

Fenwick Tree

1	3	4	1	2
0	1	2	3	4

→ Sum b/w 0 & 3?

- Either loop from 0 to 3
- or
- Use prefix sum

→ What if some update operation is also given.

This will require us to update the value of prefix sum which will take $O(n)$. So no need of prefix sum.

Need to find another solution.

Update operation :-

1	0	2	1	1	3	0	4	2	5	2	2	3	1	0	2
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

1	0	1	2	0	0	0	0	1	0	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

↑
3
 $(1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (1,8) (1,9) (1,10) (1,11) (1,12) \dots$

Note: Any no. can be written in the power of 2.

Ex: $11 = 8 + 2 + 1$.

$(1,8)$ at the 0th index means that it stores the sum from index 1 to 8.

* Step to find the range:-

- Make the rightmost set bit 0.
- Add 1 to it.
- The range becomes (x, y)
 \downarrow
 original no.
 the no. obtained in 2nd step.

* Now getting the sum:-

- Now, first updating the value at index 1.
- We will add 1 to all the indexes which stores the sum of index 1.

Eg: $(1,1) (1,2) (1,4) (1,8) (1,16)$

- Now, after updating the value at index 1, we need to know which next index is needed to be updated.

- 2's complement
- i with the original no.
- Add to the original no.

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

Prefix sum :-

$(1,13)$

- Take 13 ($13 - 13$)
- Now make the rightmost bit 0.

$$1101 \rightarrow 1100$$

$\searrow 12$

- The range is $(9-12)$
- Take 12 & make rightmost bit 0.
 $1100 \rightarrow 1000(8)$
- The range is $(1-8)$

$$(1,8) = (13,13) + (9,12) + (1,8)$$

- $1000 \rightarrow 0000$ (break)

Q8

- 2's complement
- & with original no.
- - with original no,

$$\left. \begin{array}{l} \\ \\ \end{array} \right\} i = i - (i \& (-i))$$

```
int fen[N];
void update(int i, int add) {
    while(i < N) {
        fen[i] += add;
        i += (i & (-1));
    }
}
int sum(int i) {
    int s = 0;
    while(i > 0) {
        s += fen[i];
        i = i - (i & (-i));
    }
    return s;
}
int rangeSum(int l, int r) {
    return sum(r) - sum(l - 1);
}
```

	Time Complexity	Space Complexity	Update	Find
	$O(\log N)$	$O(N)$	$O(\log N)$	$O(\log N)$

Binary Lifting :-

1	0	2	1	1	3	0	4	2	5	2	2	3	1	0	2
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

1	1	2	4	1	4	0	12	2	7	2	11	3	4	0	29
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

(1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (1,8) (1,9) (1,10) (1,11) (1,12) (1,13) (1,14) (1,15) (1,16)

- Initially, the pointer is at 0.
- Any no. can be written in the power of 2.
 $\text{Ex: } 11 = 8 + 2 + 1.$
- Whatever is the size of array take the highest power of 2.

$$16 \Rightarrow 2^4$$

- If the size of array was 20 then also 2^4 .

- Now, sum at 16 is $29 > 11$

- Reduce the value of power, i.e 2^3 is $12 > 11$

- $2^2 = 4$

$$\text{sum}[4] = 4 < 11$$

- Add 2^1 $4 + 2 = 6$

$$\text{sum}[6] = 6$$

$$8 < 11$$

→ Find lower_bound of 11?

- If somehow I am able to find the right most index whose prefix sum is < 11 .
- We can simply add 1, and the index which we get will either be ≥ 11 .

- $2^0 = 1 \rightarrow 6 + 1 = 7$

$$\text{sum}[7] = 0$$

$$8 < 11$$

- $2^1 \rightarrow \text{stop}$

So Ans is 8

→ Find lower_bound of 27.

1	0	2	1	1	3	0	4	25	2	2	3	1	0	2	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

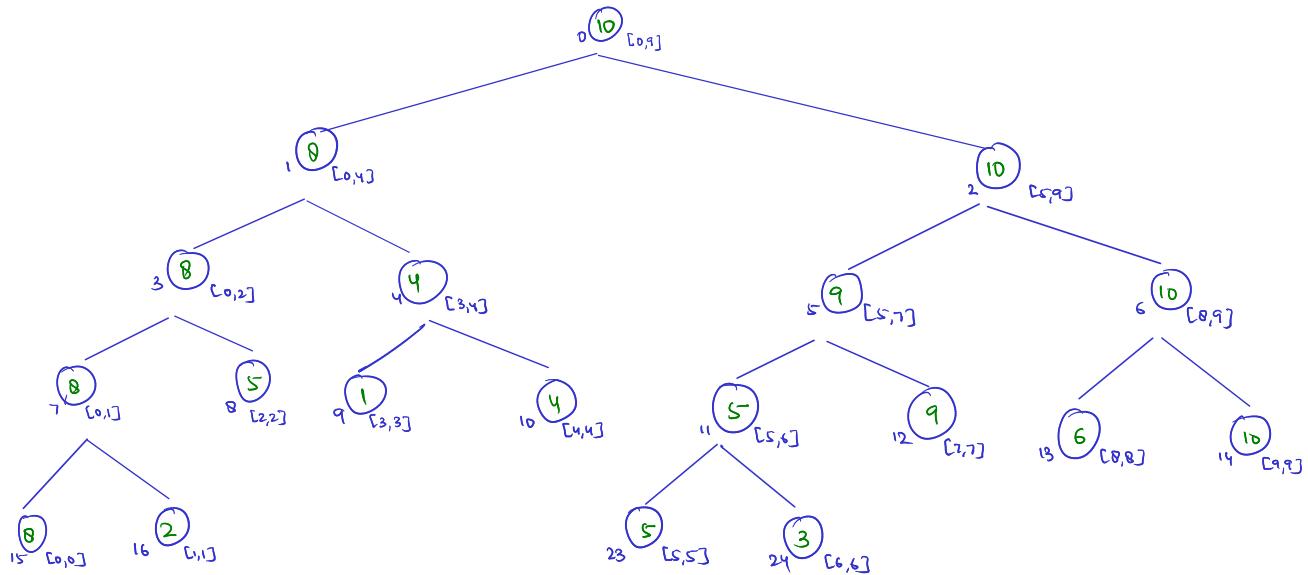
1	1	2	4	1	4	0	12	2	7	2	11	3	4	0	29
(1,1)	(1,2)	(2,3)	(3,4)	(1,4)	(5,6)	(5,6)	(7,8)	(1,8)	(9,9)	(9,10)	(10,11)	(9,12)	(13,13)	(11,15)	(11,16)

```
int find(int k) {
    int curr = 0, ans = 0, prevsum = 0;
    for(int i = log2(n); i >= 0; i--) { l=20 (for being safe, 20 because 2^20 exceeds 10^6)
        if(ft[curr + (1<<i)] + prevsum < k) {
            curr = curr + (1<<i);
            prevsum += ft[curr];
        }
    }
    return (curr + 1);
}
```

