Fenwick Tree / Binary Indexed Tree

Joy Mukherjee

IIT Bhubaneswar

Use: Range Query

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

- Prefix Sum(A, i) ($0 \leq i < n$)
- = A[0] + A[1] + … + A[i]
- Range Sum(A, i, j) ($-1 < i \leq j < n$)
- = A[i] + A[i+1] + … + A[j]
- Find prefix sum in an array for an index
  - What is the prefix sum for 2?
  - 6
  - What is the prefix sum for 4?
  - 8
- Given a range [start, end], find the sum in an array in the given range?
  - What is the range sum from 2 to 4 in the array?
  - 6
  - What is the range sum from 0 to 3 in the array?
  - 6

Problem

-1	**3**	**4**	**0**	**2**	**1**
0	1	2	3	4	5

- Query:
- 1 Prefix Sum(a, i) /
- 2 Range Sum(A, i, j)
- Given an array A of n integers, and q queries, what is the time complexity to answer all the q queries?
- Time Complexity =O(qn)

- Input:
- n = 6
- -1 3 4 0 2 1
- q = 4
- 1 2
- 2 3 5
- 1 3
- 2 1 4

- Output:
- 6
- 3
- 6
- 9

Solution

1. Maintain an 1-D array R, where R[i] stores the sum from A[0] to A[i]
2. Each query takes O(1) time, if A[] is not updated
   - Time to prepare R[] is O(n)
   - Space complexity O(n)
   - If A[] is updated frequently, then R[] need to be reconstructed every time for each update.

A	**-1**	**3**	**4**	**0**	**2**	**1**
	0	1	2	3	4	5

R	**-1**	**2**	**6**	**6**	**8**	**9**
	0	1	2	3	4	5

Solution

-1	**3**	**4**	**0**	**2**	**1**
0	1	2	3	4	5

- Query:
- 1 Prefix Sum(A, i) /
- 2 Range Sum(A, i, j)
- Given an array A of n integers, and q queries, what is the time complexity to answer all the q queries?
- RangeSum(a, i, j) =
- R[j]-R[i-1] if $i \geq 1$
- R[j]
- Time Complexity =O(q + n)

- Input:
- n = 6
- -1 3 4 0 2 1
- q = 4
- 1 2
- 2 3 5
- 1 3
- 2 1 4

- Output:
- 6
- 3
- 6
- 9

R	**-1**	**2**	**6**	**6**	**8**	**9**
	0	1	2	3	4	5

Problem

		8		-4		
A	**-1**	**3**	**4**	**0**	**2**	**1**
	0	1	2	3	4	5

- Query:
- 1 Prefix Sum(A, i) /
- 2 Range Sum(A, i, j)/
- 3 Update(A, i, x)
- Given an array A of n integers, and q queries, what is the time complexity to answer all the q queries?

- Input:
- n = 6
- -1 3 4 0 2 1
- q = 5
- 1 2
- 2 3 5
- 3 1 8
- 1 3
- 3 3 -4
- 2 1 4

- Output:
- 6
- 3
- 11
- 10

		7	**11**	**11**	**13**	**14**
R	**-1**	**2**	**6**	**6**	**8**	**9**
	0	1	2	3	4	5
				7	**9**	**10**

Issues

- q queries
- No of update queries = q/2 O(qn)   Every time we have to update R[]
- No of range queries = q/2 O(q)
- Every update to A[i] asks for the update in R[i], R[i+1], …, R[n-1]
- Time complexity = O(qn)

Benefit: FT/BIT

- O(n) space is required to maintain the Fenwick tree
- O(nlogn) time to construct the Fenwick tree
- Each search/update takes O(log n) time
- To execute q queries, O(nlogn + qlog n) time is needed.

Two’s Complement (x)

- Given an integer x
- p = Binary(x)
- If x < 0 then Prepend 1 to p
- Else Prepend 0 to p
- q = 1’s Complement of p
- (Flip the bits 0 -> 1 and 1-> 0)
- r = 2’s Complement of x
- = 1’s complement of p + 1

- 11
- p = 01011
- q = 10100
- r = q + 1 = 10100 + 1 = 10101 (Ans)

Get Parent in FT/BIT

- A[n]: A[0] to A[n-1]
- FT[n+1]
- For each index i from 1 to n, find out i's parent index.
- Parent(1011) = 1010
- Parent(1010) = 1000
- Parent(1000) = 0000
- To find the parent, we reset the rightmost set bit.
- BIT[13] = 1101 = [12, 12] = A[12]
- BIT[14] = 1110 = [12, 13] = A[12]+A[13]
- BIT[28] = 11100 = [24, 27] = A[24]+…+A[27]

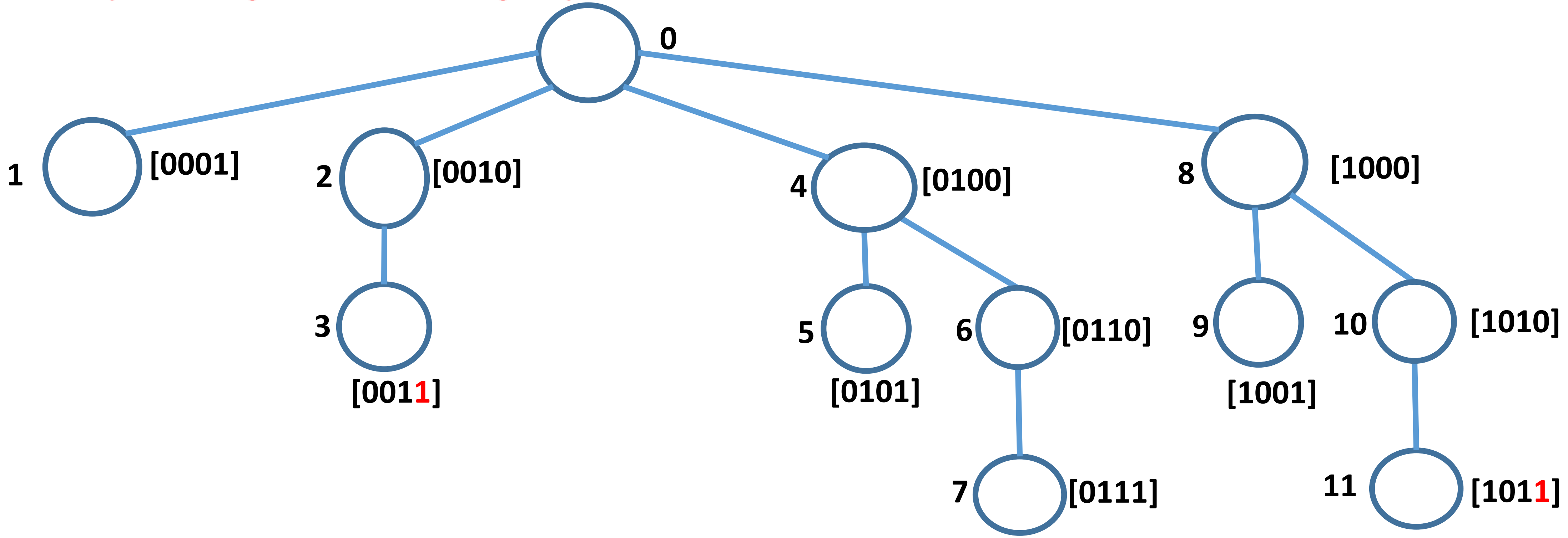
Get Parent in FT/BIT

Get Parent (x) 1010

1. Two's Complement of x   0101 + 1 = 0110
2. AND with x   0010
3. Subtract from x   1010 – 0010
4. P = x – (x & (-x))   1000

Size of BIT = Size of original array + 1

Flip the rightmost 1 to get parent



What each node of FT/BIT contains?

