 3 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |



**Why is the parent of 2 – 0?**

Binary Representation of 2 is 0010

We have to Shift the Right Most Set Bit to 0 - 0000 -0

**Which is why the parent of 2 is 0**

Binary Representation of 8 is 1000

Shifting the Right Most Set Bit to 0 – 0000 – 0

**The Parent of 8 is 0**

Binary Representation of 10 is 1010

Shifting the Right Most Set Bit to 0 – 1000 – 8

**The parent of 10 is 8**

Binary Representation of 11 is 1011

Shifting the Right Most Set Bit to 0 is 1010 – 10

**The Parent of 11 is 10**

Similary for other Elements

**To Know the Parent –**

Find the Binary Representation of the Elements

Shift the Right Most Set Bit ‘1’ to ‘0’.

Convert the Binary Number to Decimal Number.

**Understanding the Range Covered by an Index (STEP-2)**

Understanding the Range [LB, RB]

For a given index i,

Right Bound (RB) = i

Left Bound (LB) = i - LSB(i) + 1

Thus, the Fenwick Tree at index i contains the sum of the elements in the original array from index [LB, RB].

**The LSB determines how many elements are summed at a particular index.**

Finding LSB(7)

The Least Significant Bit (LSB) is found using:

**LSB(x)=x&(−x)**

Calculate -7 in Binary

To get -7, we take the two’s complement of 0111:

Invert the bits: 1000

Add 1: 1001 (which is -7 in two’s complement representation)

**Perform Bitwise AND**

7&(−7)=0111&1001

Performing bitwise AND:

0111

1001

------

0001 (which is 1 in decimal)

So, LSB(7) = 1.

**Left Bound = Index−LSB(Index)+1**

**= 7 – 1 + 1 = 7**

The right bound is always the index itself.

