Dynamic Range Minimum Queries

Inge Li Gørtz

Range Minimum Queries

- Range minimum query problem. Preprocess array A[1…n] of integers to support
  - RMQ(i,j): return the (entry of) minimum element in A[i…j].

0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	1	4	8	3

  - RMQ(2,5) = ?

Range Minimum Queries

- Range minimum query problem. Preprocess array A[1…n] of integers to support
  - RMQ(i,j): return the (entry of) minimum element in A[i…j].

0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	1	4	8	3

  - RMQ(2,5) = 2

- Solution 1. Store the array. Given a query run through array.
  - Space O(n)
  - Time O(n)

Range Minimum Queries

- Range minimum query problem. Preprocess array A[1…n] of integers to support
  - RMQ(i,j): return the (entry of) minimum element in A[i…j].

0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	1	4	8	3

  - RMQ(2,5) = 2

- Solution 2. Store a matrix with answer to all possible queries.
  - Space O(n²)
  - Time O(1)

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1		7	7	7	2	2	1	1	1	1
2			12	8	2	2	1	1	1	1
3				8	2	2	1	1	1	1
4					2	2	1	1	1	1
5						5	1	1	1	1
6							1	1	1	1
7								4	4	3
8									8	3
9										3

Dynamic Range Minimum Queries

- Dynamic Range minimum query problem. Preprocess array A[1…n] of integers to support
  - RMQ(i,j): return the (entry of) minimum element in A[i…j].
  - Add(i,k): Set A[i] = A[i] + k   (k can be negative).

0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	1	4	8	3

Add(6,2) →

0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	3	4	8	3

Dynamic Range Minimum Queries

- Dynamic Range minimum query problem. Preprocess array A[1…n] of integers to support
  - RMQ(i,j): return the (entry of) minimum element in A[i…j].
  - Add(i,k): Set A[i] = A[i] + k   (k can be negative).

0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	1	4	8	3

Add(6,2) →

0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	3	4	8	3

- Solution 1. Store the array. Given a query run through array.
  - Space O(n)
  - Query time O(n)
  - Update time O(1)

Dynamic Range Minimum Queries

- Dynamic Range minimum query problem. Preprocess array A[1…n] of integers to support
  - RMQ(i,j): return the (entry of) minimum element in A[i…j].
  - Add(i,k): Set A[i] = A[i] + k   (k can be negative).

0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	1	4	8	3

Add(6,2) →

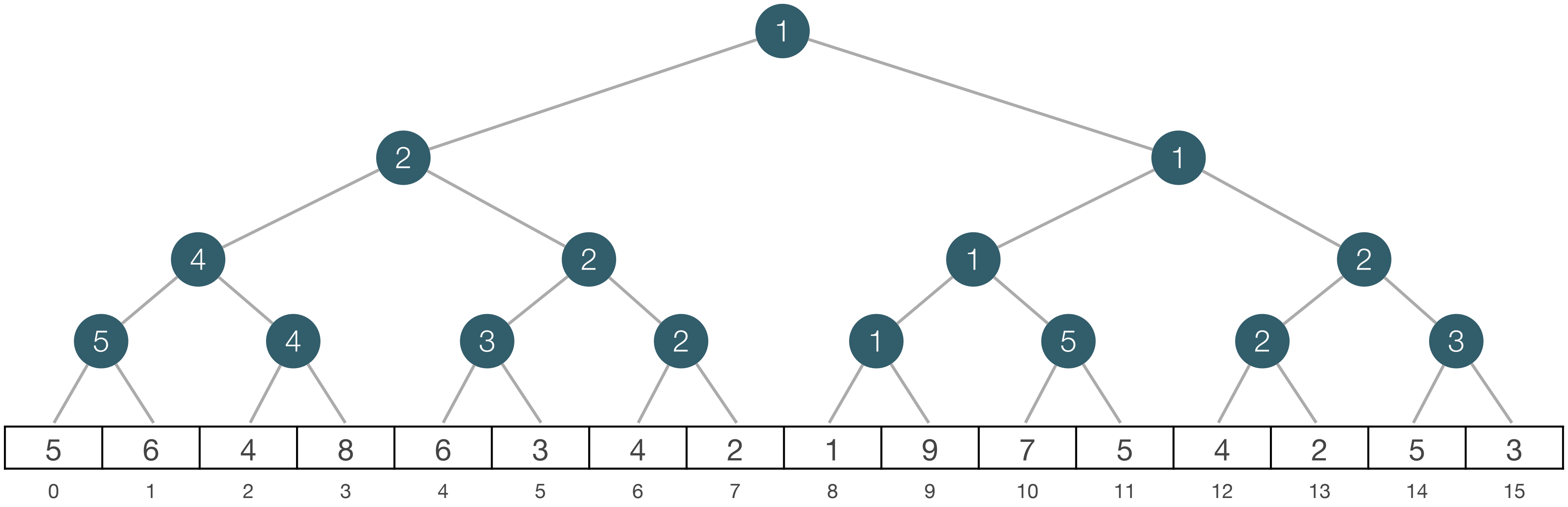
0	1	2	3	4	5	6	7	8	9
1	7	12	8	2	5	3	4	8	3

- Solution 2. Store a matrix with answer to all possible queries.
  - Space $O(n^2)$
  - Query time O(1)
  - Update time $O(n^2)$

	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1		7	7	7	2	2	1	1	1	1
2			12	8	2	2	1	1	1	1
3				8	2	2	1	1	1	1
4					2	2	1	1	1	1
5						5	1	1	1	1
6							1	1	1	1
7								4	4	3
8									8	3
9										3

Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k (k can be negative).
  - RMQ(i,j)

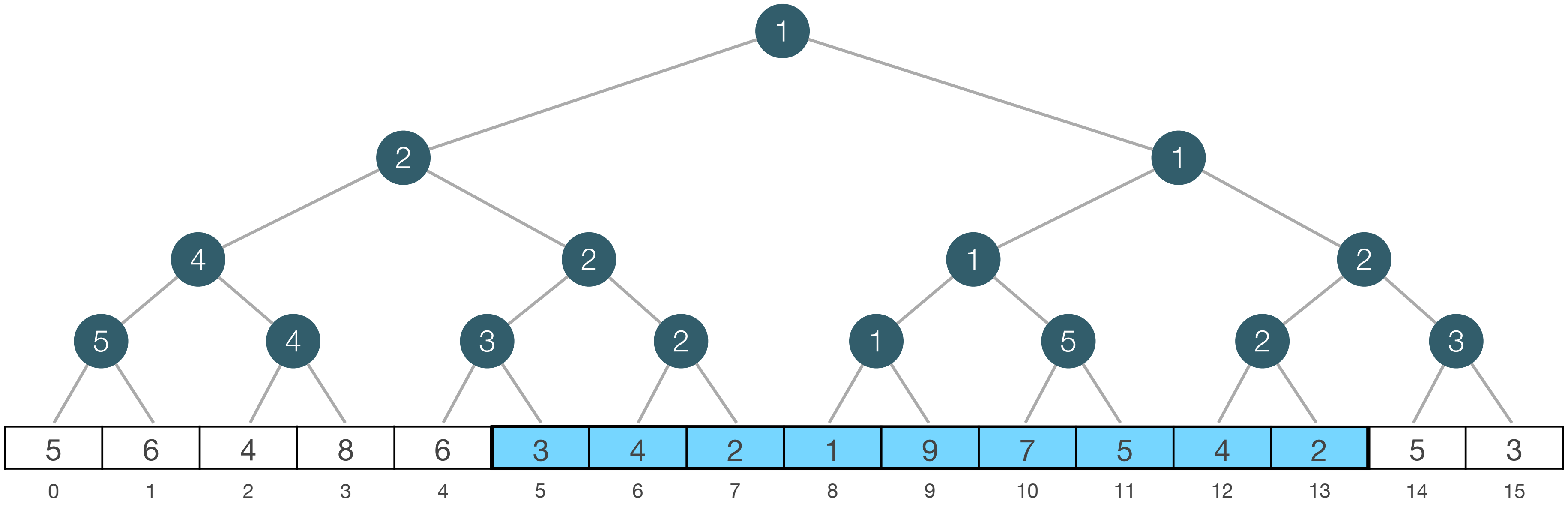


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3

Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

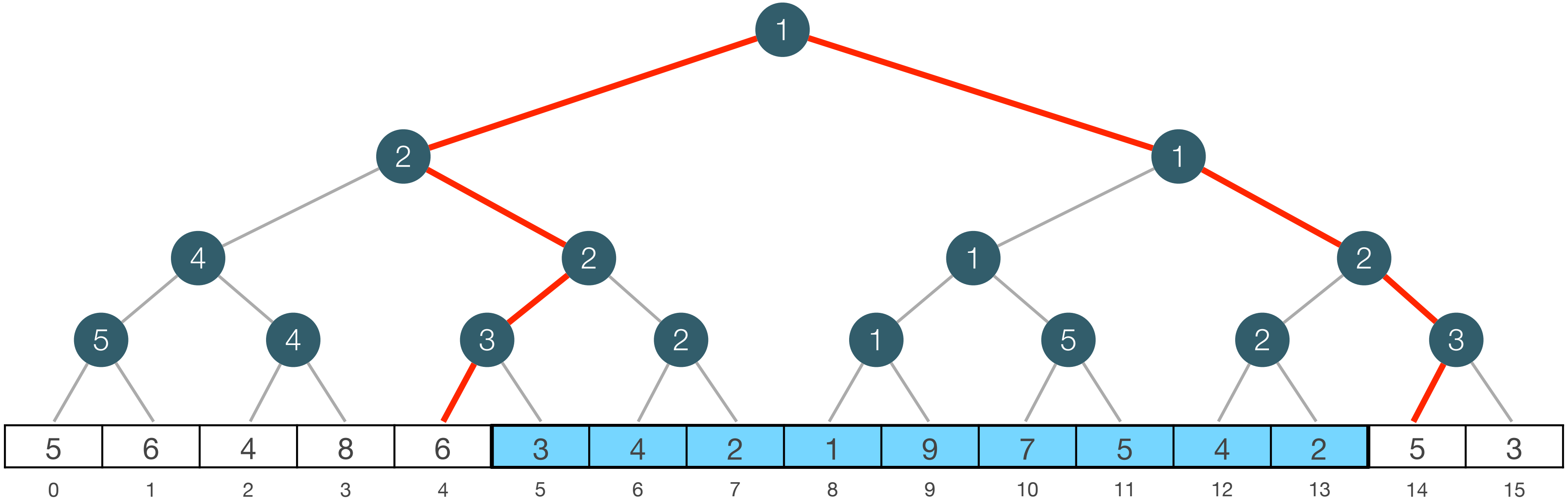
- RMQ(5,13) = ?



Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

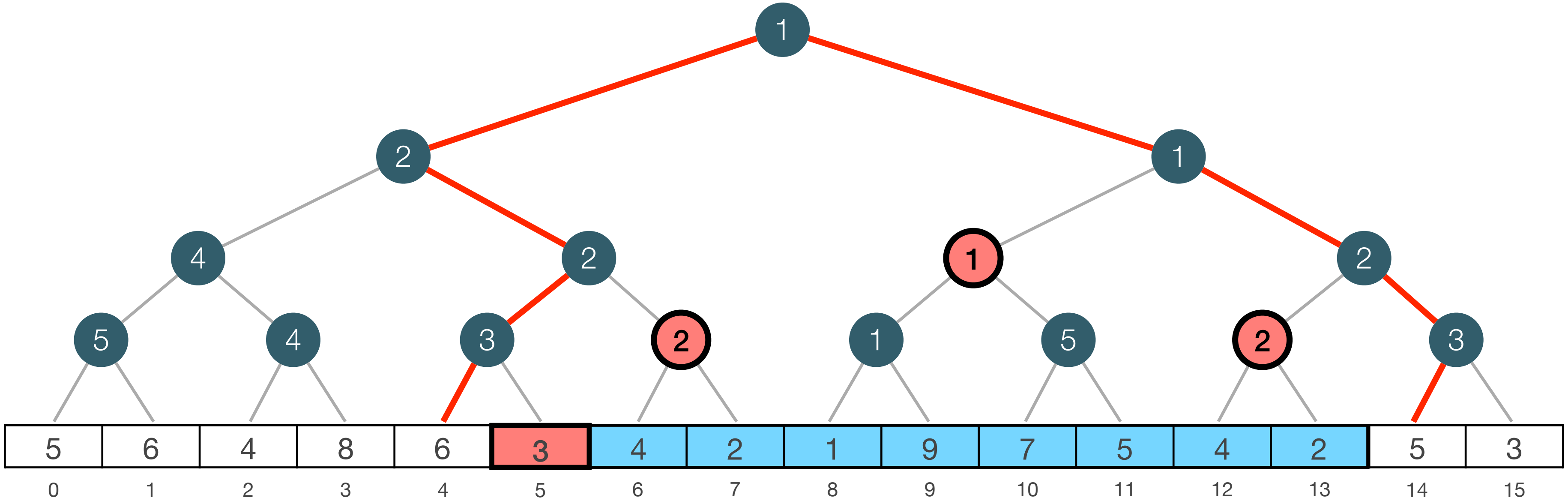
- RMQ(5,13) = ?



Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

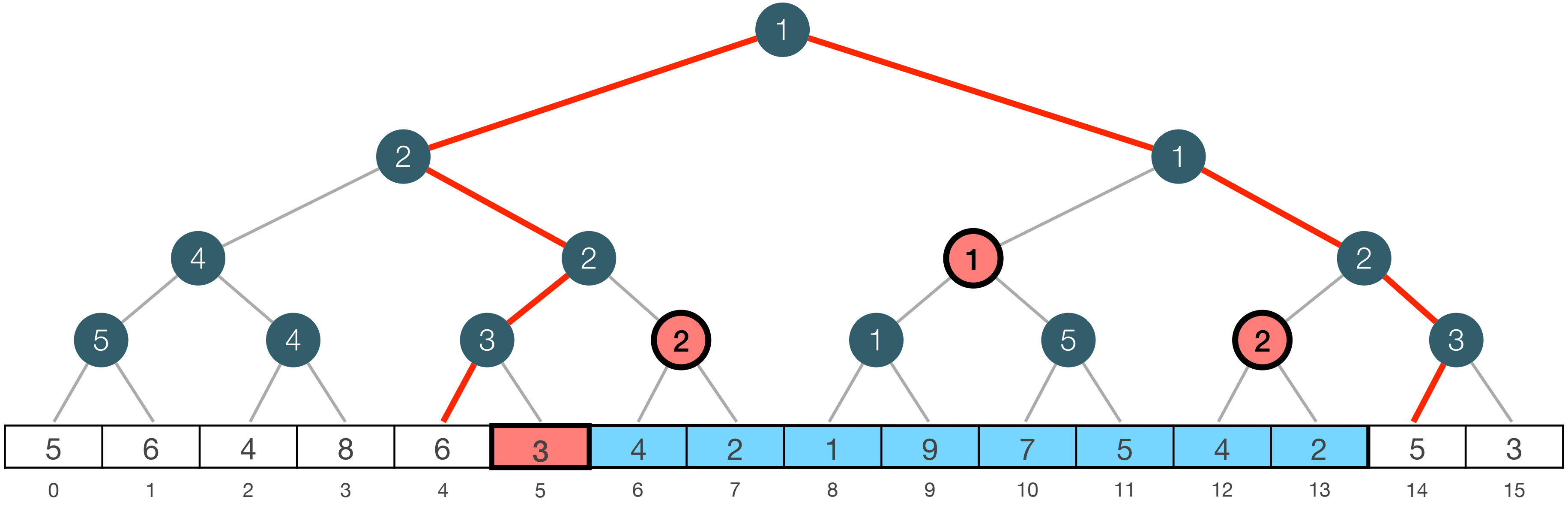
- RMQ(5,13) = ?