A	**3**	**2**	**-1**	**6**	**5**	**4**	**-3**	**3**	**7**	**2**	**3**	
	0	1	2	3	4	5	6	7	8	9	10	
BIT	**0**	**3**	**5**	**-1**	**10**	**5**	**9**	**-3**	**19**	**7**	**9**	**3**
	0	1	2	3	4	5	6	7	8	9	10	11

BIT		Start Index of A	End Index of A	Sum
0	0			0
1	$0 + 2^0$	0	0	3
2	$0 + 2^1$	0	1	5
3	$2^1 + 2^0$	2	2	-1
4	$0 + 2^2$	0	3	10
5	$2^2 + 2^0$	4	4	5

BIT		Start Index of A	End Index of A	Sum
6	$2^2 + 2^1$	4	5	9
7	$2^2 + 2^1 + 2^0$	6	6	-3
8	$0 + 2^3$	0	7	19
9	$2^3 + 2^0$	8	8	7
10	$2^3 + 2^1$	8	9	9
11	$2^3 + 2^1 + 2^0$	10	10	3

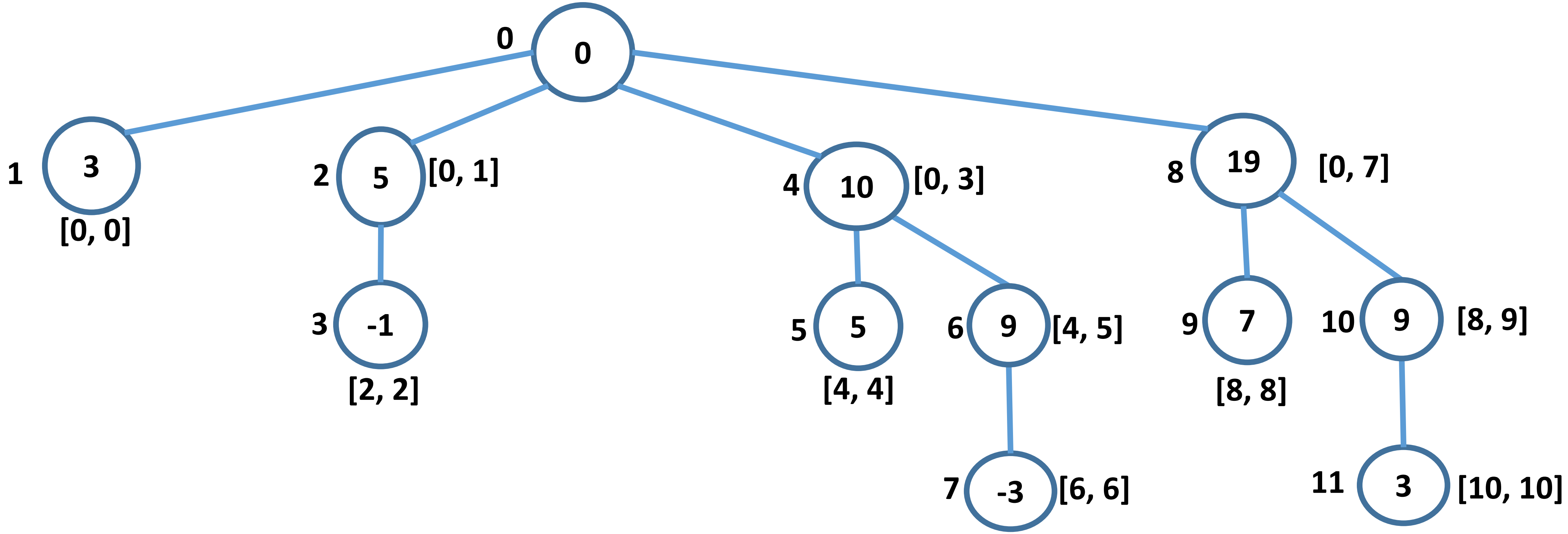
How to decide the start index of BIT[x]?

- If x is an integer in the form 2^k , then the start index is 0.
  - $x = 0 + 2^k$
  - BIT[x] = Summation of A[0] to A[$2^k - 1$]

- Otherwise x can be written as $2^k + 2^m + 2^p$ such that $k > m > p$
  - BIT[x] = Summation of A[$2^k + 2^m$] to A[$2^k + 2^m + 2^p - 1$]

Fenwick Tree

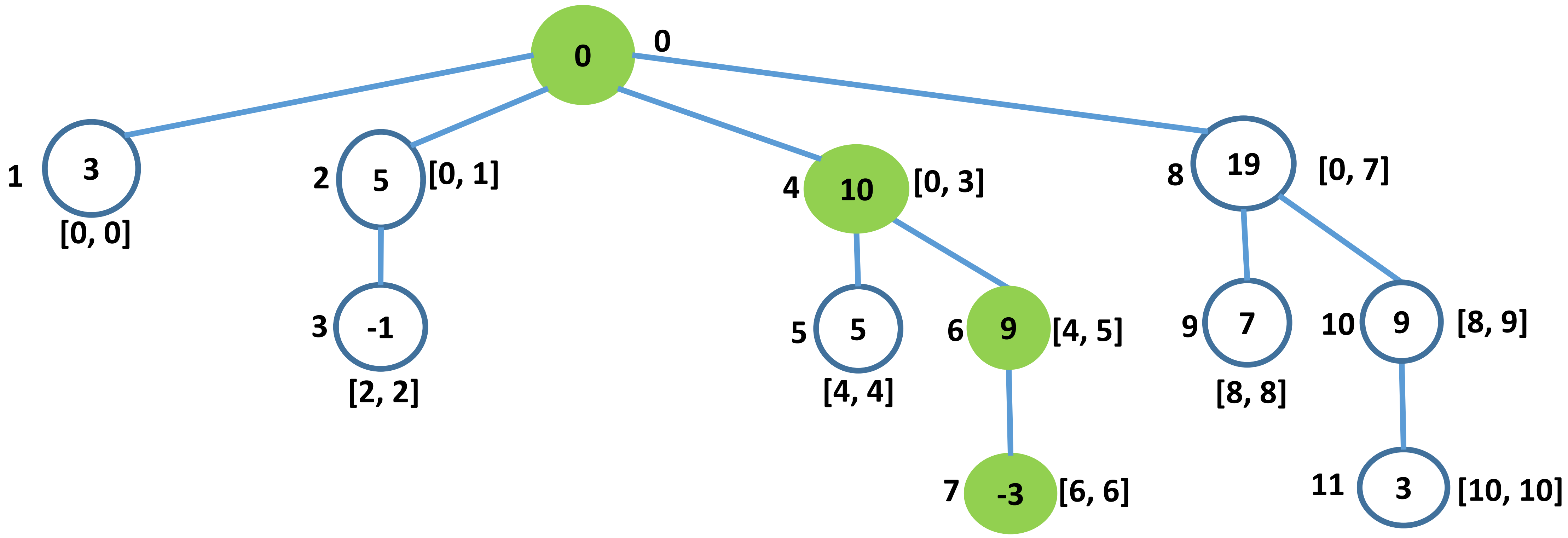
3	**2**	**-1**	**6**	**5**	**4**	**-3**	**3**	**7**	**2**	**3**	
0	1	2	3	4	5	6	7	8	9	10	
0	**3**	**5**	**-1**	**10**	**5**	**9**	**-3**	**19**	**7**	**9**	**3**
0	1	2	3	4	5	6	7	8	9	10	11



