Fenwick or Binary Indexed Trees

→ Use case :- Prefix sum of an array

Eg

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

Sum from index 0 to 4 = 16
Sum from index 0 to 6 = 17

Alternate - ① prefix based sum array.

Take another array ⇒

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

(cumulative sum at each index)

Disadvantage -: not suitable if array is large and there are many updates in the array.

② Segment tree
complicated, high complexity

So, we use fenwick tree.

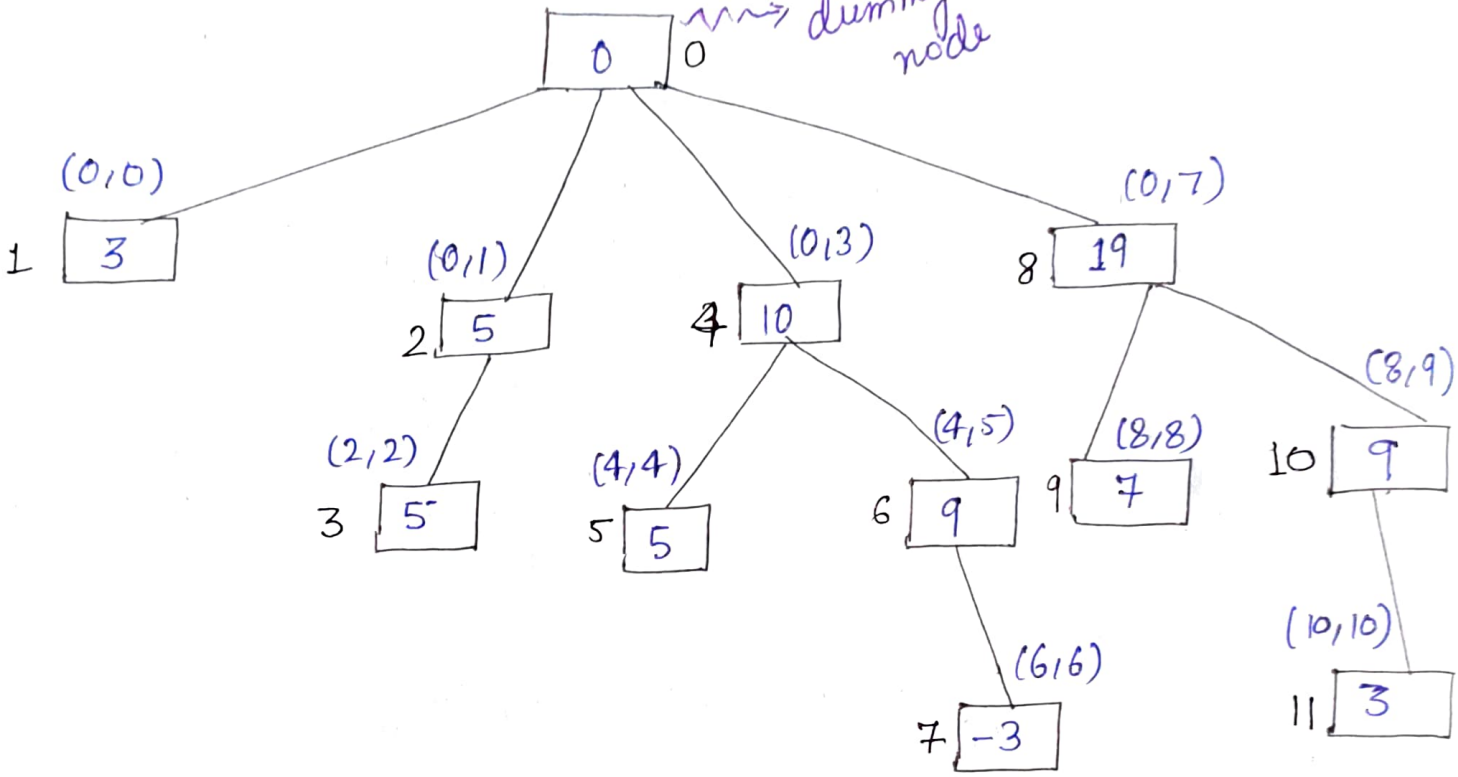
Space = O(n)
Time to search = O(log n)
Time to update = O(log n)
Time to create tree = O(n log n)

Eg Create a fenwick tree given an array -:

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

In tree, to find parent → flip the right-most bit in binary form.

So,
2's parent ⇒ 10 → 00 → 0
8's parent ⇒ 1000 → 0000 → 0
10's parent ⇒ 1010 → 1000 → (8) → (0)
11's parent ⇒ 1011 → 1010 → (10)



Every number can be represented in binary form,

$1 = 0 + 2^0$
$2 = 0 + 2^1$
$3 = 2^1 + 2^0$
$4 = 0 + 2^2$

$5 = 2^2 + 2^0$
$6 = 2^2 + 2^1$
$7 = 2^2 + 2^1 + 2^0$
$8 = 0 + 2^3$

$9 = 2^3 + 2^0$
$10 = 2^3 + 2^1$
$11 = 2^3 + 2^1 + 2^0$

}
starting from 0,
next 0 to 2^2 (4) elements are stored in 4th cell.