Segment trees

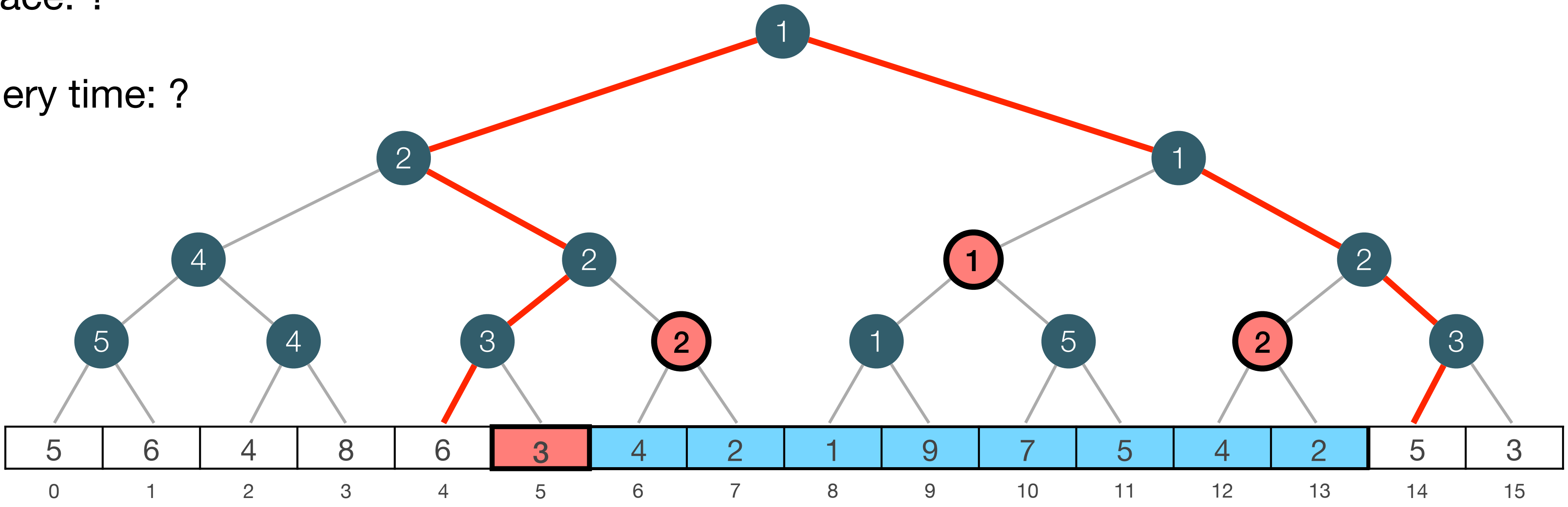
- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = min(3, 2, 1, 2) = 1



Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)
- RMQ(5,13) = min(3, 2, 1, 2) = 1
- Space: ?
- Query time: ?



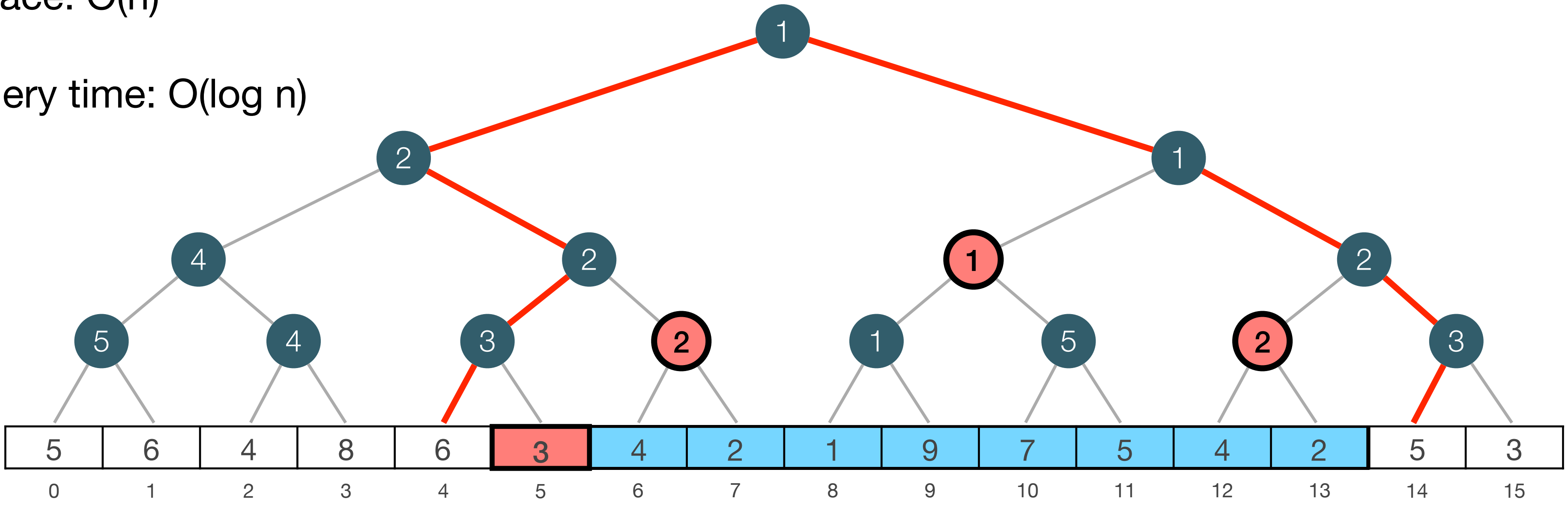
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = min(3, 2, 1, 2) = 1

- Space: O(n)

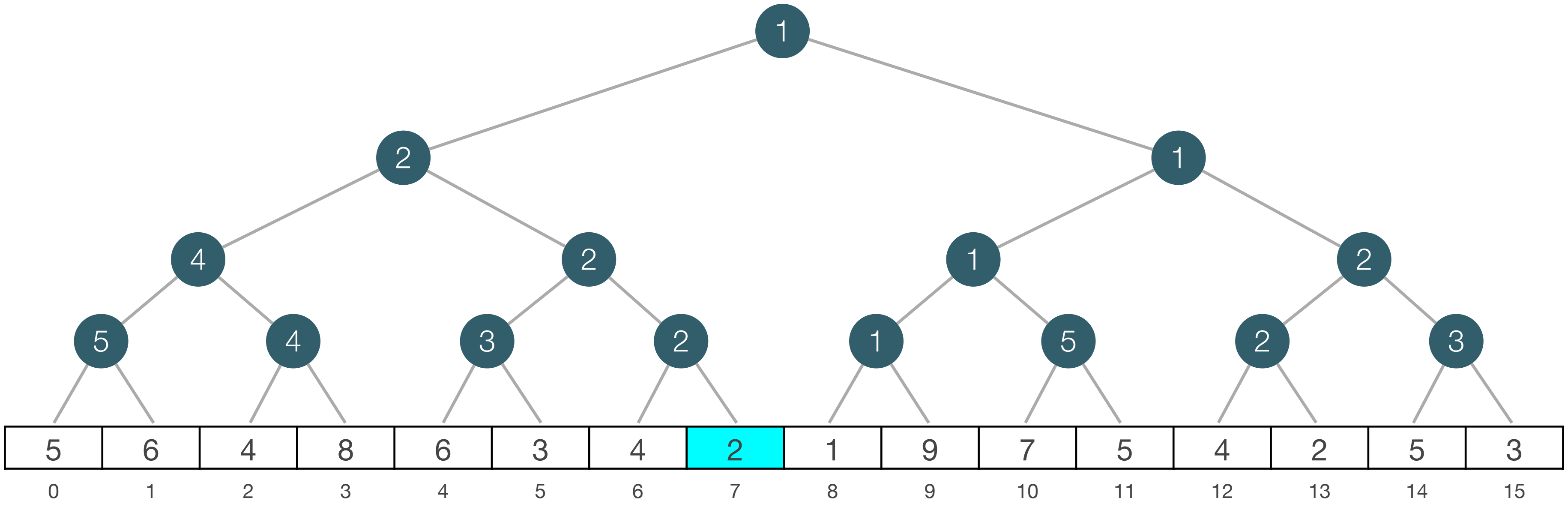
- Query time: O(log n)



Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

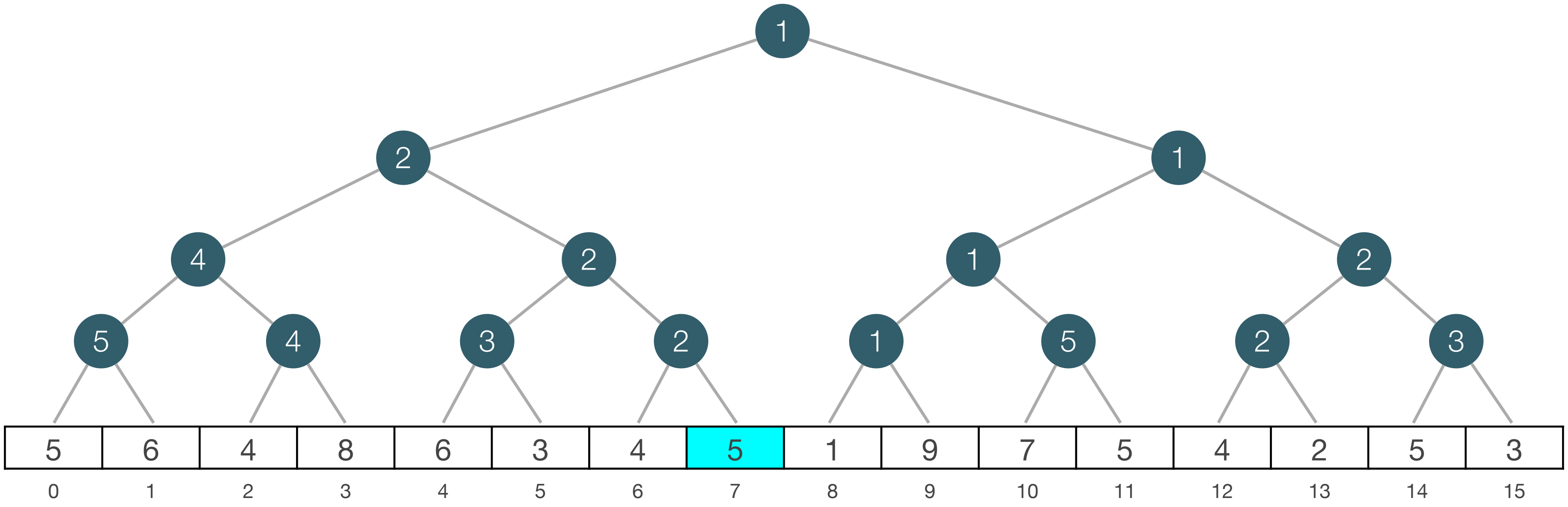
- Add(7,3)



Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k (k can be negative).
  - RMQ(i,j)

- Add(7,3)

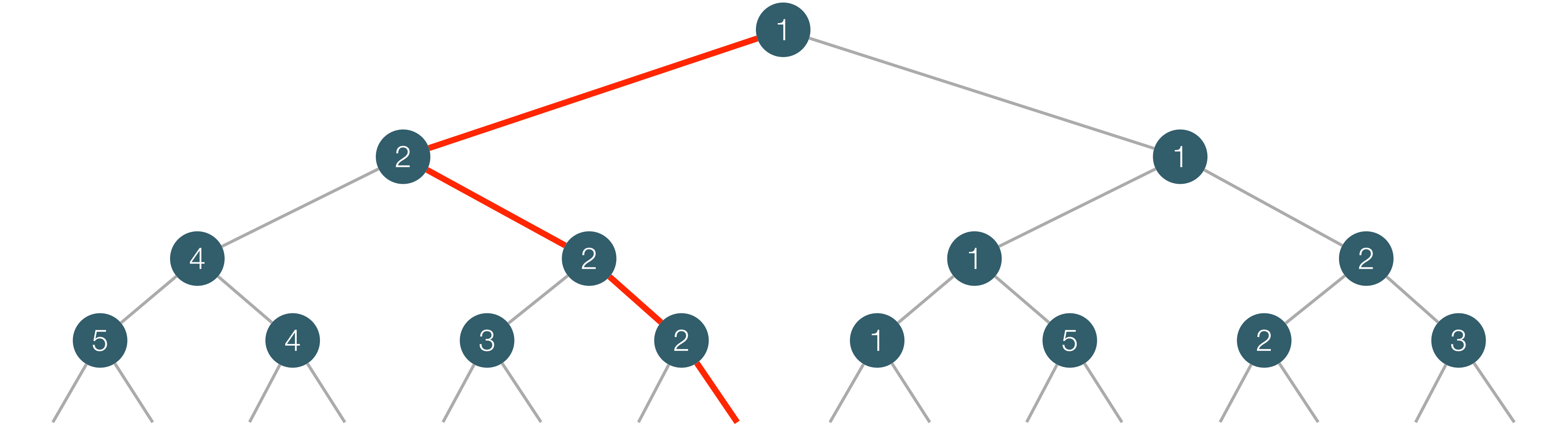


5	6	4	8	6	3	4	5	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- Add(7,3)

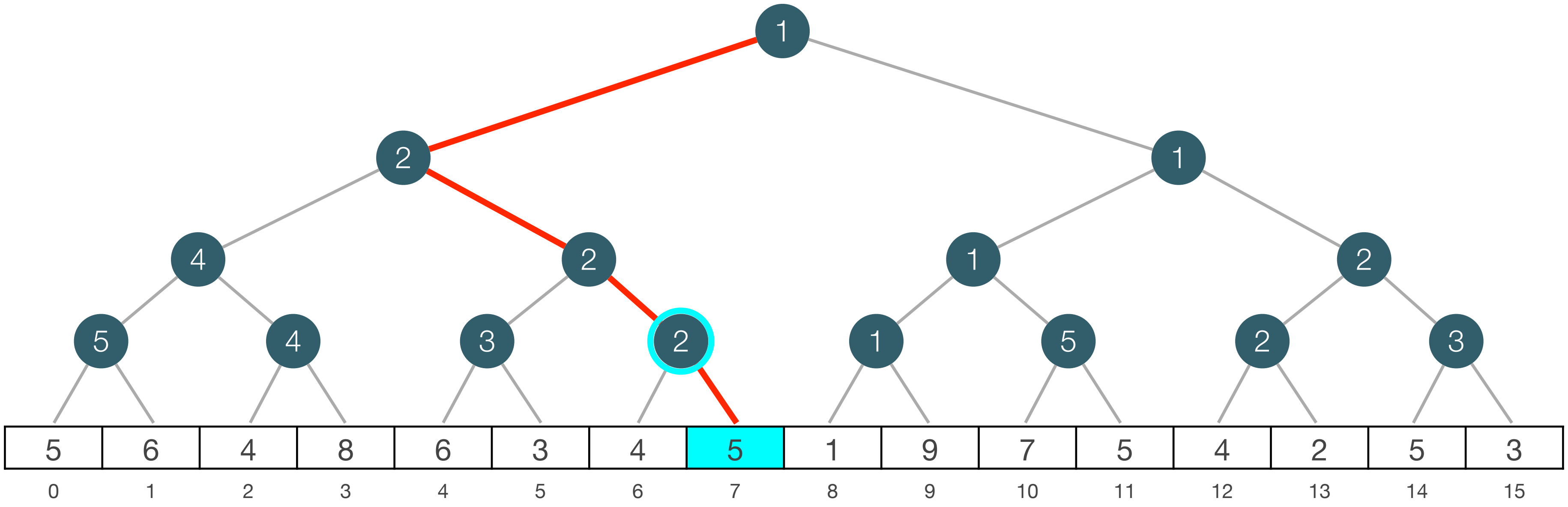


5	6	4	8	6	3	4	5	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- Add(7,3)