Prefix sum(A, 6)

Go to index 7 and sum up all the integers from the node up to the root (-3+9+10+0=16)

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



Prefix sum(BIT[], x)

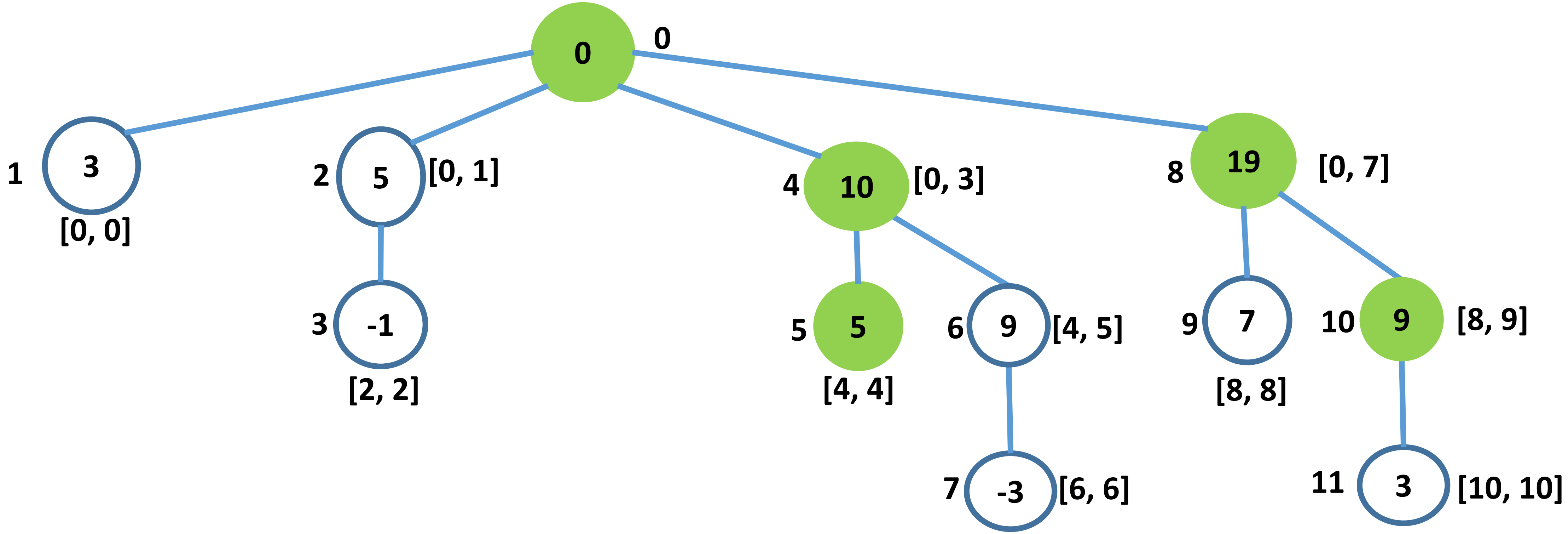
```
int PrefixSum(BIT[], x) {
        int sum = 0;
        x++;
        while(x > 0) {
                sum += BIT[x];
                x      -= (x & (-x));
        }
        return sum;
}
```

Range Sum(A, 5, 9) = Prefix sum(A, 9) - Prefix sum(A, 4)

Go to index 10 and sum up all the integers from the node up to the root

- Go to index 5 and sum up all the integers from the node up to the root (28 – 15 = 13)

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



Range sum(BIT[], x, y)

```
int RangeSum(BIT[], x, y) {
        if(x > 0)
                return PrefixSum(BIT, y) - PrefixSum(BIT, x-1);
        return PrefixSum(BIT, y);
}
```

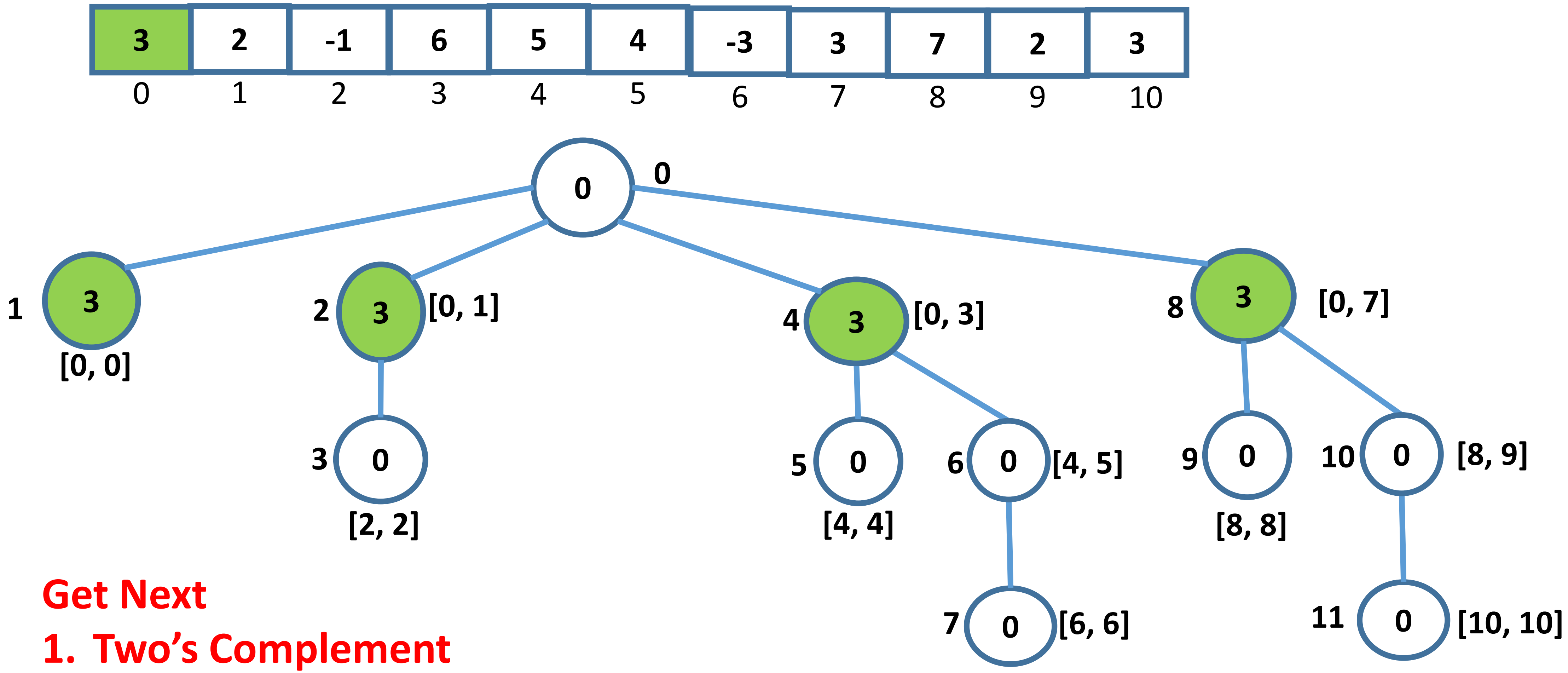
Get Next (x)