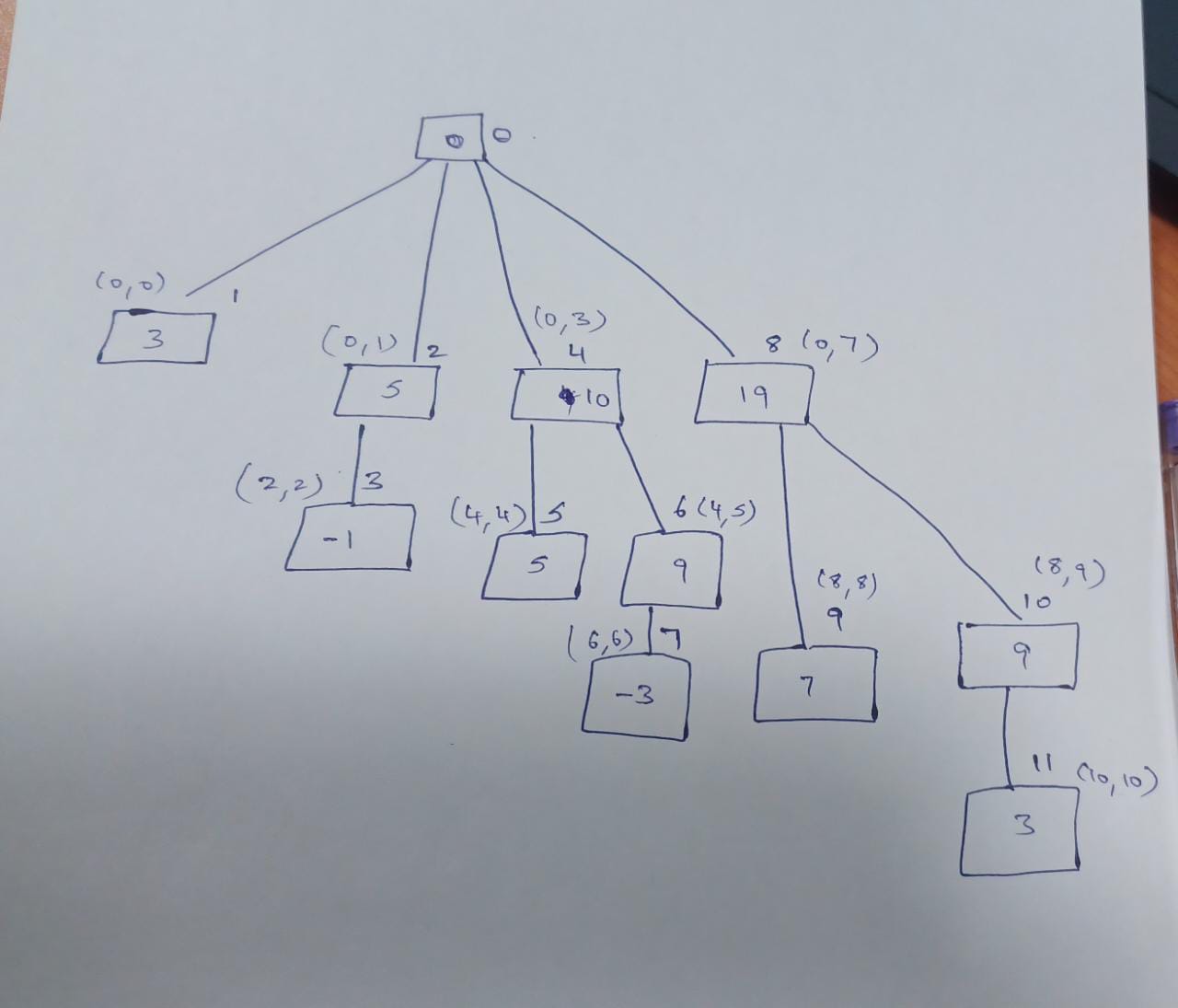
**Right Bound=Index**

Index = [LeftBound, RightBound]

Index(7) = (7,7) 1 – based

**The Below Table shows the 0 Based Index Range and Index Range Sum(Last 2 Columns)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Index** | **Convert to Binary** | **Find(LSB) LSB(x)= x & (-x)** | **LSB (Decimal)** | **Left Bound =**  **Index−LSB(Index)+1** | **Index Range (0-based) Index Range=[Left Bound,**  **Right Bound]** | **Index Range Sum** |
| **1** | 1 | 1 | 1 | 1 | [0] | 3 |
| **2** | 10 | 10 | 2 | 1 | [0,1] | 3 + 2 = 5 |
| **3** | 11 | 1 | 1 | 3 | [2] | -1 |
| **4** | 100 | 100 | 4 | 1 | [0,1,2,3] | 3+2+(-1)+6=10 |
| **5** | 101 | 1 | 1 | 5 | [4] | 5 |
| **6** | 110 | 10 | 2 | 5 | [4,5] | 5+4=9 |
| **7** | 111 | 1 | 1 | 7 | [6] | -3 |
| **8** | 1000 | 1000 | 8 | 1 | [0,1,2,3,4,5,6,7] | 3+2+(-1)+6+5+4 +(-3)+3=19 |
| **9** | 1001 | 1 | 1 | 9 | [8] | 7 |
| **10** | 1010 | 10 | 2 | 9 | [8,9] | 7+2=9 |
| **11** | 1011 | 1 | 1 | 11 | [10] | 3 |



We have find the above steps with the help of the below line

To find the Prefix sum of (0,5)

We have to go 6th Node ,the sum is 9

The Parent of 6th Node is 4, The sum is 10+9,

The Parent of 4th Node is 0, The sum is 19

**How to Get the Parent (Step-1)**

1. 2’s Complement
2. AND it with original number
3. Subtract from original number