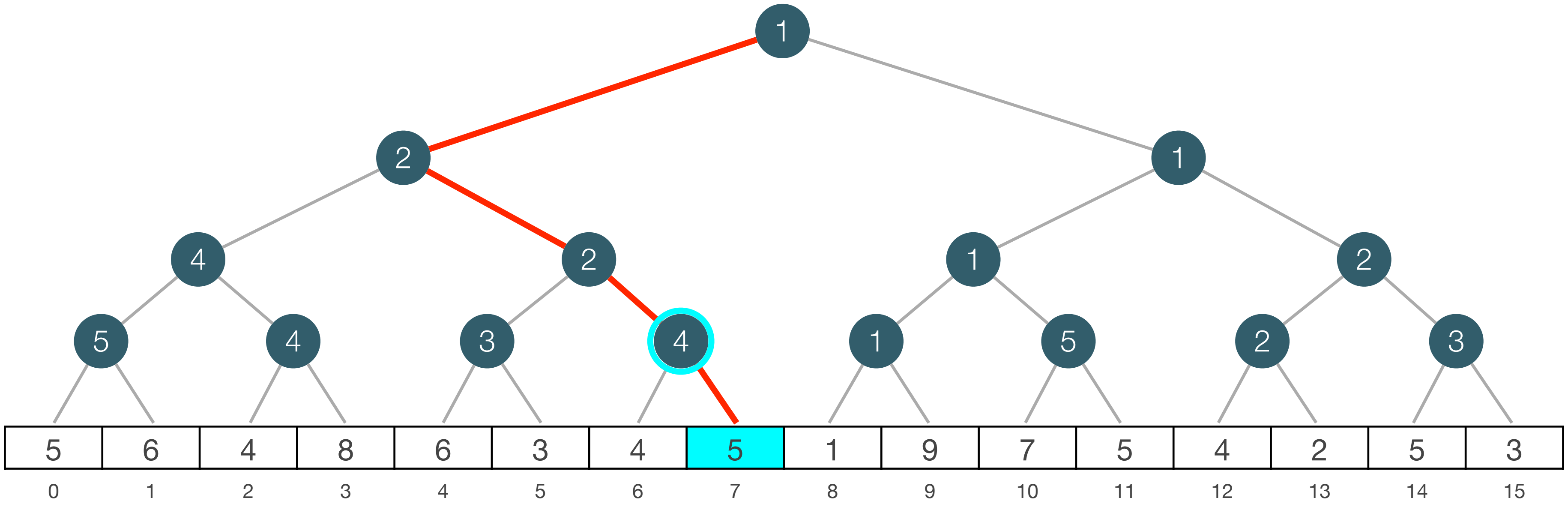


5	6	4	8	6	3	4	5	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- Add(7,3)

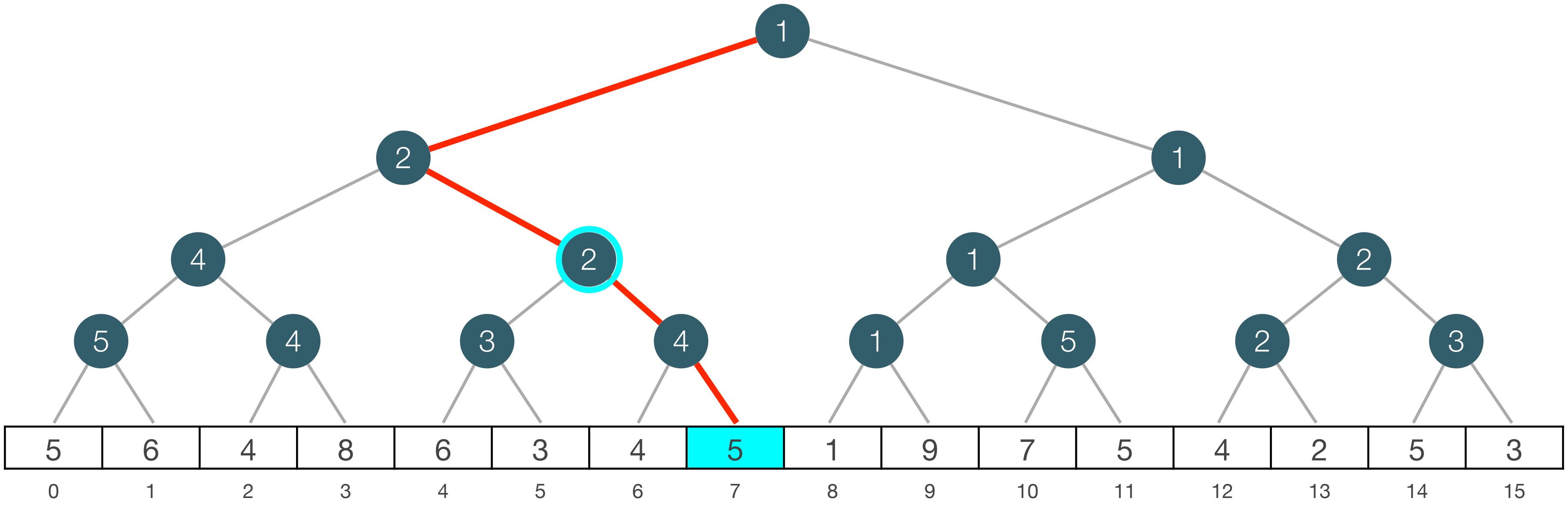


5	6	4	8	6	3	4	5	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- Add(7,3)

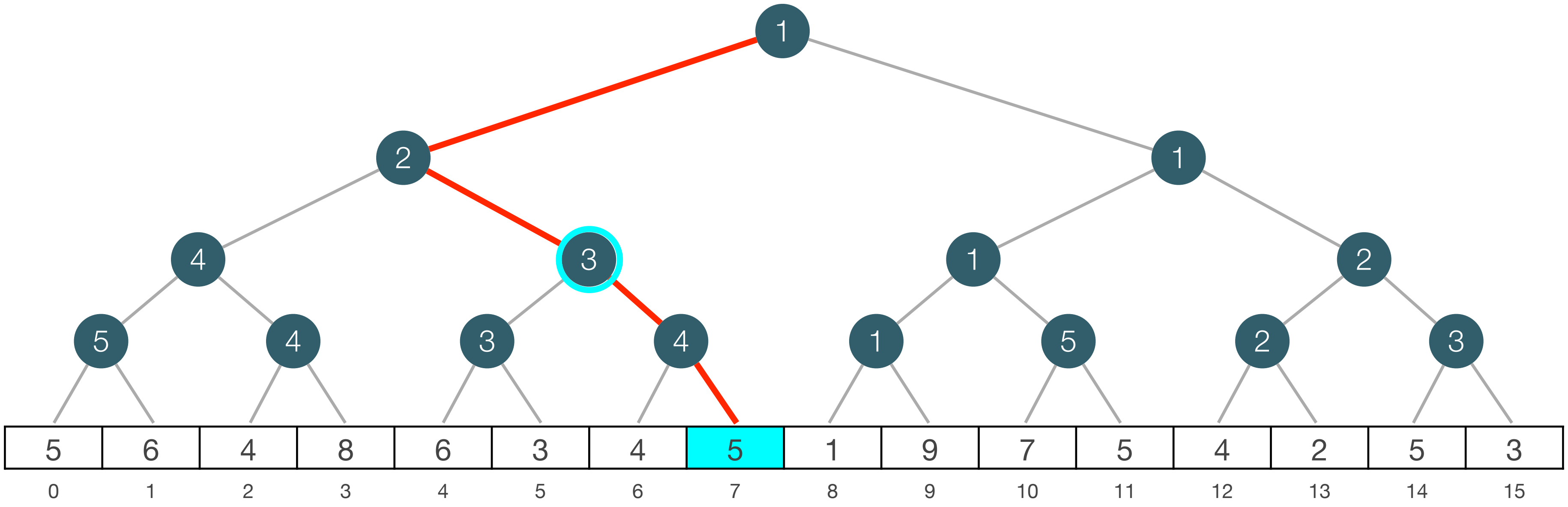


5	6	4	8	6	3	4	5	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k (k can be negative).
  - RMQ(i,j)

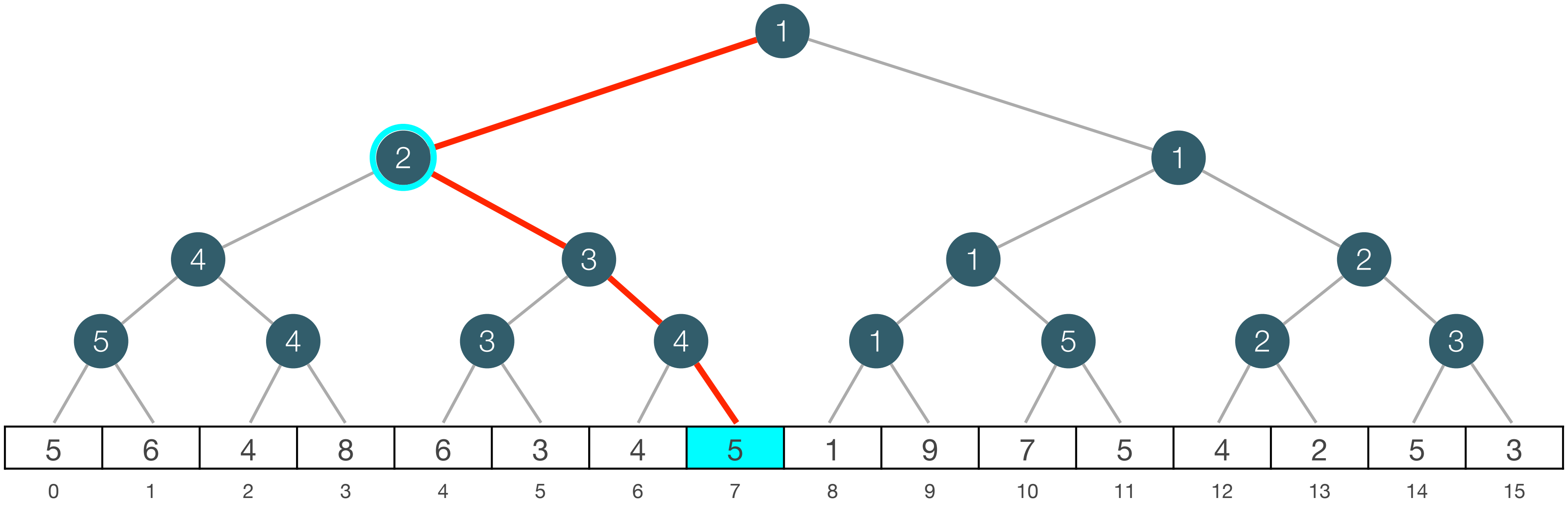
- Add(7,3)



Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

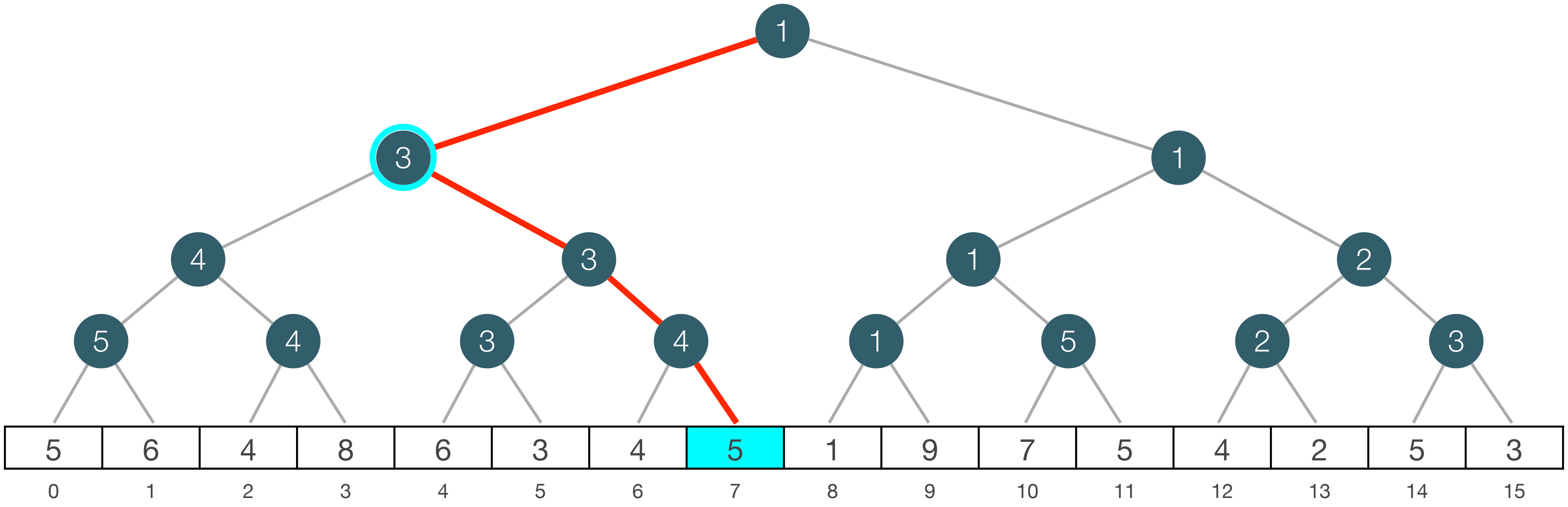
- Add(7,3)



Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

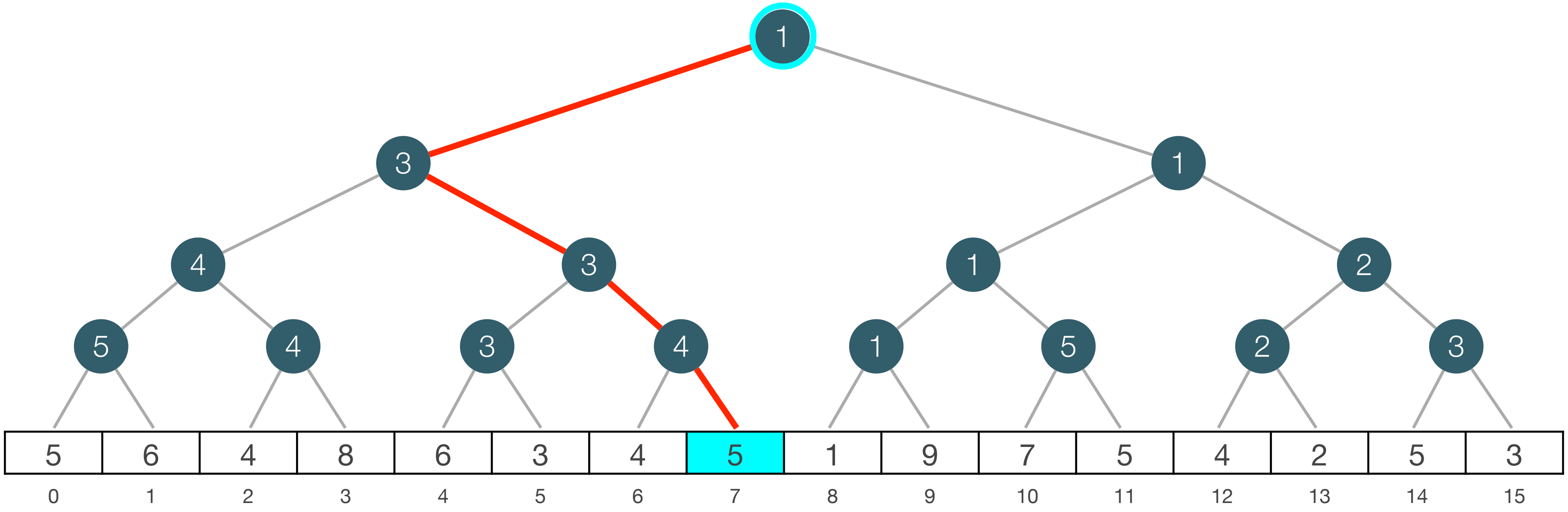
- Add(7,3)



Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k (k can be negative).
  - RMQ(i,j)

- Add(7,3)