1. Two's Complement (x)
2. AND with x
3. ADD to x

$N = x + (x \& (-x))$

x	Binary	2's Complement	AND	ADD	Next(x)
1	00001	11111	00001	00010	2
2	00010	11110	00010	00100	4
3	00011	11101	00001	00100	4
4	00100	11100	00100	01000	8
5	00101	11011	00001	00110	6
6	00110	11010	00010	01000	8
7	00111	11001	00001	01000	8
8	01000	11000	01000	10000	16
9	01001	10111	00001	01010	10
10	01010	10110	00010	01100	12
11	01011	10101	00001	01100	12

Construct Fenwick Tree

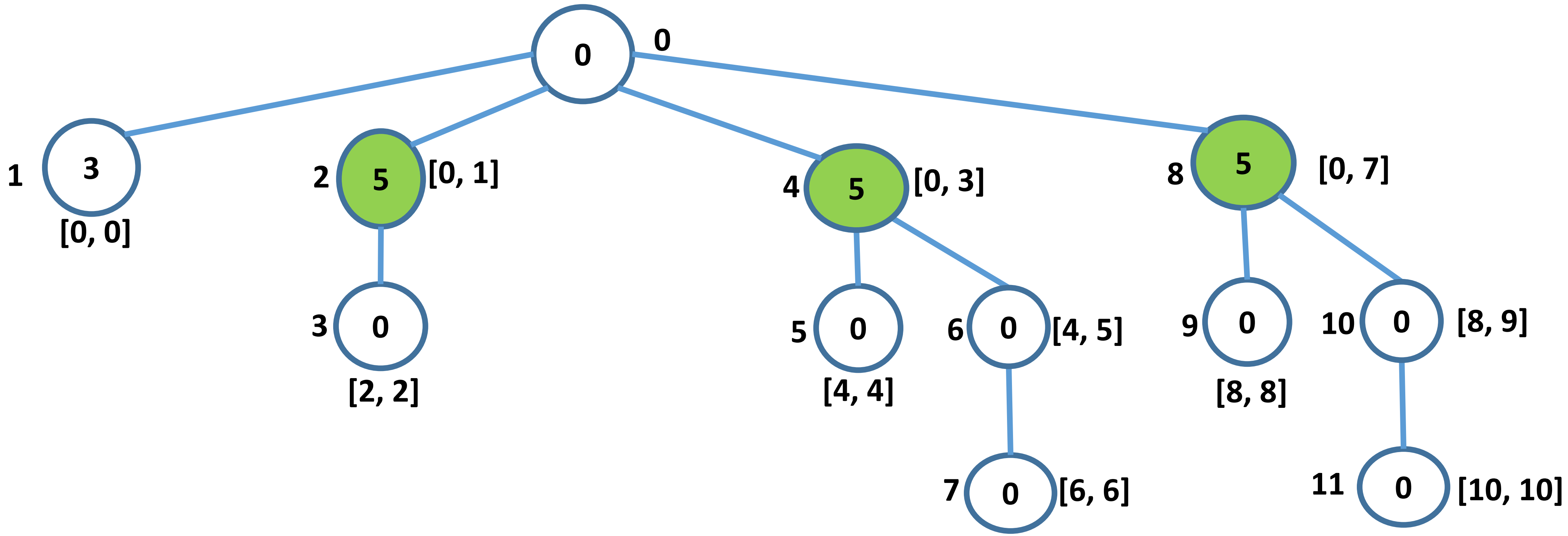


Get Next

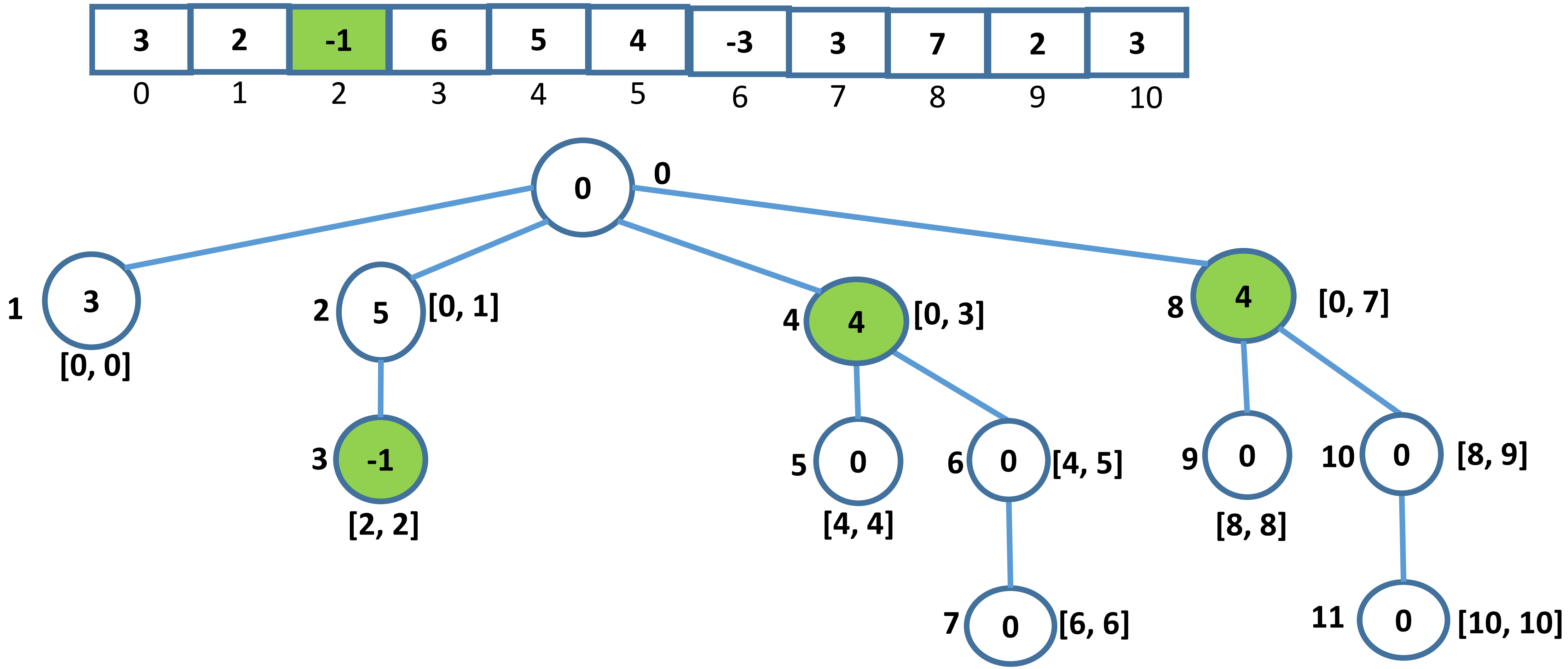
1. **Two's Complement**
2. **AND with original number**
3. **ADD to original number**