Eg ⇒ Given a range, find prefix sum.
(Searching)

→ range = 0 to 5

Take index 6.

Goto node 6 ⇒ 9

Goto parent of 6th node (4th node) ⇒ 9+10

Goto parent of 4th node (0th node) ⇒ 9+10+0

= 19 (sum)

flip the right-most bit in the binary representation.

→ range 0 to 9

Take index 10, value = 9

parent of 10th node ⇒ 8th node, value = 9+19

parent of 8th node ⇒ 0th node, value = 9+19+0

sum = 28

⇒ Time to get the value of sum = height of fenwick tree

In worst case = O(log n)

for prefix sum

To get parent of a node

- Get 2's complement of a number
- AND it with the original number.
- Substract from original number.

Eg parent of 7

binary = 1 1 1

2's complement = 000 + 1 = 001
(flip + add 1)

AND ⇒
$$\begin{array}{r} 1\ 1\ 1 \\ 0\ 0\ 1 \\ \hline 0\ 0\ 1 \end{array}$$

subtract from original no. =
$$\begin{array}{r} 1\ 1\ 1 \\ -\ 0\ 0\ 1 \\ \hline 1\ 1\ 0 \end{array} = 6$$

∴ 6th node is parent of 7th node.

Efficiently filling the tree (O.(logn))

Get Next –; (get next node & update)

2's complement
AND with original number
add to original number.

⇒ Create binary indexed tree = O(n logn)
{n elements}

Update binary indexed tree = O(logn)

Basic idea –:

Given an array of size N,

a – $N < 10^5$ Queries $Q \leq 10^5$

Queries Q ⇒ (1) Given index i, replace a[i] with 'x'.

(11) Sum of 'l' to 'r' (generally gives TLE)

So, we use segment trees (more complex)

or binary indexed trees (less complex).

⇒ Bit manipulation trick to find rightmost set bit in a binary representation of a number –:

Eg: $x = 1\ 1\ 0\ 1\ 1\ 0\ \underline{1}\ 0\ 0$

$x = a\ 1\ b \rightarrow (0\,0\,0 \ldots\ldots 0)$

$-x$ = 2's complement of x

$= (x)' + 1$

$= (a\ 1\ b)' + 1$

$= a'\ 0\ (1\,1\,1 \ldots 1) + 1$

$= a'\ 1\ (0\,0 \ldots 0)$

$-x = a'\ 1\ b$

So, $x = a\ 1\ b$

& $-x = a'\ 1\ b$

$(00\ldots0)\ 1\ (00\ldots0)$

$\Rightarrow (x\ \&\ -x) = 00\ldots0\underline{1}00\ldots0$

only rightmost bit set.

$x - (x\ \&\ -x)$ = To remove rightmost set bit.

Eg $x = 6$ $\quad x = 110$

$-x = 001 + 1 \quad = 010$

$x \& -x = \begin{array}{r} 110 \\ 010 \\ \hline 010 \end{array} \Rightarrow x \& -x = 010$

$x - (x \& -x) = \begin{array}{r} 110 \\ 010 \\ \hline 100 \end{array}$

For binary indexed trees,

Given 'a' array of size N.

Bit array - 0 1 2 3 4 .... 7 8 9 10 11 12 13

Every index in Bit array stores partial sum

Let i & j be two indexes

j = remove last bit (i)

$j+1 \Rightarrow i$

$j = i - (i \& -i)$

bit[i] = sum of j+1 to i

Eg. bit[13] = $13 = 1101$

remove last bit $\Leftarrow 1100 \longrightarrow 12+1 = 13$

bit[13] = 13 → 13 (sum of 13 to 13 index)

Eg bit[12] = $1100 \rightarrow 1000 \rightarrow 8+1 \rightarrow 9$

bit[12] = 9 → 12 (sum of 9 to 12th index)

Eg bit[11] → $1011 \rightarrow 1010 \rightarrow 10+1 \rightarrow 11$

bit[11] = 11 → 11 (sum of 11 to 11th index)

bit[10] = 9 to 10th index.

So, using Bit array,

Sum (1, 13) = bit[13] + bit[12] + bit[8]

bit[13] → 13, 13

13 = 1101 → 1100 (12)

bit[12] → 9, 12

12 = 1100 → 1000 (8)

bit[8] → 1, 8

8 = 1000 → 0000 (0)

⇒ Hence,

```
int sum (int i) {
    int ans = 0;
    for ( ; i > 0 ; i -= (i & -i)) {
        ans += bit[i];
    }
    return ans;
}
```

⇒ O(log n) } size of array.

⇒ To find sum in the range of l to r,

l, r = sum(r) − sum(l−1)

How to construct 'Bit' array −:

Initially all values in the array are 0's.

If we want to add x at some index i,

Eg ⇒ 13 → 1101

+ 0001 (add the least set bit)

1110 → 14

+ 0010

10000 → 16 ⇒ sum of (1, 16)

+ 10000

100000 → 32 ⇒ sum of (1, 32)

So, we can find all the numbers that include 'x' while we add x to bit[i].

⇛ ~ add x in index 'i'

```
void update (int i, int x) {
    for (; i<=N; i+=(i&-i)) {
        bit[i] += x;
    }
}
```