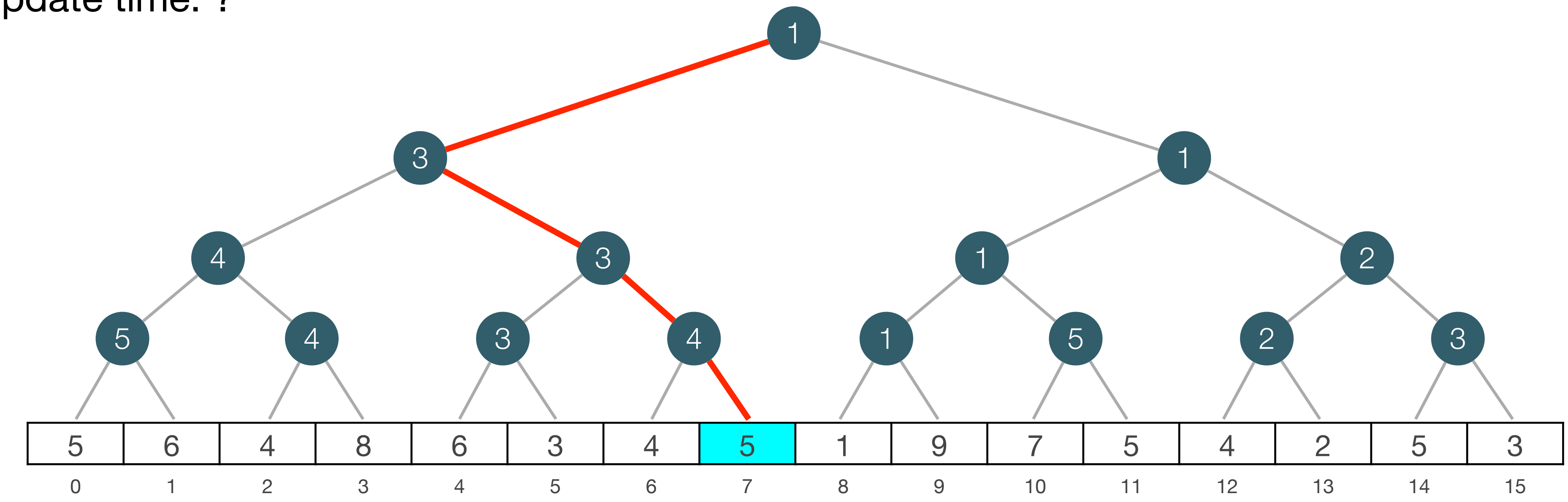


Segment trees

- Dynamic RMQ: Support following operations.
    - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
    - RMQ(i,j)

- Add(7,3)

- Update time: ?

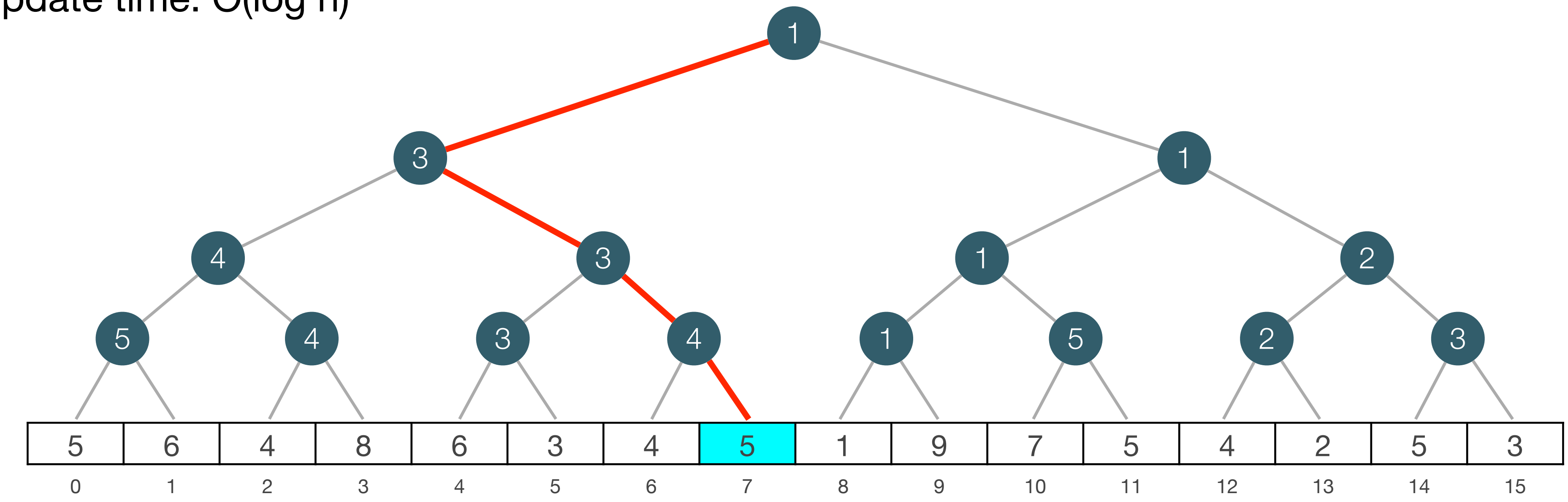


Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- Add(7,3)

- Update time: O(log n)



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

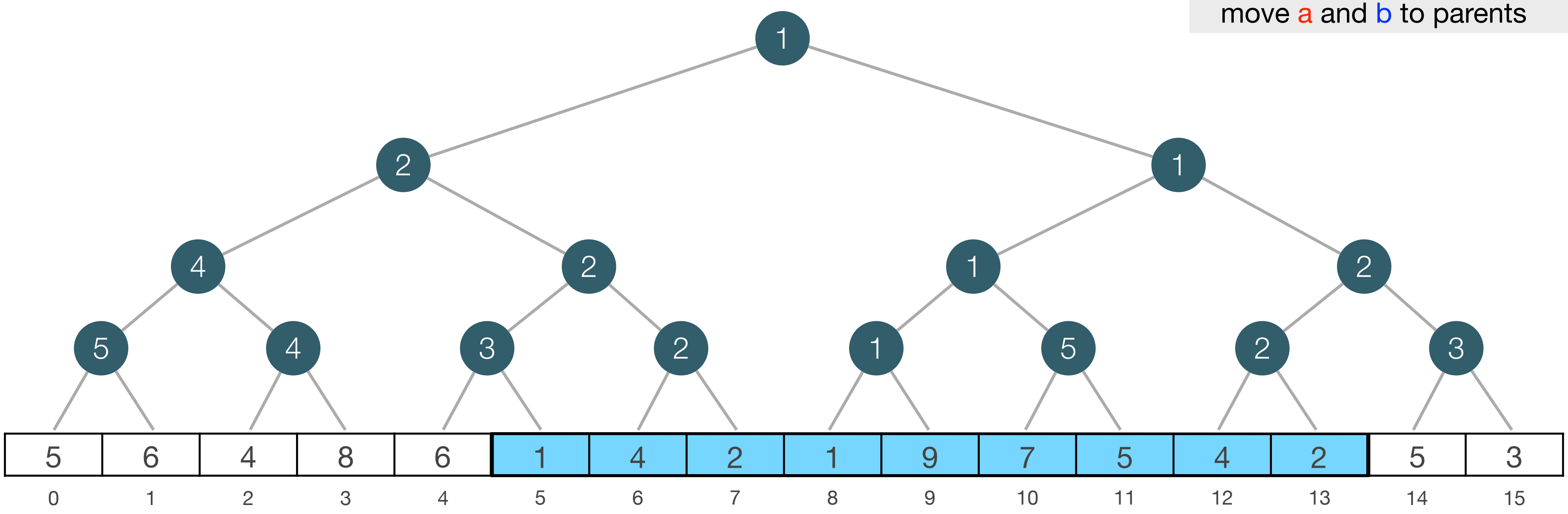
Compact Implementation

Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
```



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

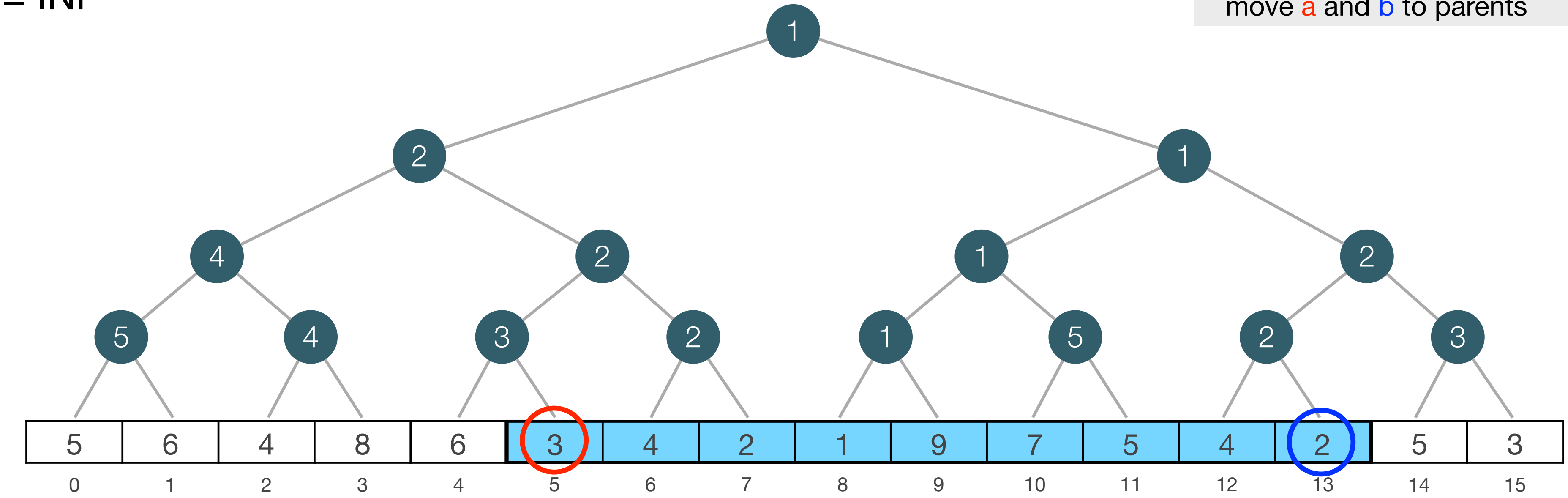
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

- s = INF

```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
```



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

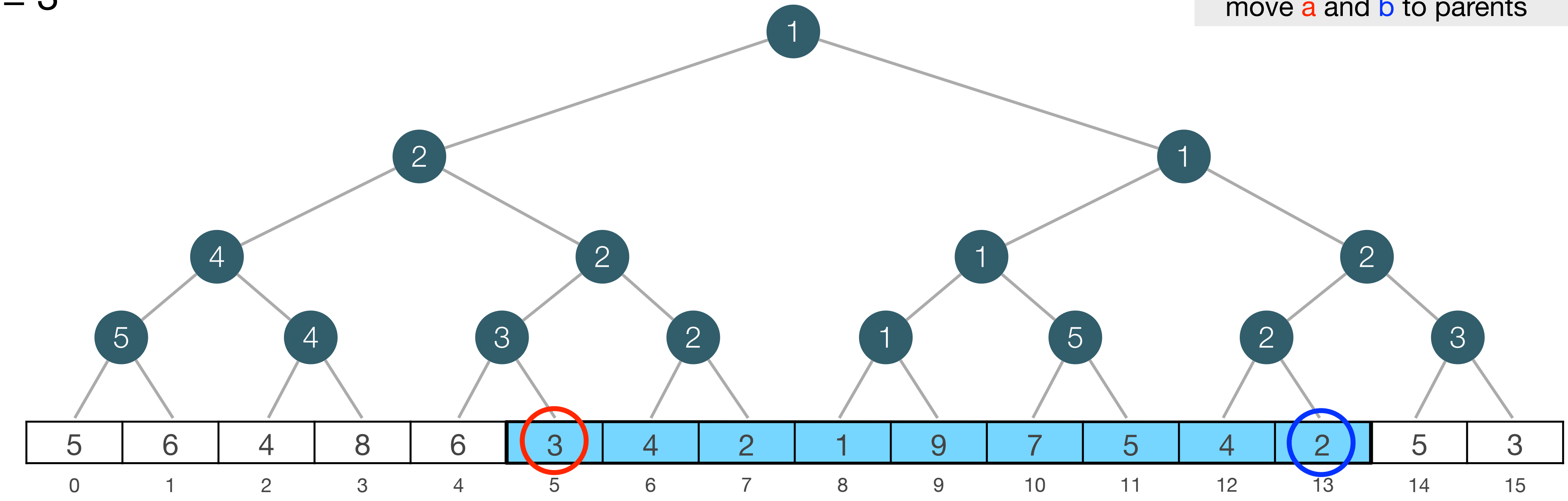
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

- s = 3

```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
```



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

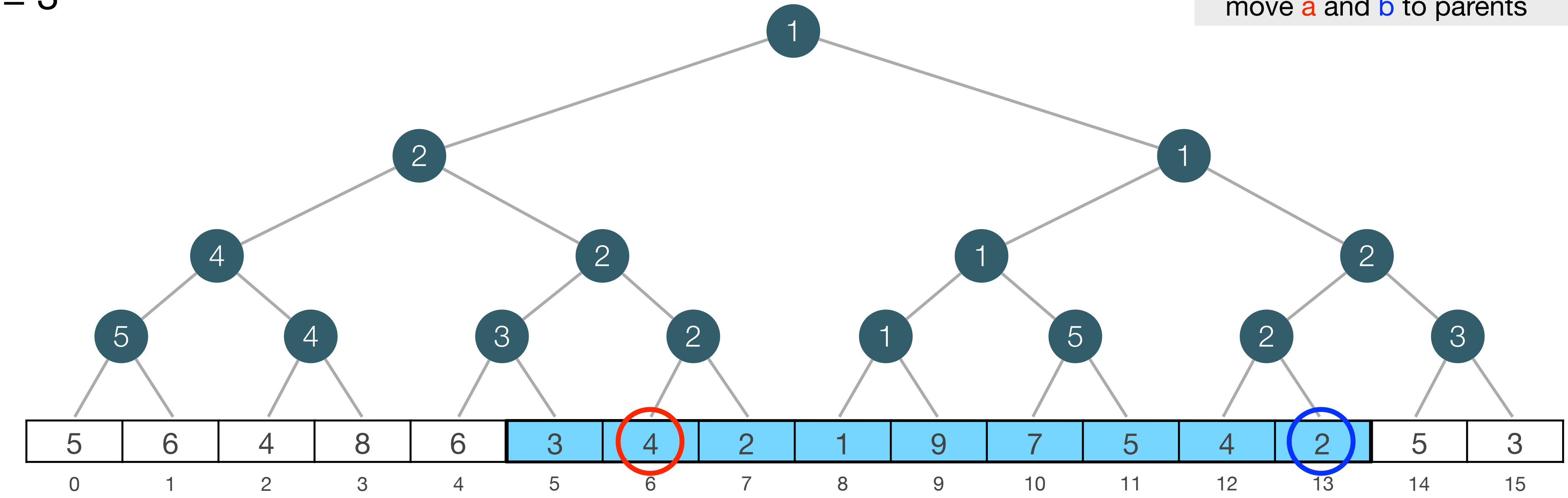
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

- s = 3