Construct Fenwick Tree

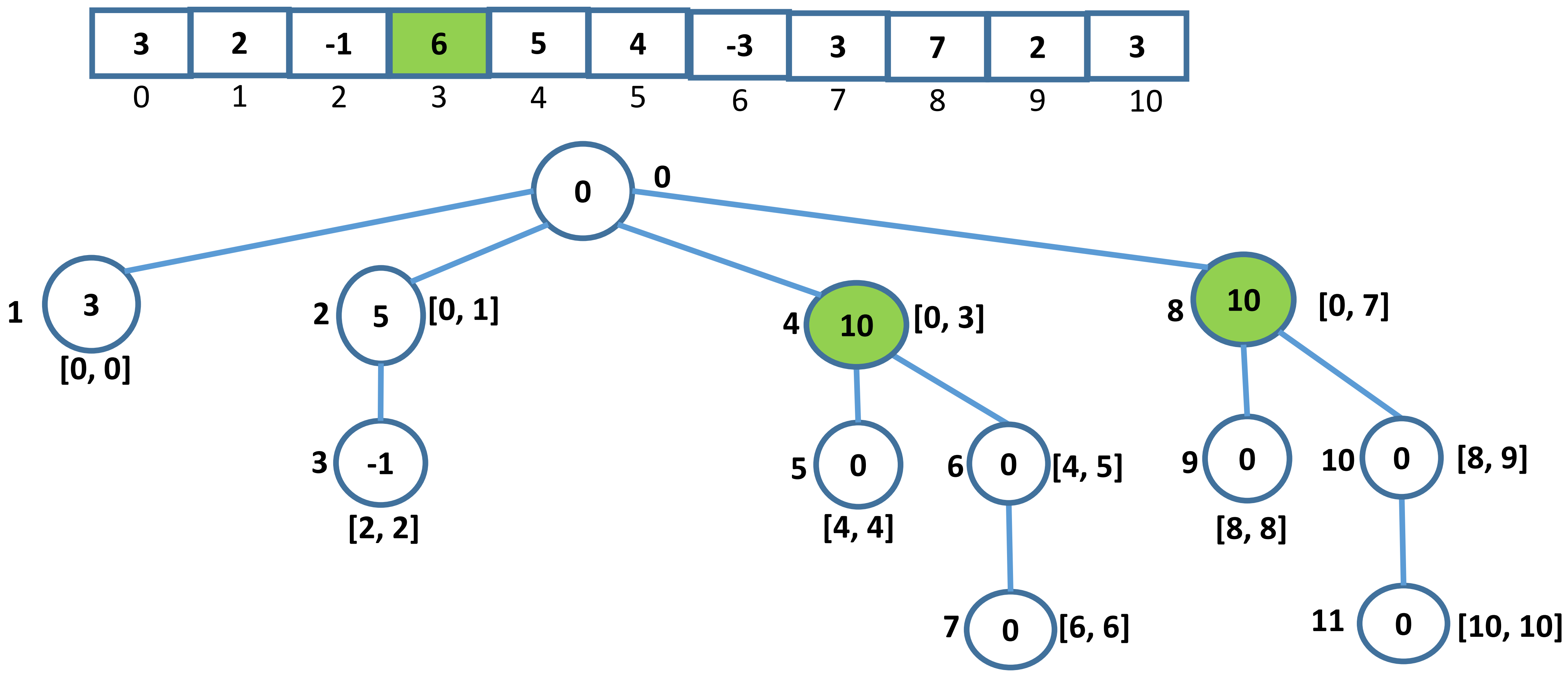
0	1	2	3	4	5	6	7	8	9	10
3	2	-1	6	5	4	-3	3	7	2	3



