Main game of BIT tree is that of the range of elements its going to store.

Inventor of Fenwick tree gave a very important unit with which we can play.

"Range" For any index

BIT[index] is going to store a range of elements from [index, index-1, index-2, ----]

⇓

few elements.

what is the # of elements its going to store?

Range = [index & (-index)]

⇓

for any number this is the Least Significant Bit and of its Binary representation.

⟹ Only thing in bit tree. Bit tree is a manipulation of LSB. If you get this you understand LSB.

Storage map.

index	binary	LSB value	Sum which is stored.
1	0001	1	arr[1]
2	0010	2	arr[2], arr[1]
3	0011	1	arr[3], arr[2]
4	0100	4	arr[4], arr[3], arr[2], arr[1]
5	0101	1	arr[5]
6	0110	2	arr[6], arr[5]
7	0111	1	arr[7]
8	1000	8	arr[8], arr[7], arr[6], arr[5], ..., arr[1]
9	1001	1	arr[9]
10	1010	2	arr[10], arr[9]

BIT[index] is going to store info of the range

BIT(arr[index], arr[index-1], arr[index-2], ...)

([index & (-index)])

(range)

Lets understand Sum Function. first
As its more intuitive.

sum(index)

or aggregation(index) ⇒ should give me (1 to index) all aggregated details.

lets say index = 7

7 ⇒ 0 1 1 1 → (BIT[7] = arr[7]) only this info is stored in BIT[7]

1 2 3 4 5 6 7 . . . .

⇓

BIT[7]

Now tell me → will the info of arr[7] stored in any node/index where (index < 7) ? will it be

No, right. Because BIT[index] stores details from

arr[index] ---- to arr[index - range + 1]

range = (index & -index)

(P.T.O)

1 2 3 4 5 6 7 - - - -

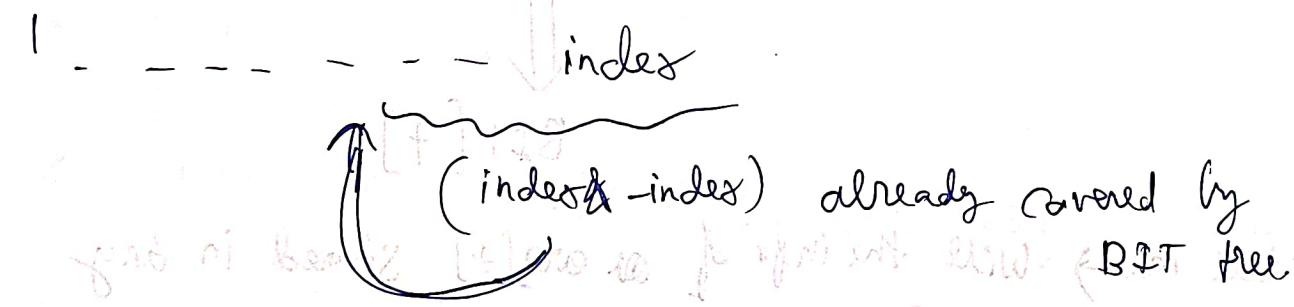
⇓

BIT[7].

BIT[index] → arr[index], - - - , arr[index − range + 1]

(range = index & (-index))

If this range is already covered in BIT[index], then we need to do rest of the searching.



So we can move our index to a position where rest of the details are stored.

$$newIndex = index - \frac{(index \,\&\, -index)}{range}.$$

Since newIndex = index - range

⇓

arr[index], ... arr[index-range+1]

So we have to reach an index where the details of (index - range + 1) is there.

So what is that index ⇒ [index - range + 1 − 1]

⇓ just the previous one

↳ [index - range]

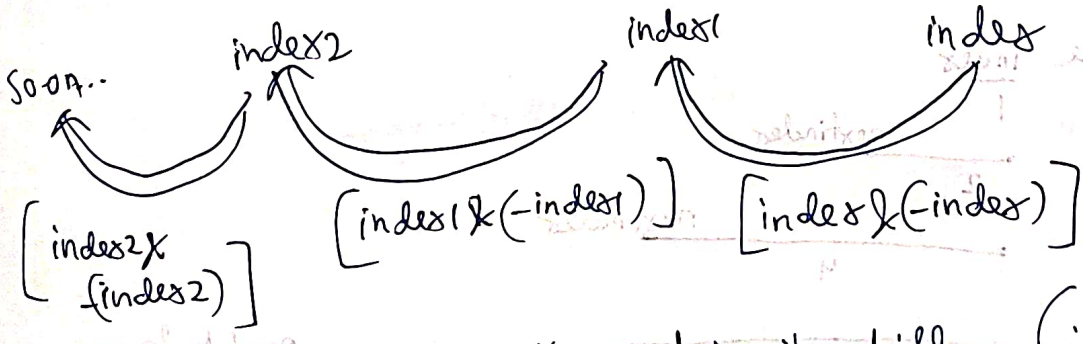
here we move to

index -= index & (-index).

⇓

(this is nothing but the LSB)

Each jump backwards we are just ~~reducing the~~ removing the LSB from the binary representation of ~~inde~~ index



We keep doing it till (index > 0)

Now that we have a solid understanding of sum, Lets look at update.