```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
```



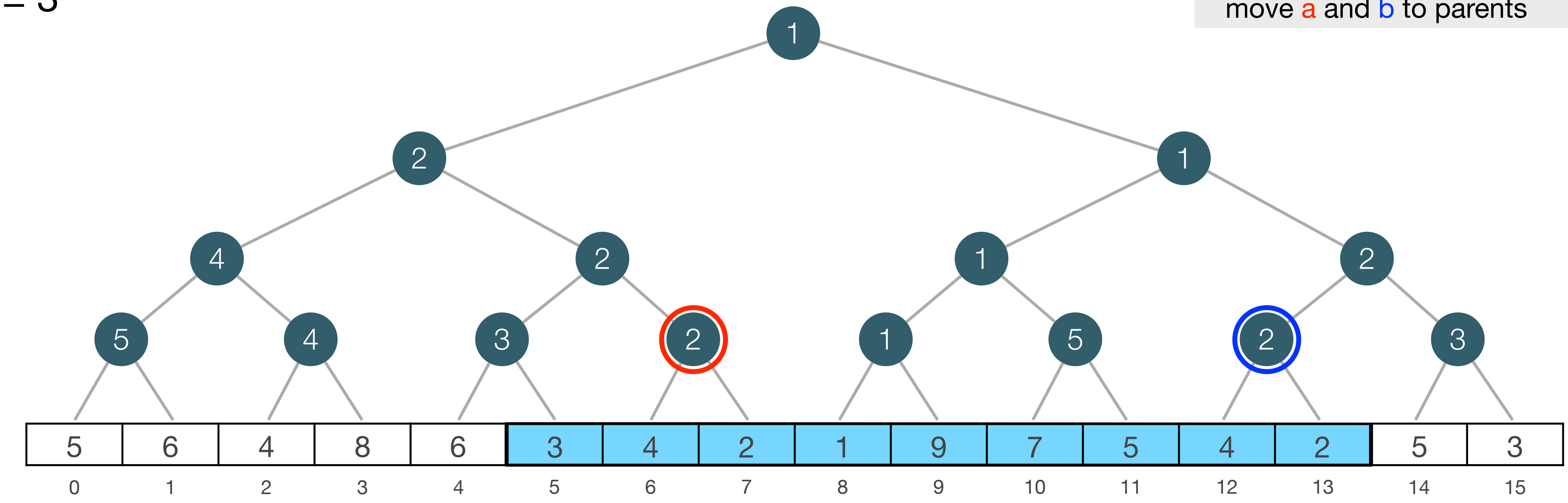
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

- s = 3

```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
```



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

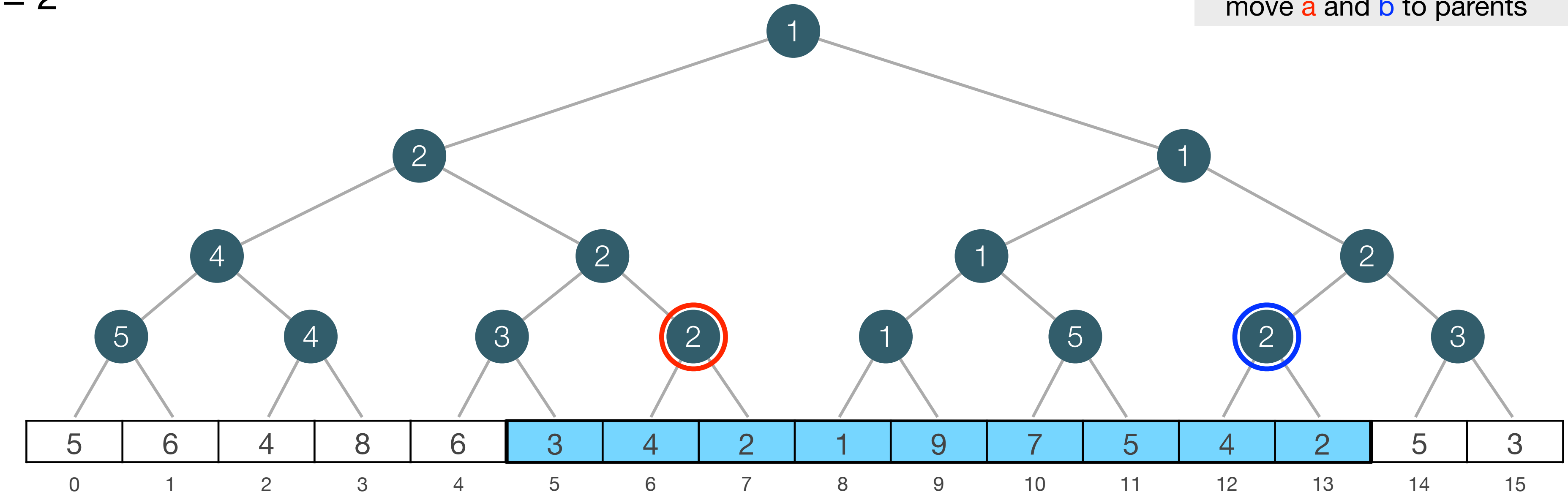
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

- s = 2

```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
```



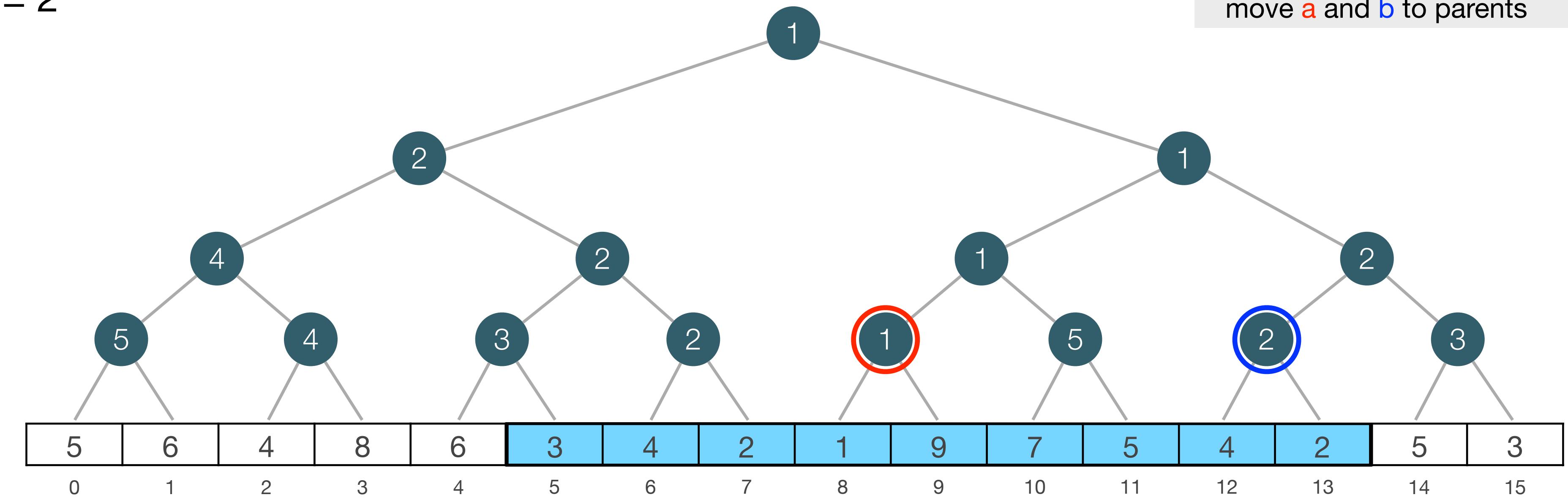
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

- s = 2

```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
```



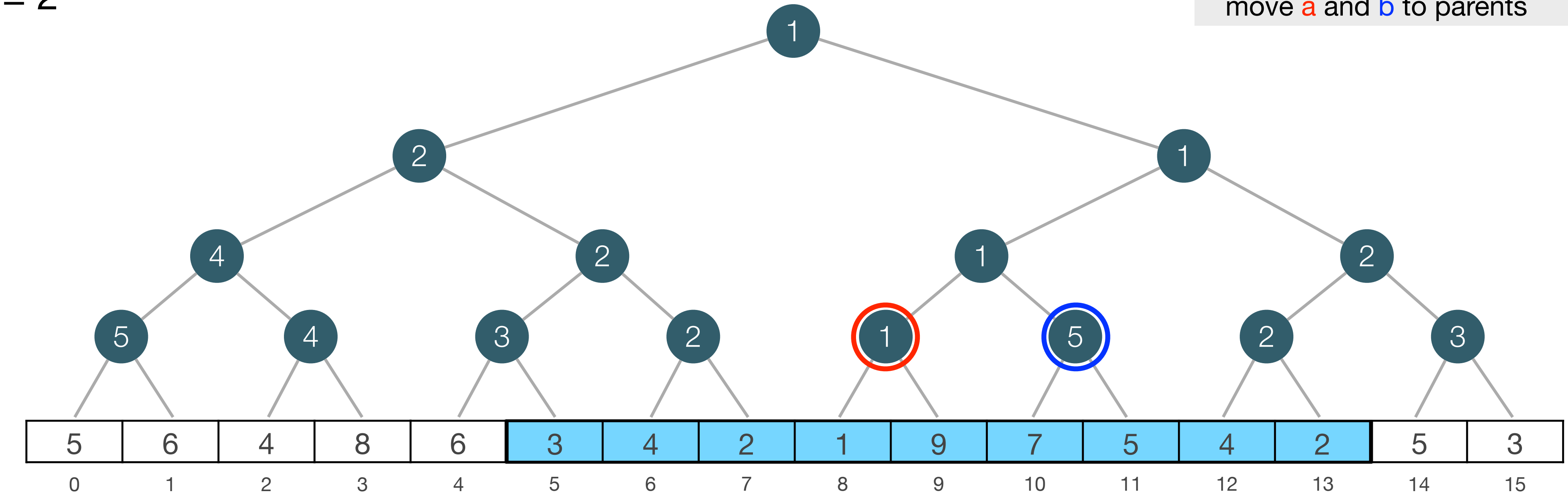
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

- s = 2

```
s = INF
a = i, b = j
while (a not right of b):
    if (a right child):
        s = min(s, tree[a])
        move a to the right
    if (b left child):
        s = min(s, tree[b])
        move b to the left
    move a and b to parents
```



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

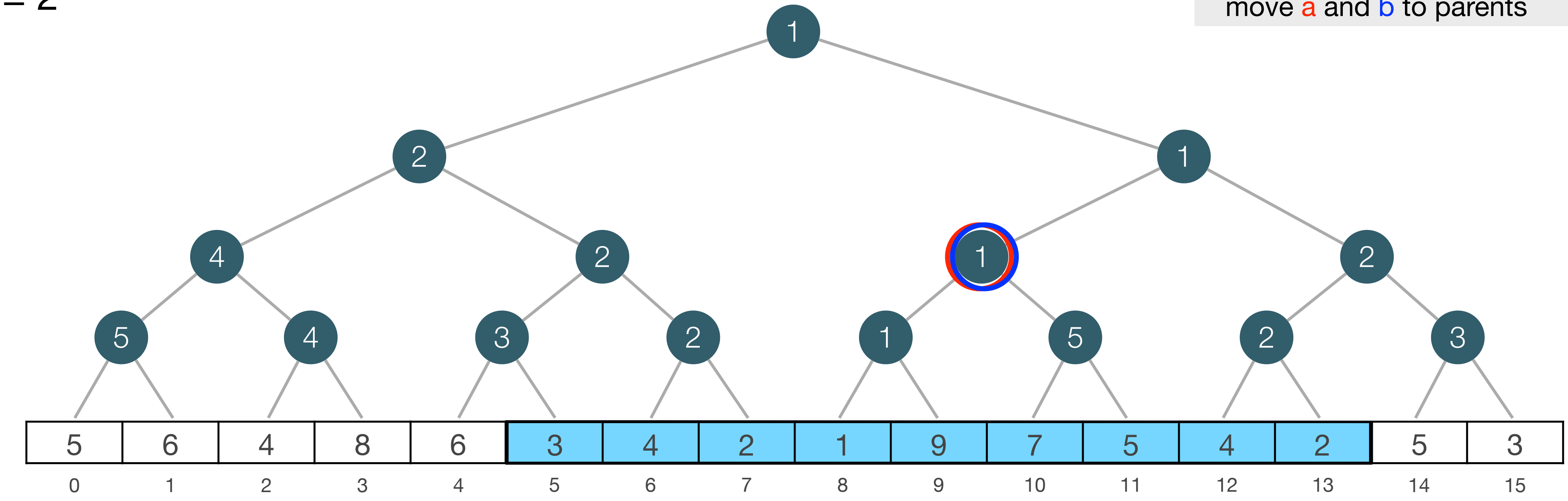
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

- s = 2

```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
```



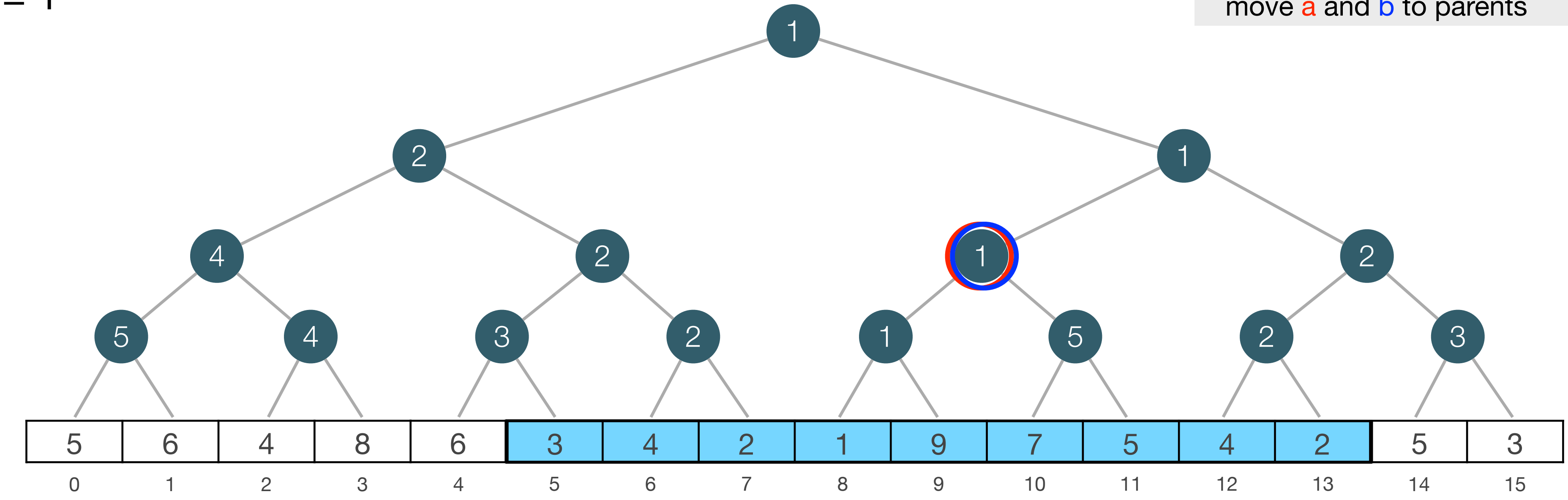
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

- s = 1