Construct Fenwick Tree

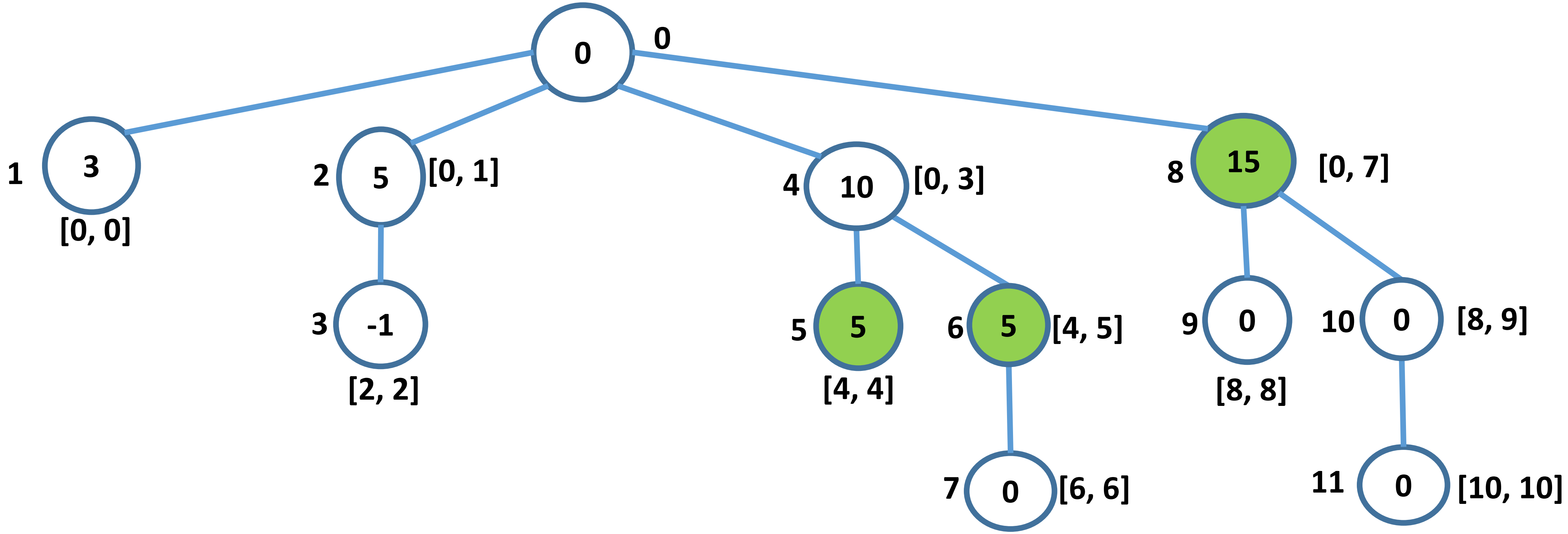


Construct Fenwick Tree

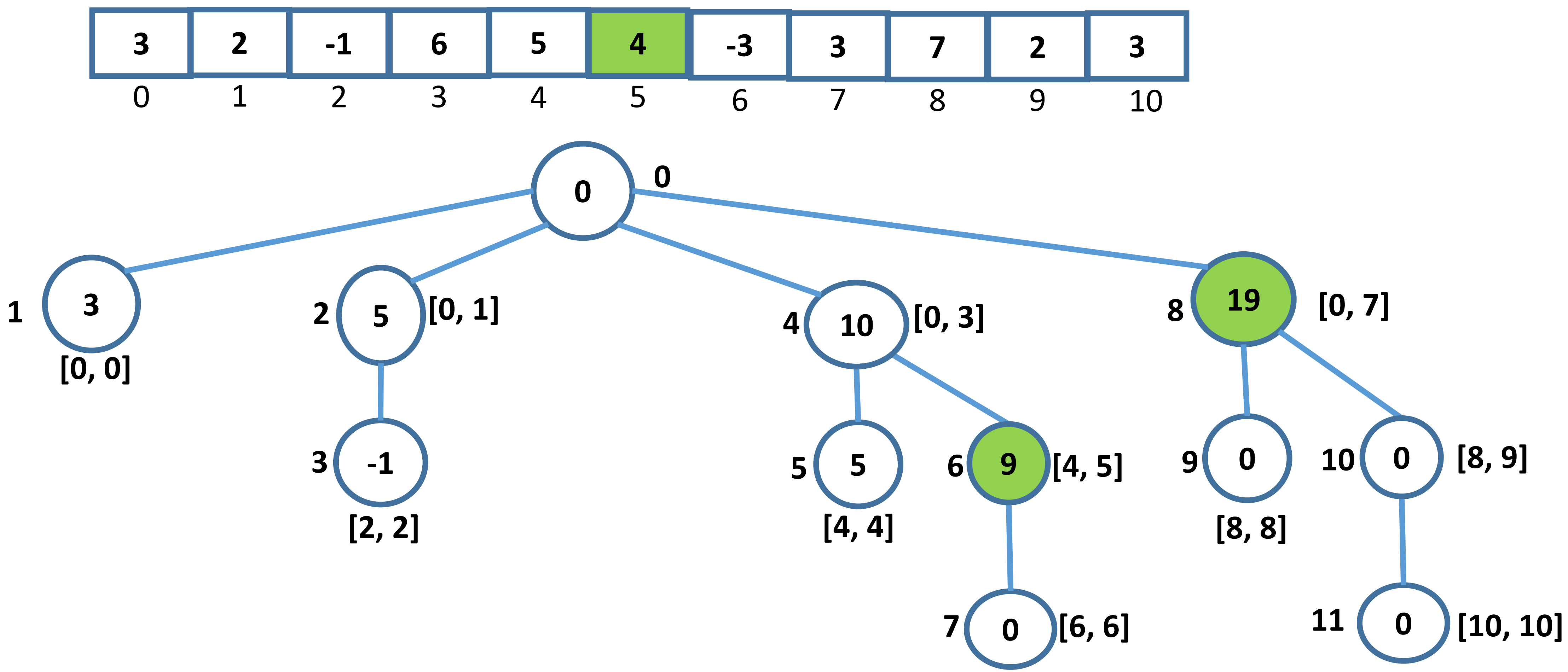


Construct Fenwick Tree

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