Segment Tree

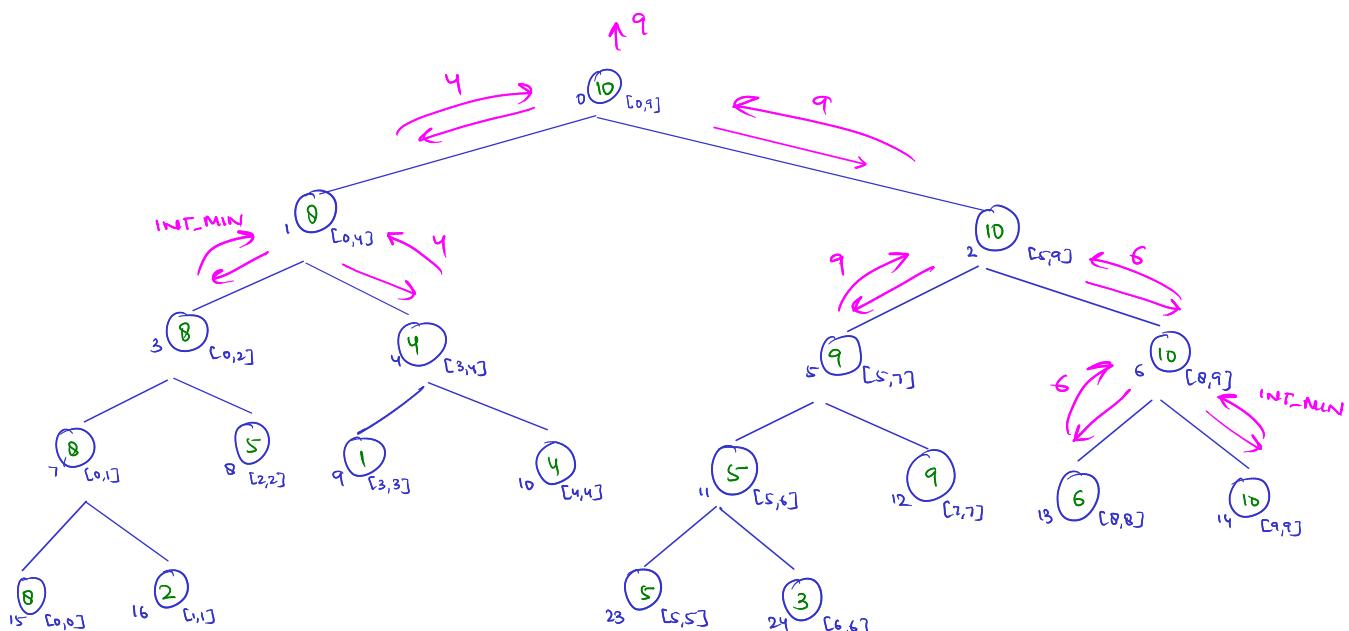
$$a[] = \{8, 2, 5, 1, 4, 5, 3, 9, 6, 10\} \quad n=10$$

→ Find the max value in the range of $[3, 8] \rightarrow 9$



→ For 3-8 there is no such exact range. So we have 3 cases:-

- Lies in the range (return the node value)
- Doesn't lies (return INT_MIN)
- Overlaps (go to left & right)



Time Complexity	Build	Query
Space Complexity	$O(n)$	$O(\log n)$

```

int a[100005], seg[4 * 100005];

void build(int ind, int low, int high) {
    if(low == high) {
        seg[ind] = a[low];
        return;
    }
    int mid = (low + high) / 2;
    build(2 * ind + 1, low, mid);
    build(2 * ind + 2, mid + 1, high);
    seg[ind] = max(seg[2 * ind + 1], seg[2 * ind + 2]);
}

int query(int ind, int low, int high, int l, int r) {
    if(low >= l && high <= r) {
        return seg[ind];
    }
    if(high < l || low < r) return INT_MIN;
    int mid = (low + high) / 2;
    int left = query(2 * ind + 1, low, mid, l, r);
    int right = query(2 * ind + 2, mid + 1, high, l, r);
    return max(left, right);
}

int main() {
    int n;
    cin>>n;
    for(int i = 0; i < n; i++) {
        cin>>a[i];
    }
    build(0, 0, n - 1);
    int q;
    cin>>q;
    while(q--) {
        int l, r;
        cin>>l>>r;
        cout<<query(0, 0, n - 1, l, r);
    }
    return 0;
}

```