```
s = INF
a = i, b = j
while (a not right of b):
    if (a right child):
        s = min(s, tree[a])
        move a to the right
    if (b left child):
        s = min(s, tree[b])
        move b to the left
    move a and b to parents
```



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

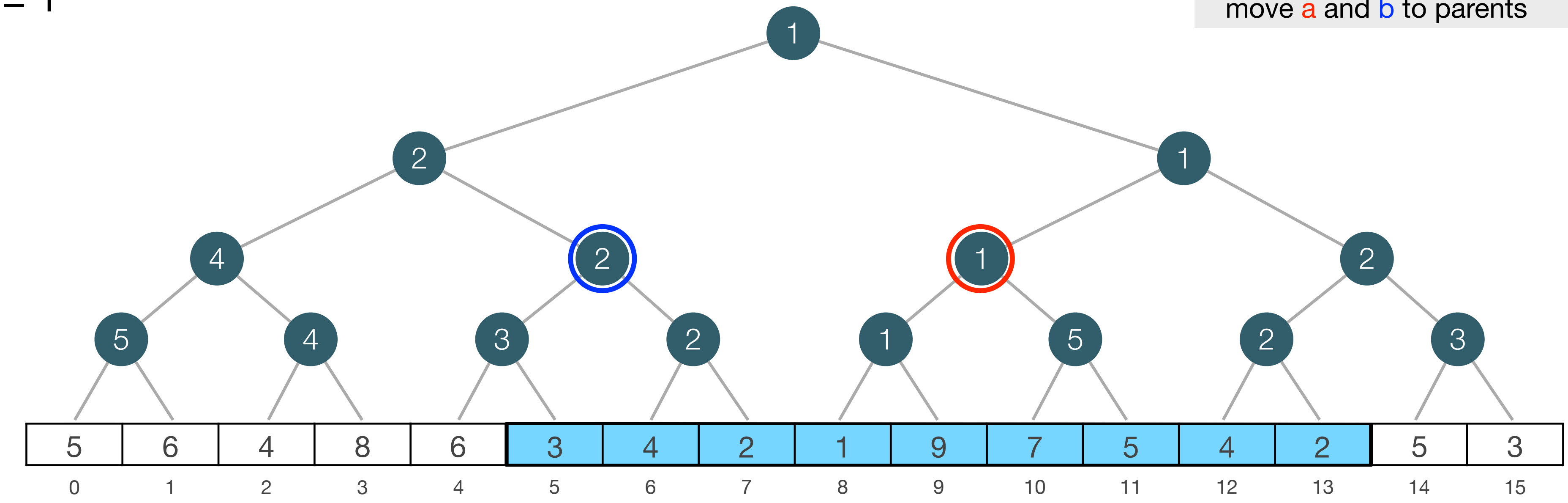
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = ?

- s = 1

```
s = INF
a = i, b = j
while (a not right of b):
    if (a right child):
        s = min(s, tree[a])
        move a to the right
    if (b left child):
        s = min(s, tree[b])
        move b to the left
    move a and b to parents
```



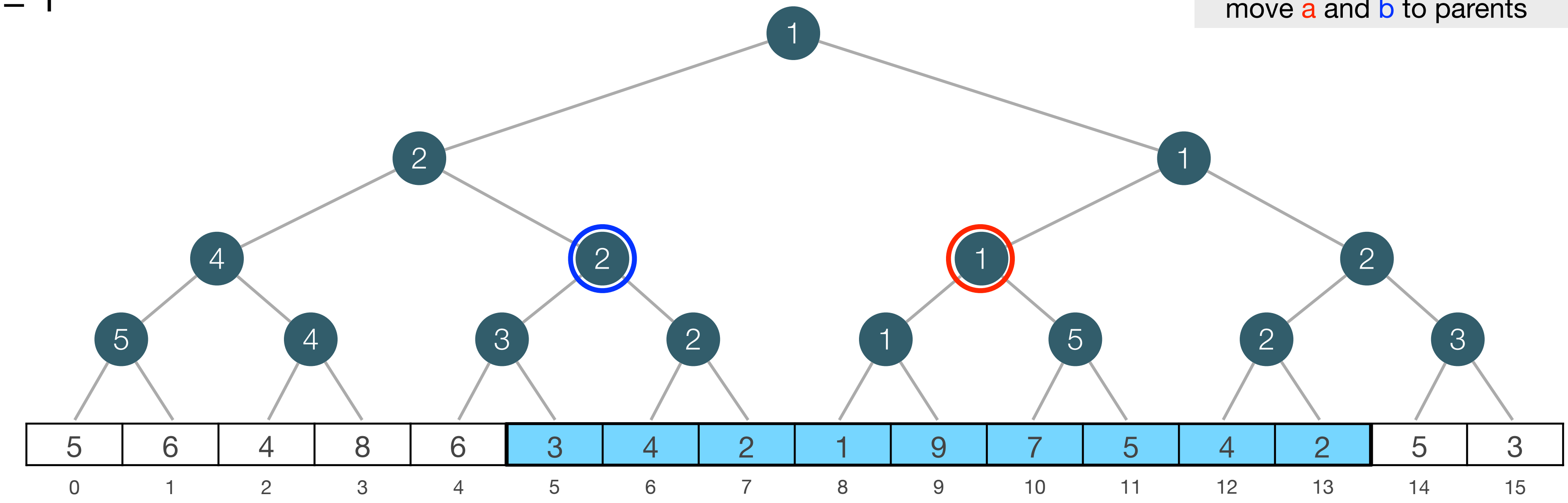
Segment trees

- Dynamic RMQ: Support following operations.
  - Add(i, k): Set A[i] = A[i] + k   (k can be negative).
  - RMQ(i,j)

- RMQ(5,13) = 1

- s = 1

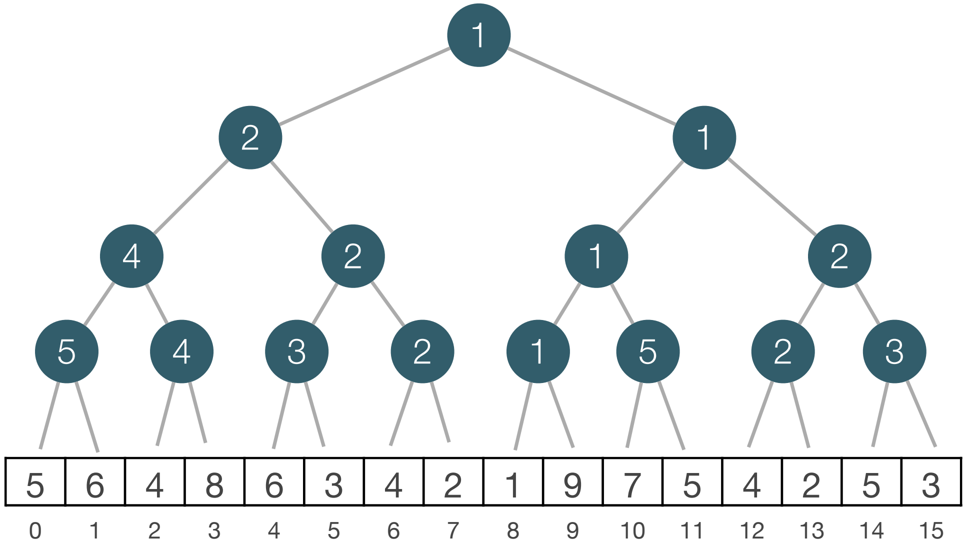
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
```



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

Segment trees

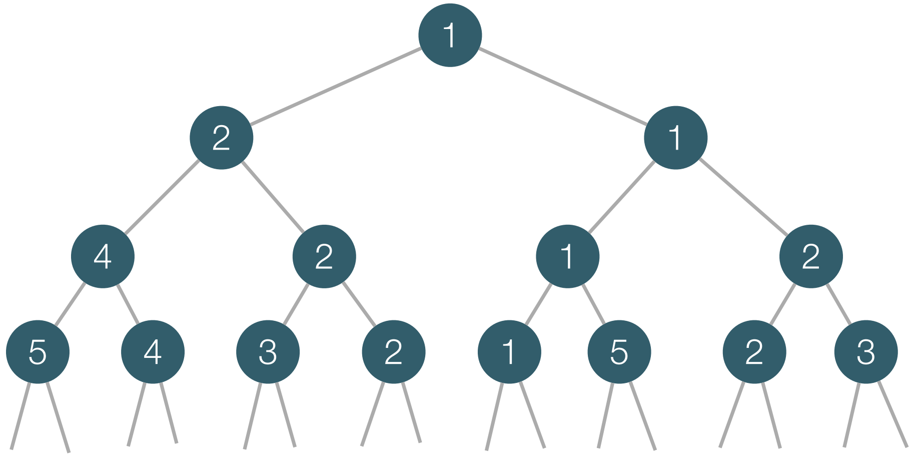
- Layout of tree:



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

Segment trees

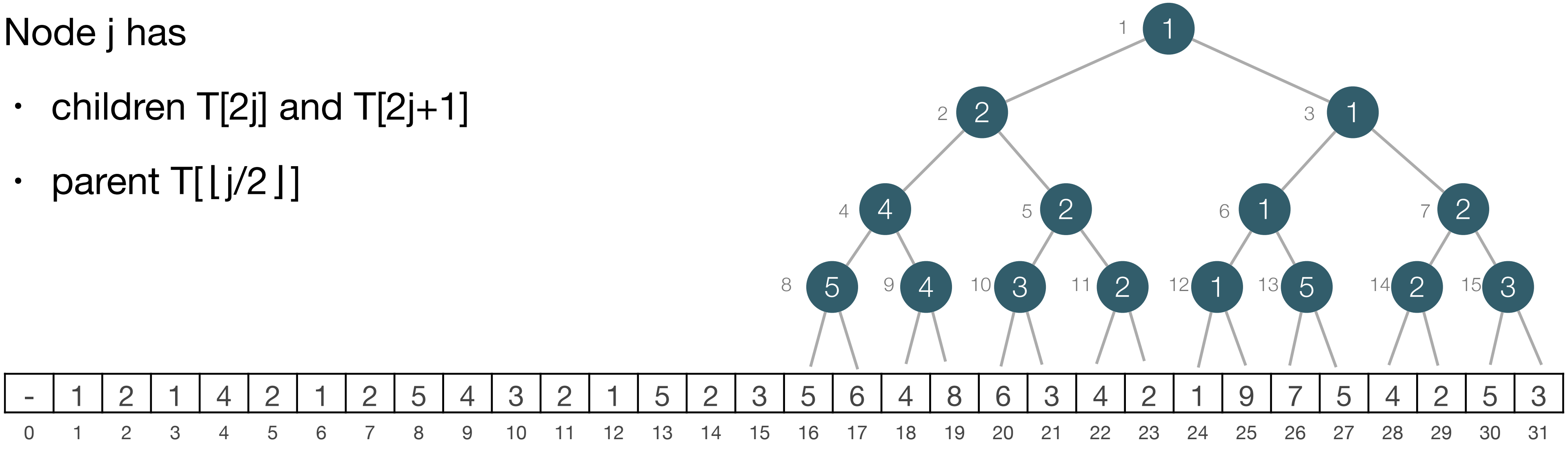
- Layout of tree:
  - Array T of length 2n.
  - T[n+i] = A[i]



-																5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

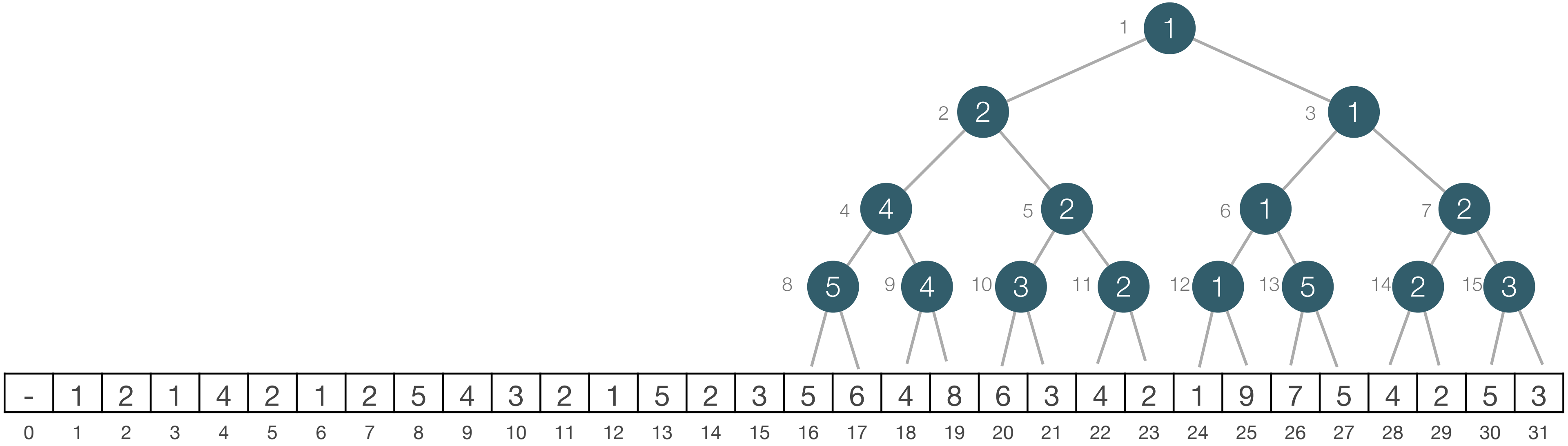
Segment trees

- Layout of tree: heap layout
    - Array T of length 2n.
    - T[n+i] = A[i]
    - T[1] is the root
    - T[2] is the left child and T[3] is the right child of the root.
    - Node j has
        - children T[2j] and T[2j+1]
        - parent T[⌊j/2⌋]



-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Segment trees



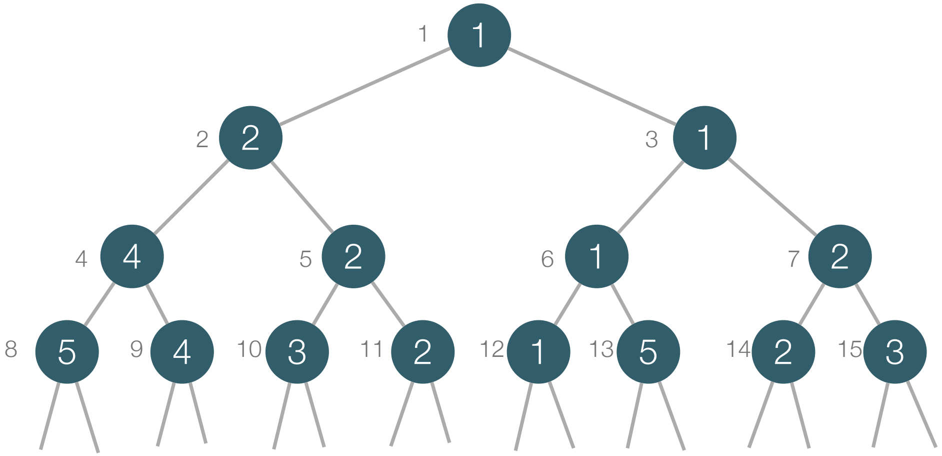
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3

```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a = ⌊a/2⌋, b = ⌊b/2⌋
return s
```

Segment trees

RMQ(3,9) = ?