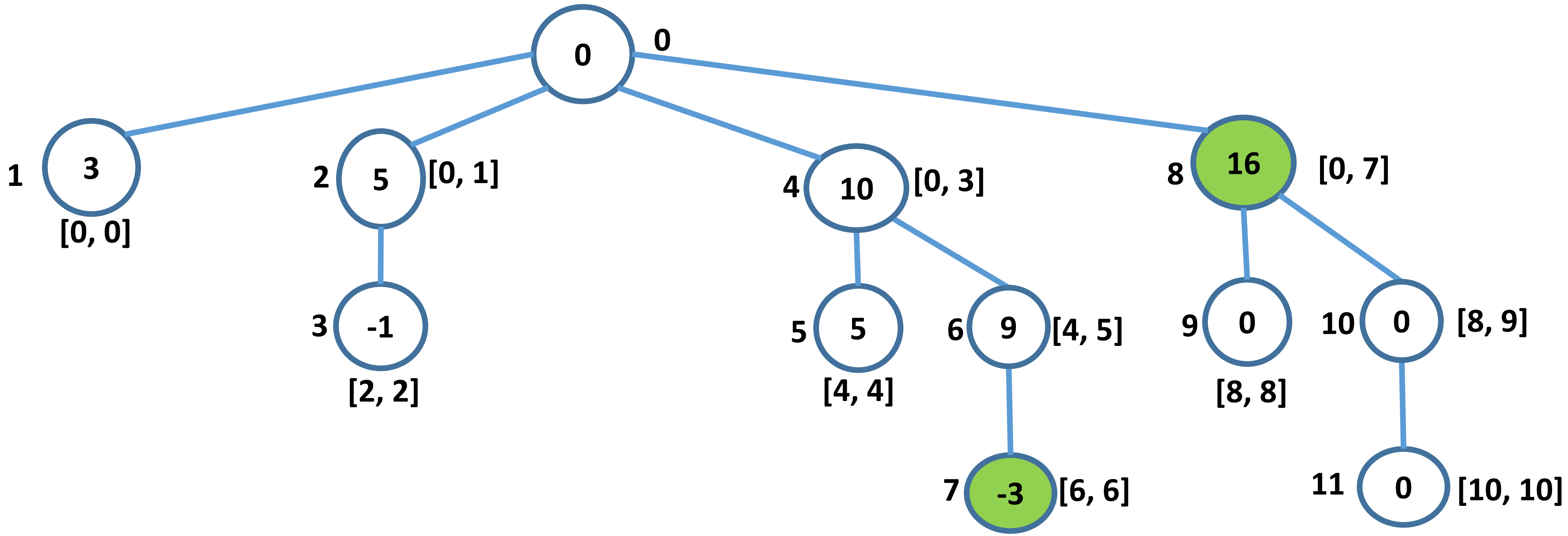


Construct Fenwick Tree

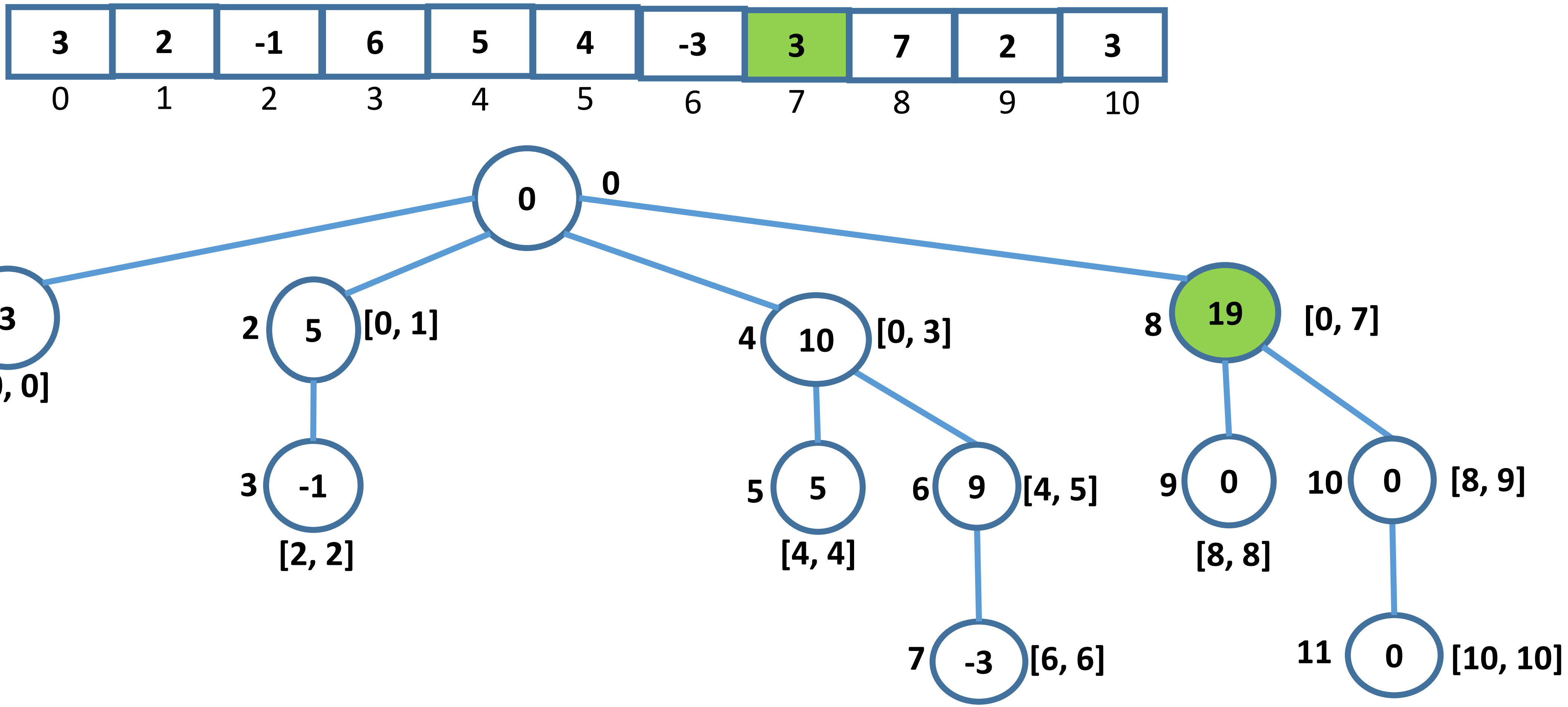


Construct Fenwick Tree

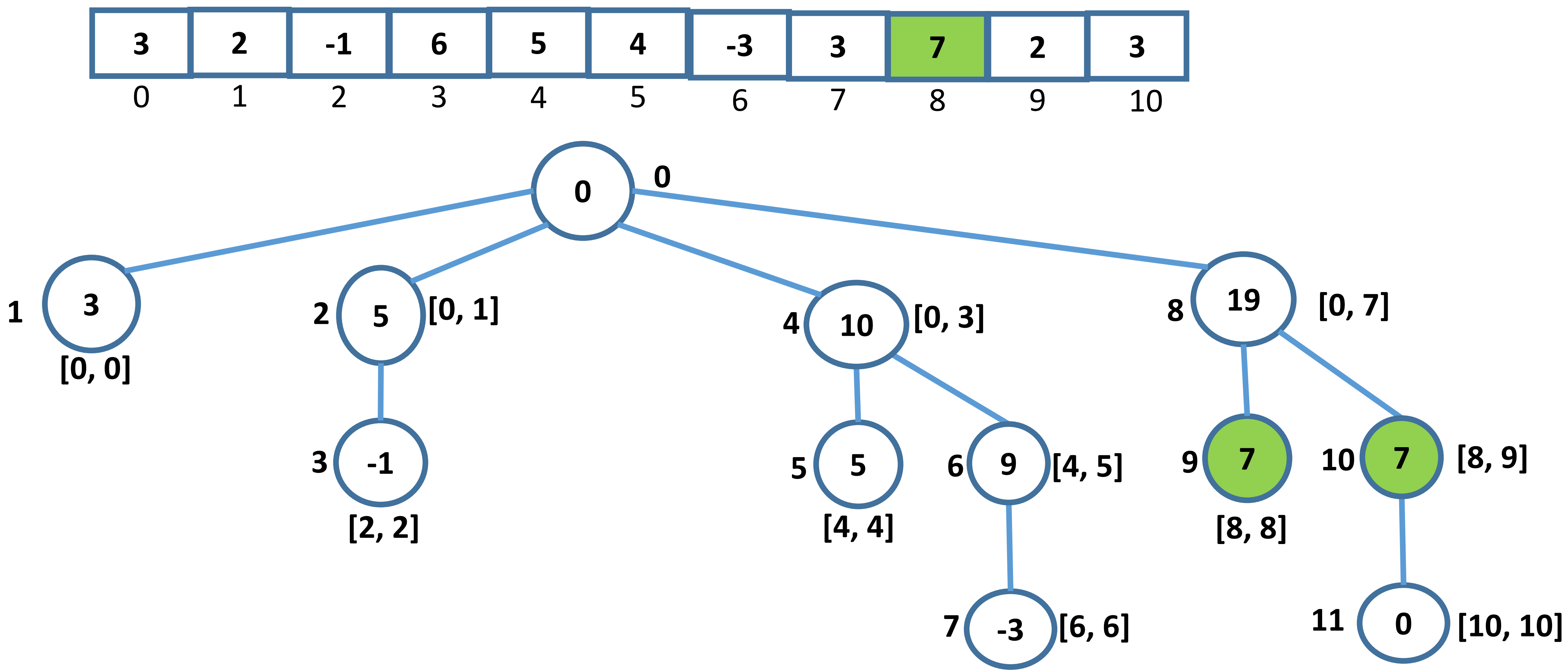
3	2	-1	6	5	4	-3	3	7	2	3
0	1	2	3	4	5	6	7	8	9	10