Binary Represent of 7 – 111

2’s Compliment – 000+1 =001

AND it with original number

111

001

\_\_\_\_

001

\_\_\_

3)111-001=6

Look at other example

Take the number – 10

Binary Representation of 10 is 1010

1)2’s Compliment is 0101 + 1 = 0110

2) AND it with the Original Number

1010 AND 0110 = 0010

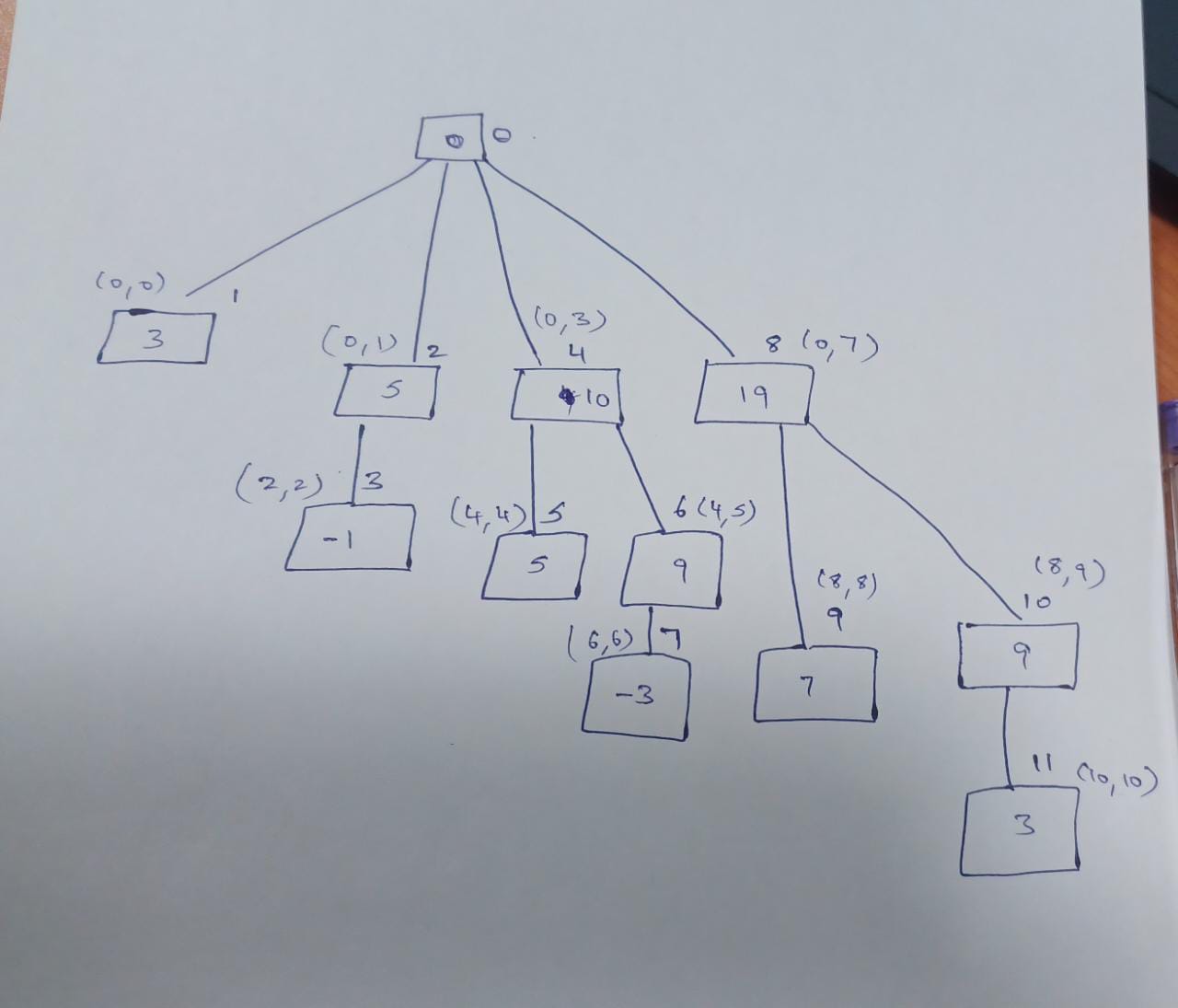
3) Subtract from Original Number = 1010 – 0010 = 1000(8)

10’s Parent is 8

**GET NEXT (Step-2)**

1. 2’s Compliment
2. AND with Original Number
3. Add to the Original Number

Same Like Previous



Applications Fenwick tree can be used to calculate Range Sum,

i.e., finding the sum within range. Sum(1,7) in the below example

array is Value 5 2 9 -3 5 20 10 -7 2 3 -4 0 -2 15 5

Index 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

5 + 2 + 9 + (-3) + 5 + 20 + 10 = 48 in traditional way, but by using Fenwick tree we can do this in minimum time.

The following method demonstrates that Range Sum Calculation.

int sum(int i)

{

int sum = 0;

while ( i > 0 )

{

sum +=BIT[i];

i -= i & -i; //flip the last set bit

}

return sum;

}

Example: - to find sum from the range 1 to 7

we use Sum(7) = sum(00111) = BIT[00111] + BIT[00110] +BI T[00100]

= BIT[7] +BIT[6] + BIT[4]

= range(7,7) + range(5,6) + range(1,4)

= 10 + 25 + 12 =48

To compute the sum (7)

while loop iterates only three times

Sum (8) = sum (01000) = BIT [01000] = BIT [8] =range (8,8) =41

To compute sum(8) while loop iterates only once.

This is faster to compute the range sums in large arrays