-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

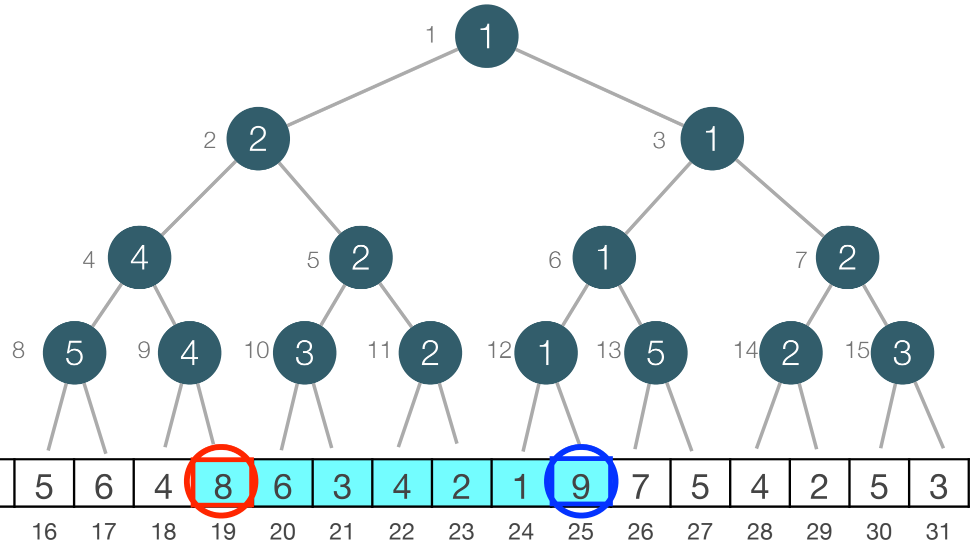
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a = ⌊a/2⌋, b = ⌊b/2⌋
return s
```

Segment trees

RMQ(3,9) = ?

s = INF



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3

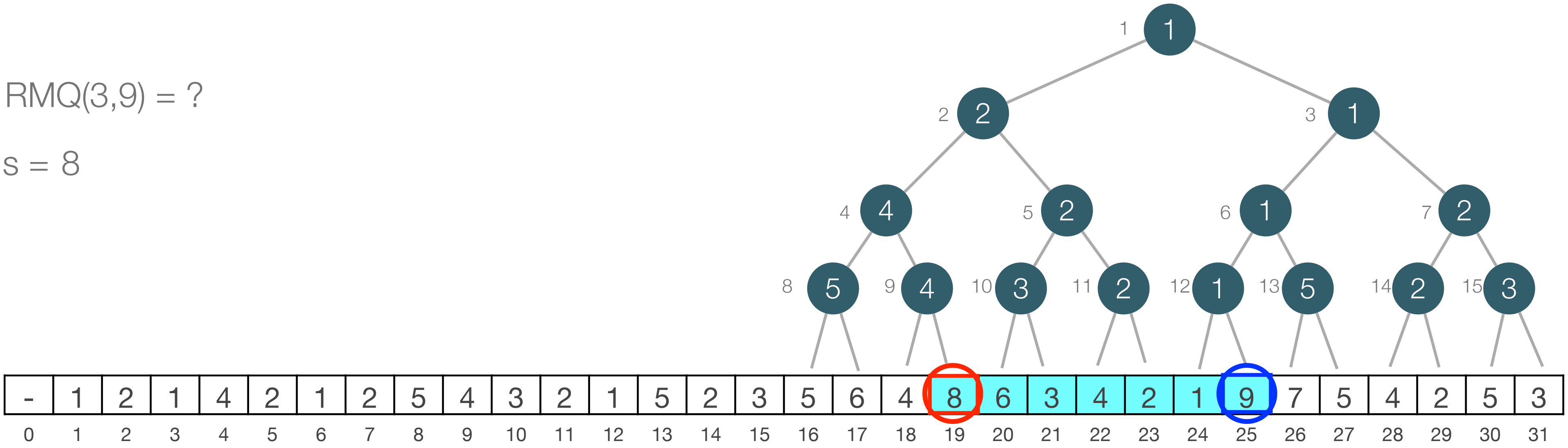
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a = ⌊a/2⌋, b = ⌊b/2⌋
return s
```

Segment trees

RMQ(3,9) = ?

s = 8



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3

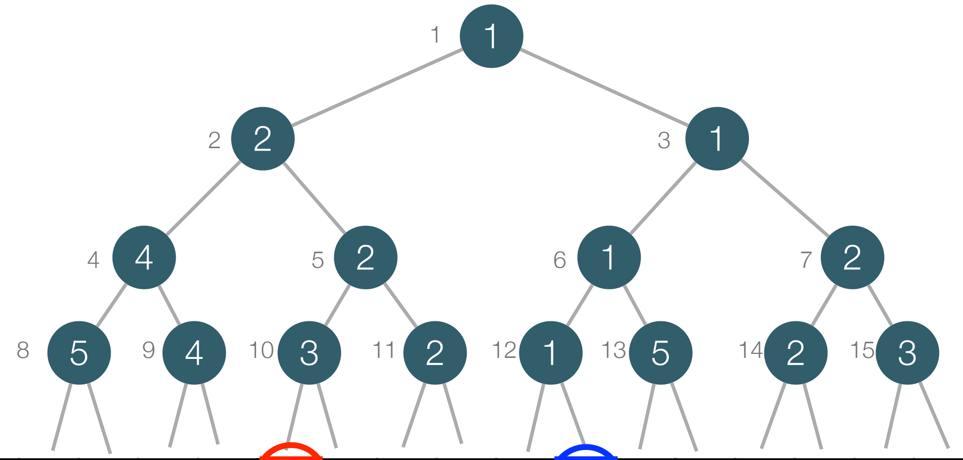
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a = ⌊a/2⌋, b = ⌊b/2⌋
return s
```

Segment trees

RMQ(3,9) = ?

s = 8



-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

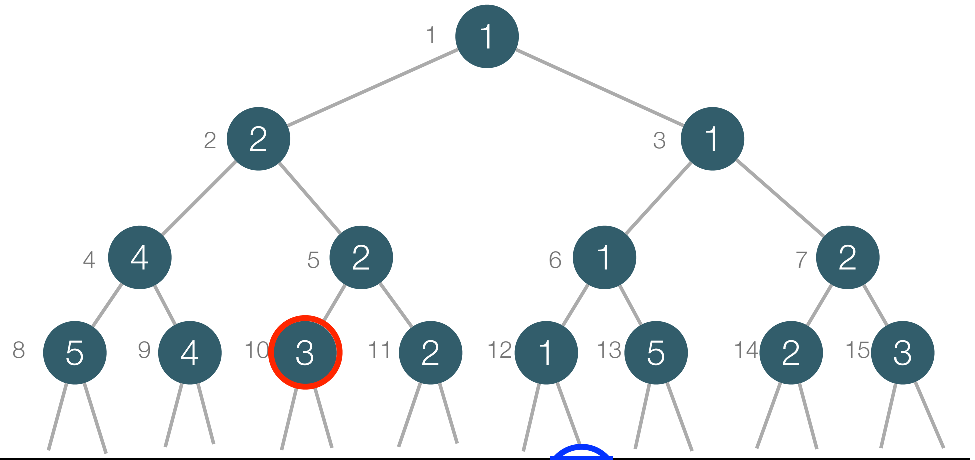
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a = ⌊a/2⌋, b = ⌊b/2⌋
return s
```

Segment trees

RMQ(3,9) = ?

s = 8



-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

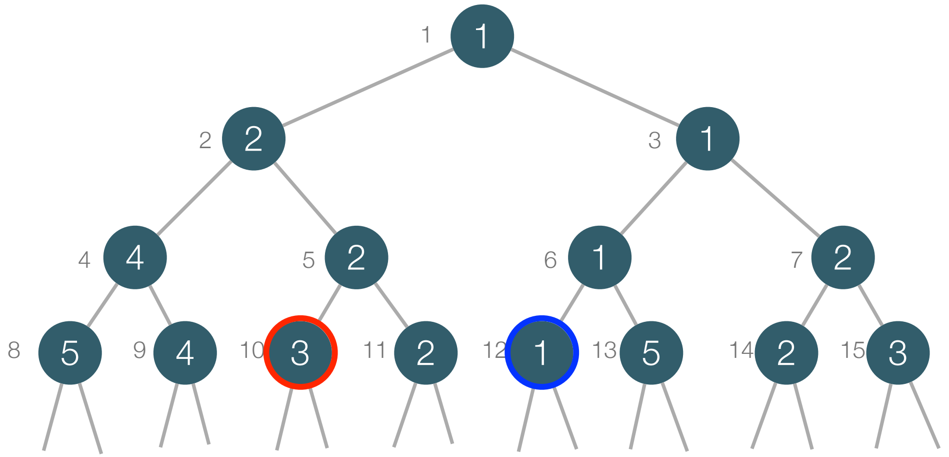
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a = ⌊a/2⌋, b = ⌊b/2⌋
return s
```

Segment trees

RMQ(3,9) = ?

s = 8



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3

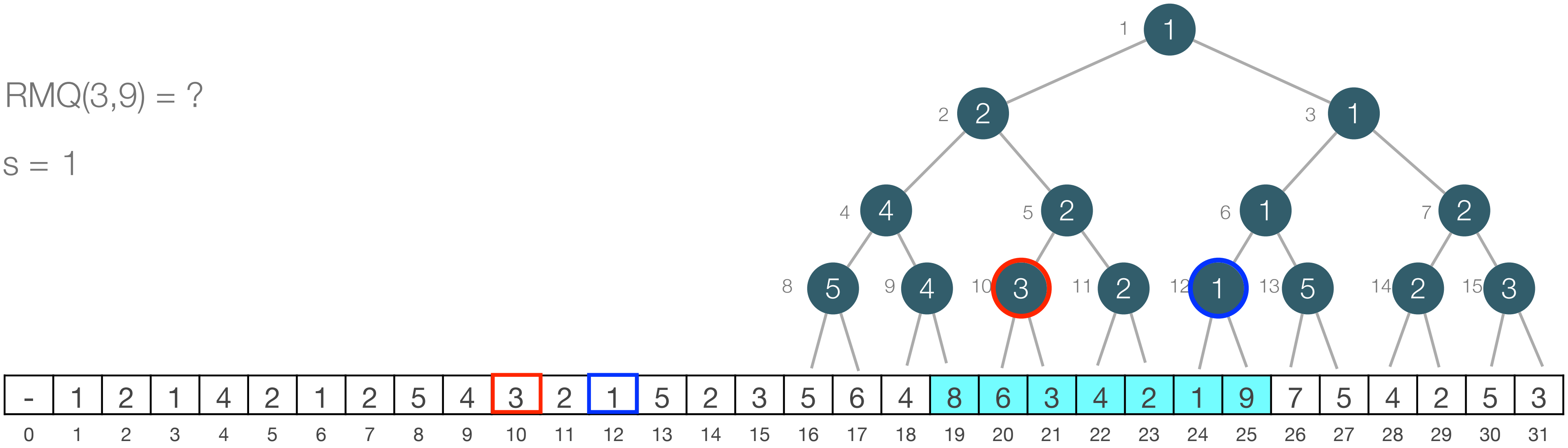
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a =⌊a/2⌋, b = ⌊b/2⌋
return s
```

Segment trees

RMQ(3,9) = ?

s = 1



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3

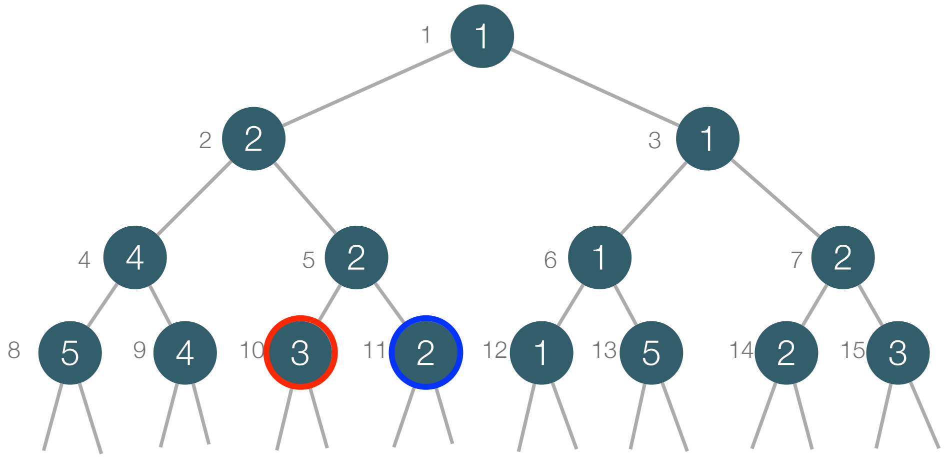
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a = ⌊a/2⌋, b = ⌊b/2⌋
return s
```

Segment trees

RMQ(3,9) = ?

s = 1



-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

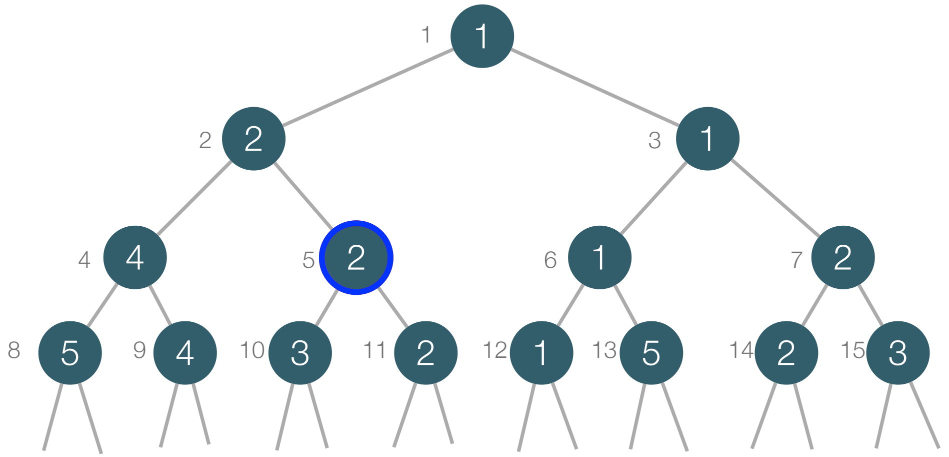
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a = $\lfloor a/2 \rfloor$, b = $\lfloor b/2 \rfloor$
return s
```

Segment trees

RMQ(3,9) = ?

s = 1



-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

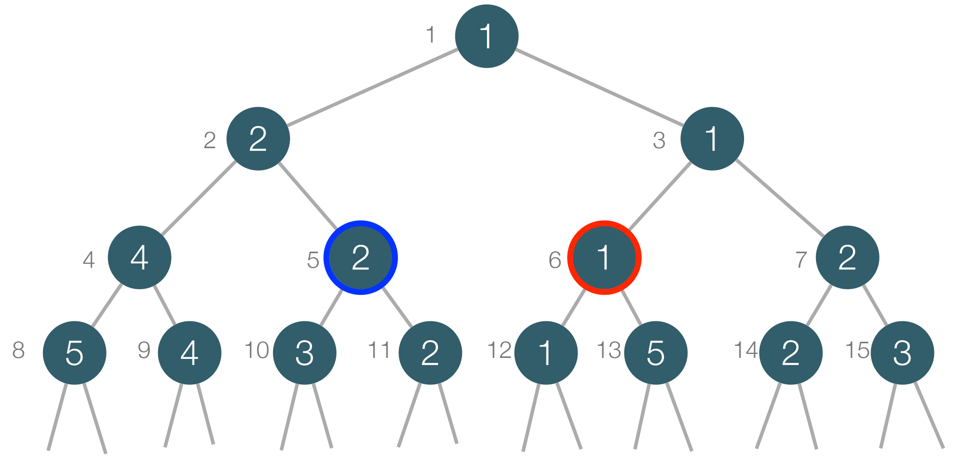
```
s = INF
a = i, b = j
while (a not right of b):
    if (a right child):
        s = min(s, tree[a])
        move a to the right
    if (b left child):
        s = min(s, tree[b])
        move b to the left
    move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
    if (a % 2 == 1):
        s = min(s, T[a])
        a = a + 1
    if (b % 2 == 0):
        s = min(s, T[b])
        b = b - 1
    a = ⌊a/2⌋, b = ⌊b/2⌋
return s
```

Segment trees

RMQ(3,9) = ?