Construct Fenwick Tree

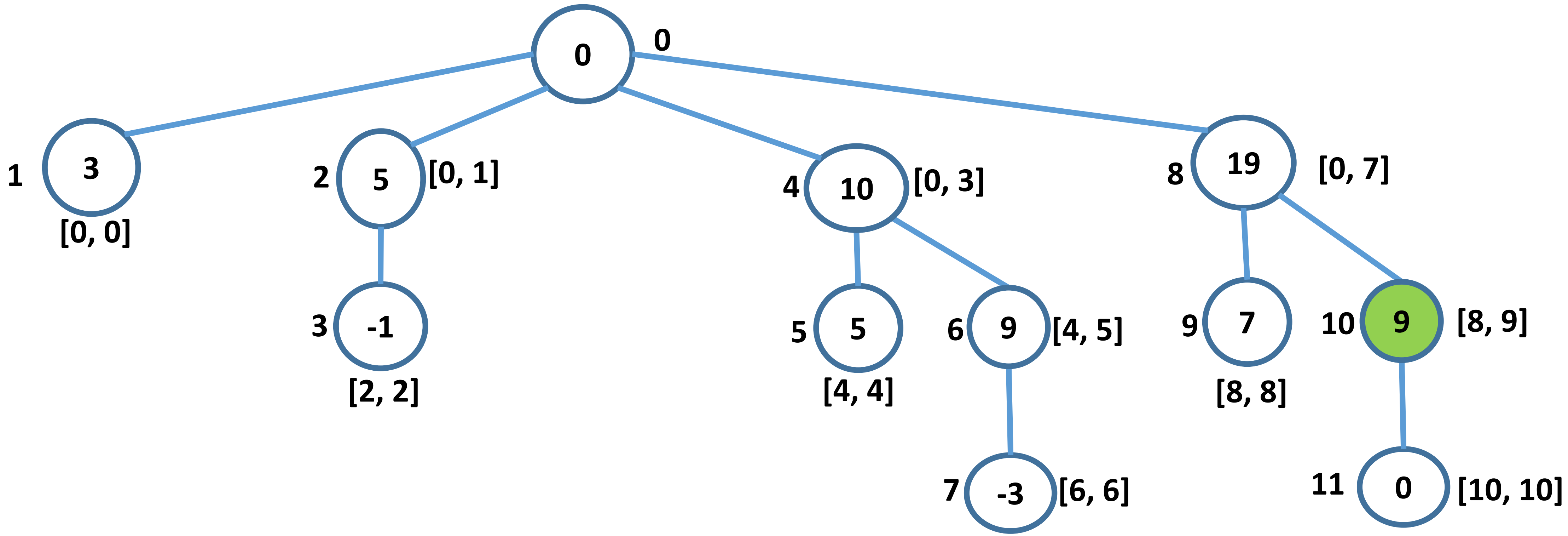


Construct Fenwick Tree

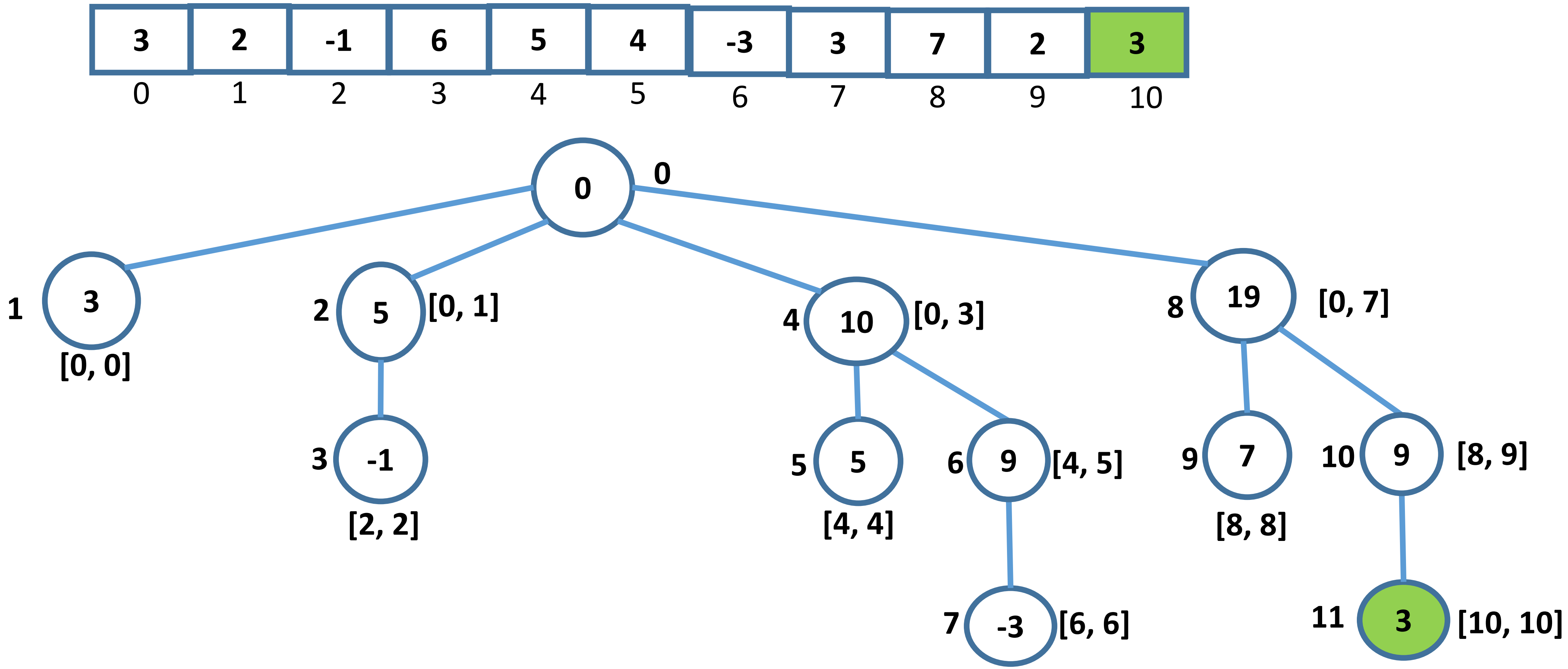


Construct Fenwick Tree

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



Construct Fenwick Tree

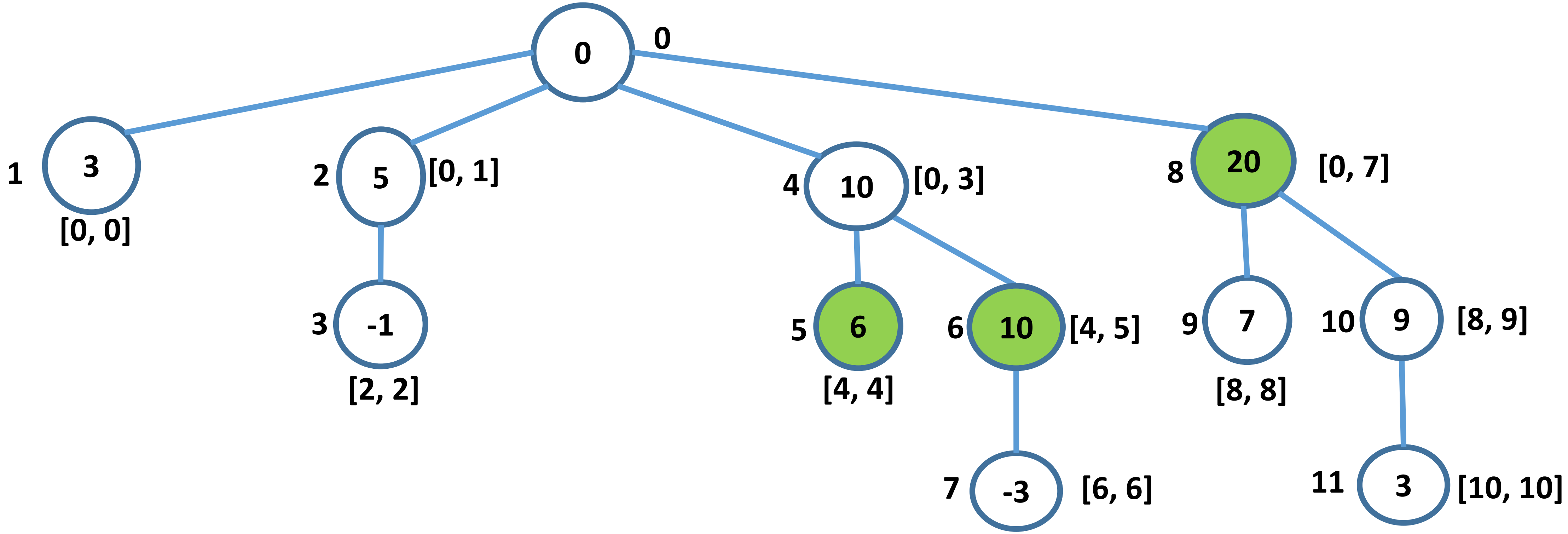


ConstructFenwickTree(n, A[], BIT[])

```
ConstructFenwickTree(n, A[], BIT[]) {    // n is size of A
        int i;
        for(i = 0; i <= n; i++)
                BIT[i] = 0;
        for(i = 0; i < n; i++) {
                int x = i+1;
                while(x <= n) {
                        BIT[x] += A[i];
                        x      += (x & (-x));
                }
        }
}
```

Update Fenwick Tree

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



Update(n, A[], BIT[], i, p)

```
Update(n, A[], BIT[], i, p) {    // Set A[i] = p
        int x = i+1;
        int netChange = p - A[i];
        while(x <= n) {
                BIT[x] += netChange;
                x      += (x & (-x));
        }
        A[i] = p;
}
```