1 2 3 . . . (index) . . . . . N

Lets suppose you are updating (index, +val)

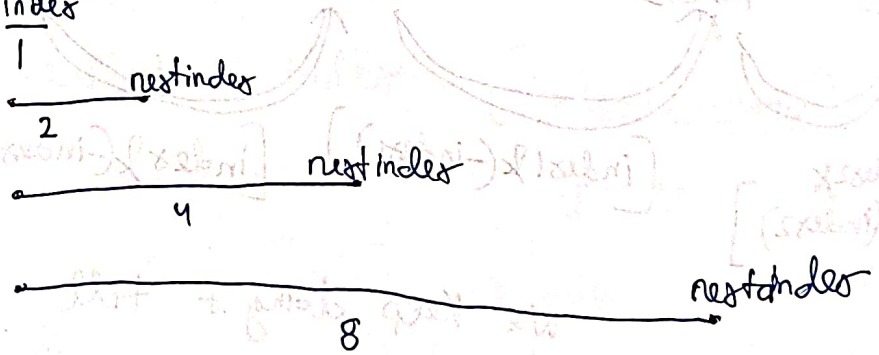
we have to wonder now if I add val to BIT[index] how is it affecting my BIT tree and which indexes to be exact is it affecting

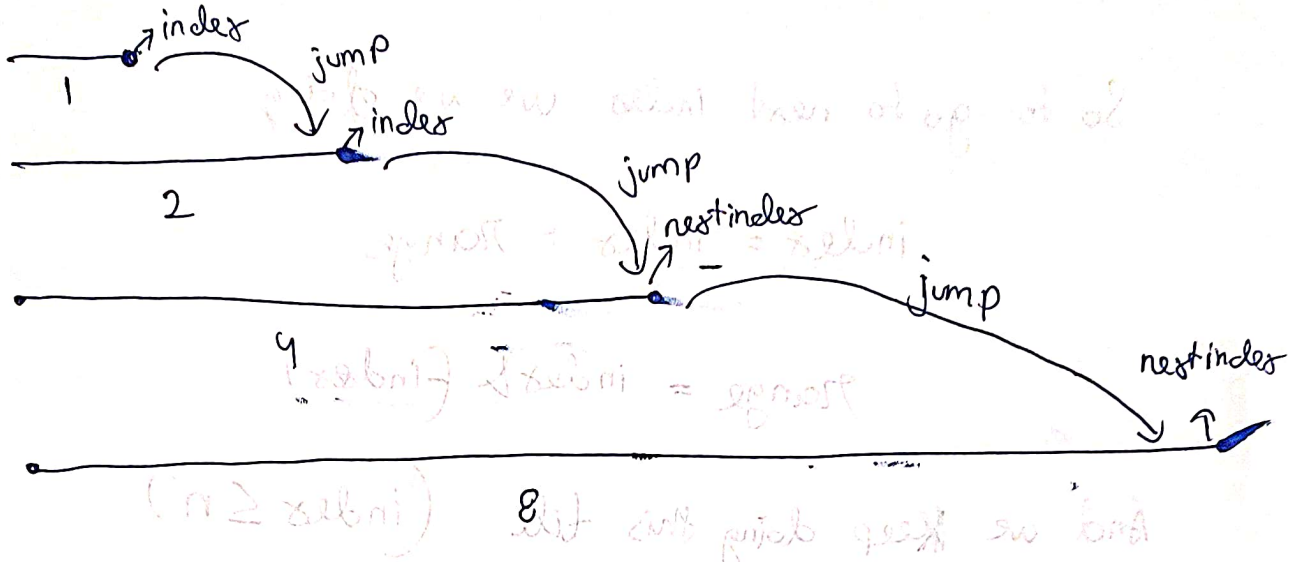
to understand update first you need to understand range

range = index & (-index) ⟹ this is always a single bit.

Single bit meaning, this will be always a powers of 2.

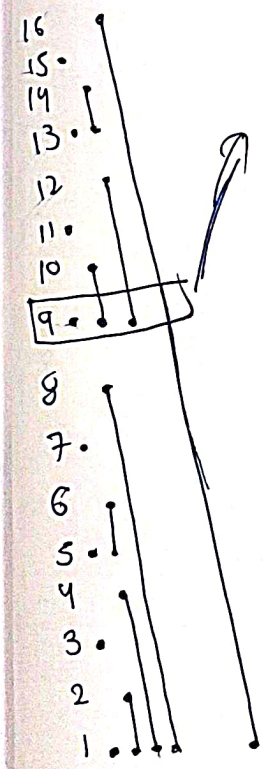
ranges will be like





updating arr[index] will update consecutively all the ranges next. It will affect the range (1) which is index itself; it will affect range (2). (index, index+1), it will affect (index, index+1, index+2, index+3) → range (4). and so on.

↗ Lets consider Array of size 16.. Range always power of 2



if i do an update on arr[9] then notice how it affects. arr[9] → arr[10] → arr[12] → arr[16].

(range 1) (range 2) (range 4) (range 8)

index nextindex1 nextindex2 nextindex3.

range = index & (-index)

[nextindex1] = index + range → (index) & (-index)

[nextindex2] = nextindex1 + range → (nextindex1) & (-nextindex).

So to go to next index we are doing

index = index + range

range = index & (-index)

And we keep doing this till (index ≤ n)

```
void update(int index, int val){

    while (index ≤ n){
        BIT[index] += val;
        index = index + (index & (-index))
    }
}
```

Next time you forget intuition behind sum and update of Fenwick tree just go through this pdf.