s = 1



-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

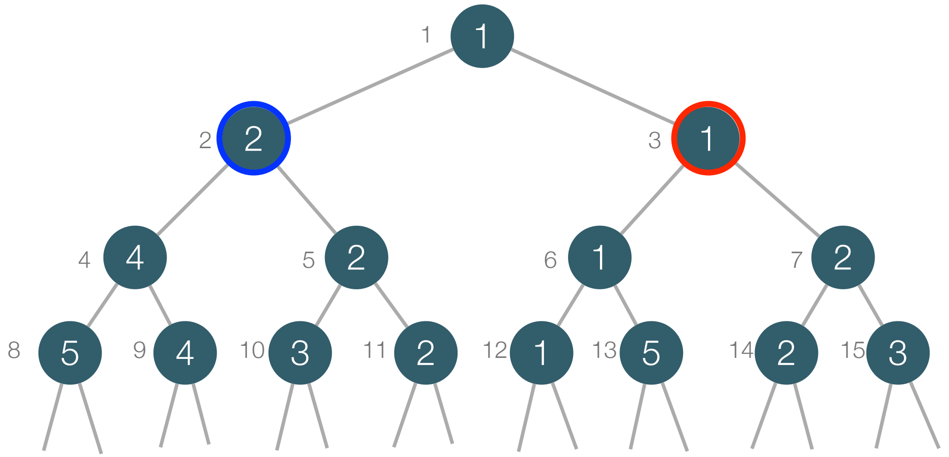
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a = $\lfloor a/2 \rfloor$, b = $\lfloor b/2 \rfloor$
return s
```

Segment trees

RMQ(3,9) = ?

s = 1



-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

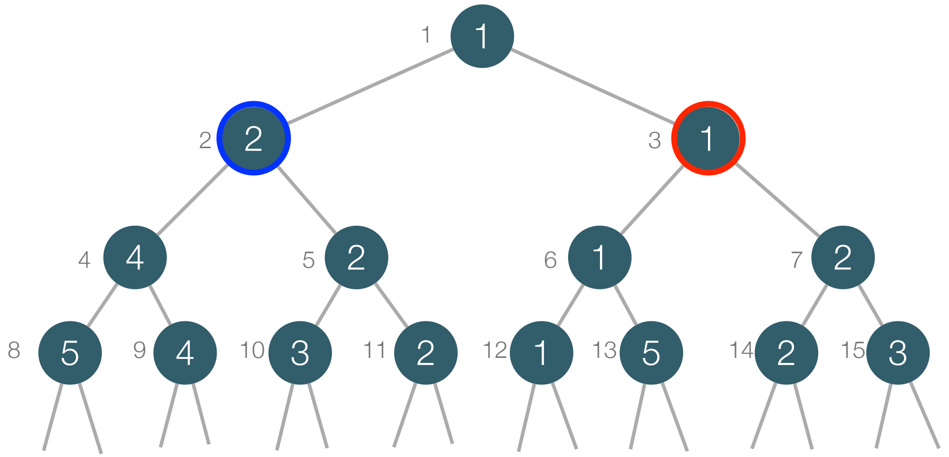
```
s = INF
a = i, b = j
while (a not right of b):
   if (a right child):
      s = min(s, tree[a])
      move a to the right
   if (b left child):
      s = min(s, tree[b])
      move b to the left
   move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
   if (a % 2 == 1):
      s = min(s, T[a])
      a = a + 1
   if (b % 2 == 0):
      s = min(s, T[b])
      b = b - 1
   a = $\lfloor a/2 \rfloor$, b = $\lfloor b/2 \rfloor$
return s
```

Segment trees

RMQ(3,9) = 1

s = 1



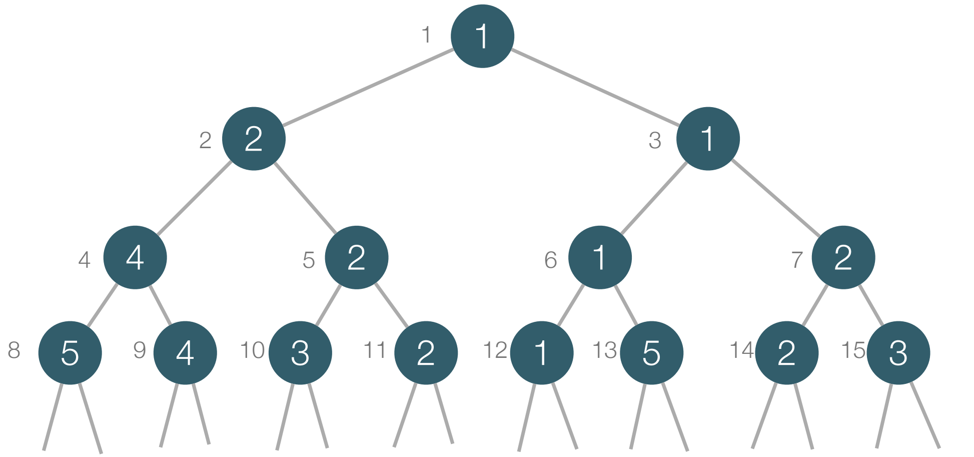
-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
s = INF
a = i, b = j
while (a not right of b):
    if (a right child):
        s = min(s, tree[a])
        move a to the right
    if (b left child):
        s = min(s, tree[b])
        move b to the left
    move a and b to parents
return s
```

```
s = INF
a = n + i, b = n+ j
while (a ≤ b):
    if (a % 2 == 1):
        s = min(s, T[a])
        a = a + 1
    if (b % 2 == 0):
        s = min(s, T[b])
        b = b - 1
    a = $\lfloor a/2 \rfloor$, b = $\lfloor b/2 \rfloor$
return s
```

Segment trees

Add(8,5)

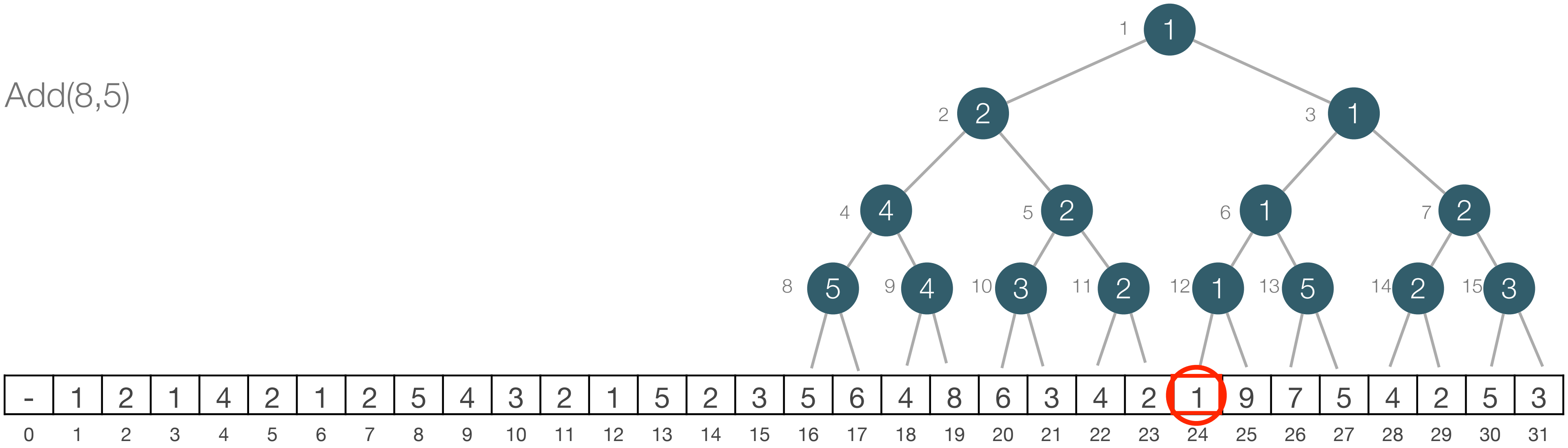


-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = ⌊x/2⌋
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = ⌊x/2⌋
```

Segment trees

Add(8,5)

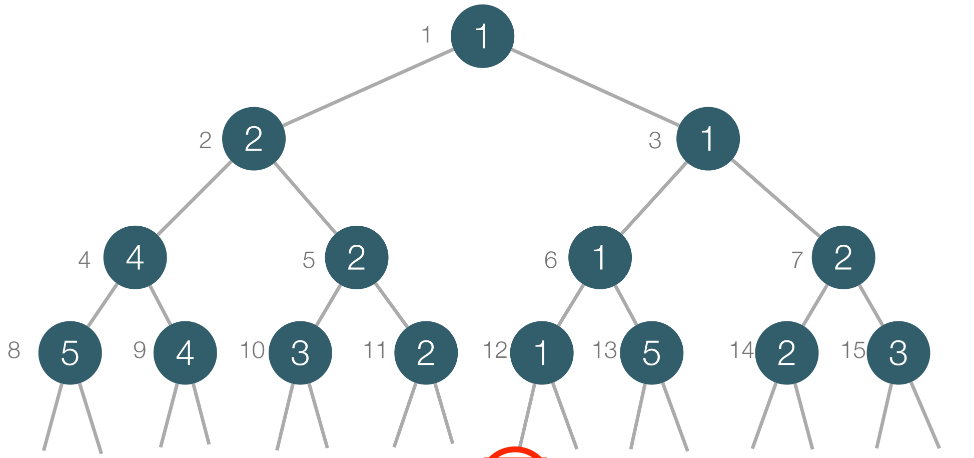


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	1	9	7	5	4	2	5	3

```
Add(i, k):
  $x = i + n$
  $T[x] = k$
  $x = \lfloor x/2 \rfloor$
  while ($x \geq 1$):
    $T[x] = \min(T[2x], T[2x+1])$
    $x = \lfloor x/2 \rfloor$
```

Segment trees

Add(8,5)

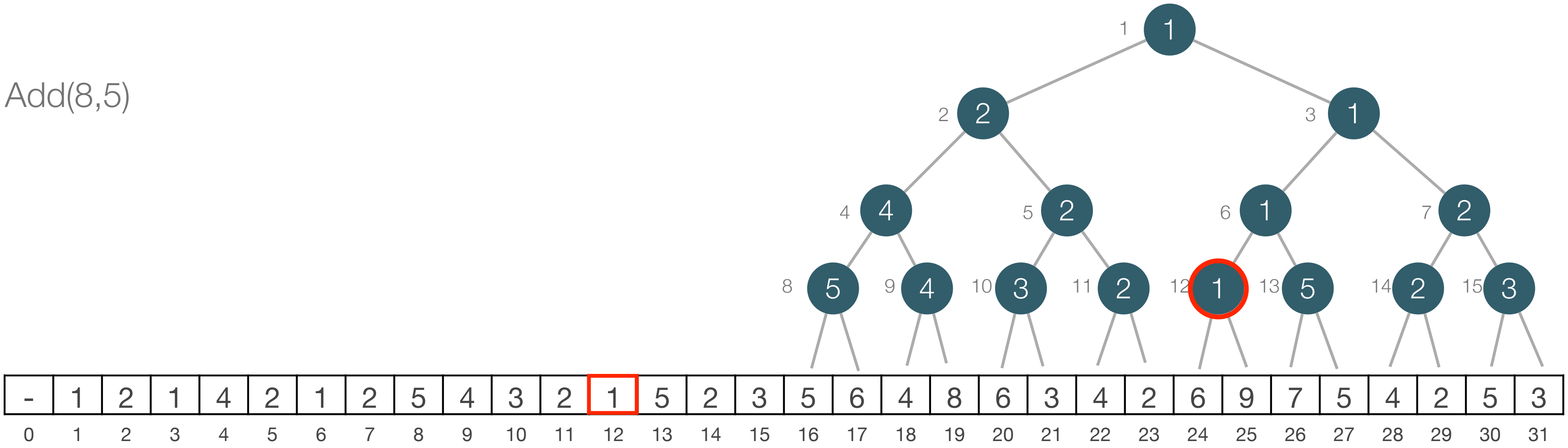


-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	6	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = ⌊x/2⌋
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = ⌊x/2⌋
```

Segment trees

Add(8,5)

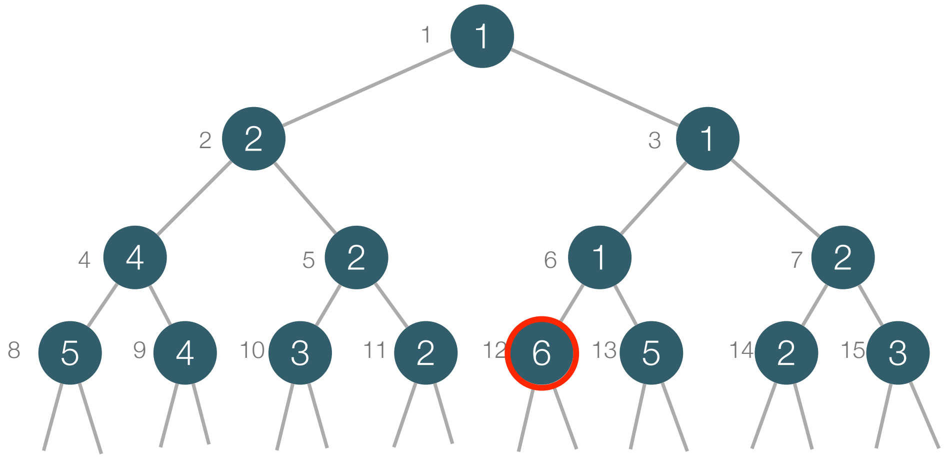


-	1	2	1	4	2	1	2	5	4	3	2	1	5	2	3	5	6	4	8	6	3	4	2	6	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = $\lfloor x/2 \rfloor$
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = $\lfloor x/2 \rfloor$
```

Segment trees

Add(8,5)

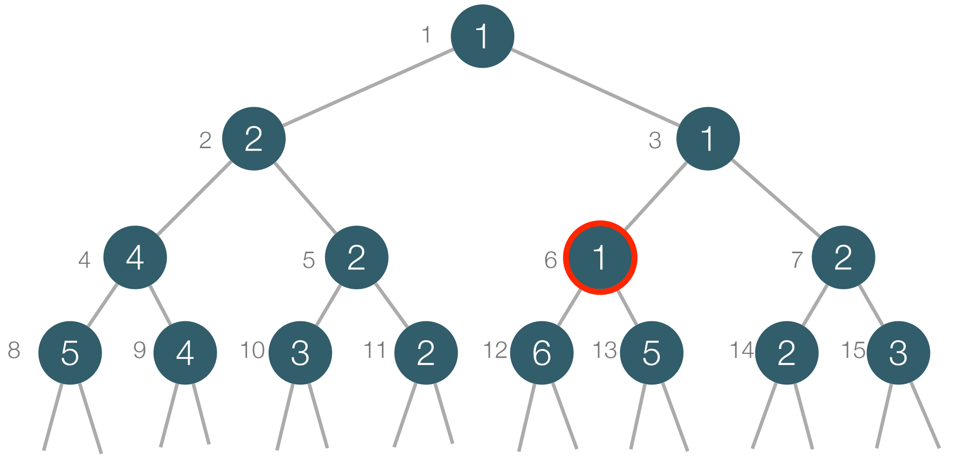


-	1	2	1	4	2	1	2	5	4	3	2	6	5	2	3	5	6	4	8	6	3	4	2	6	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = ⌊x/2⌋
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = ⌊x/2⌋
```

Segment trees

Add(8,5)

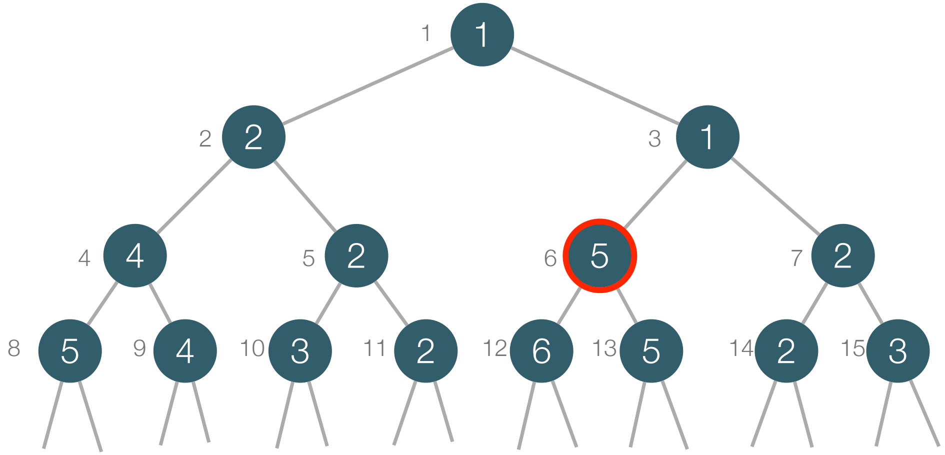


-	1	2	1	4	2	1	2	5	4	3	2	6	5	2	3	5	6	4	8	6	3	4	2	6	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = ⌊x/2⌋
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = ⌊x/2⌋
```

Segment trees

Add(8,5)

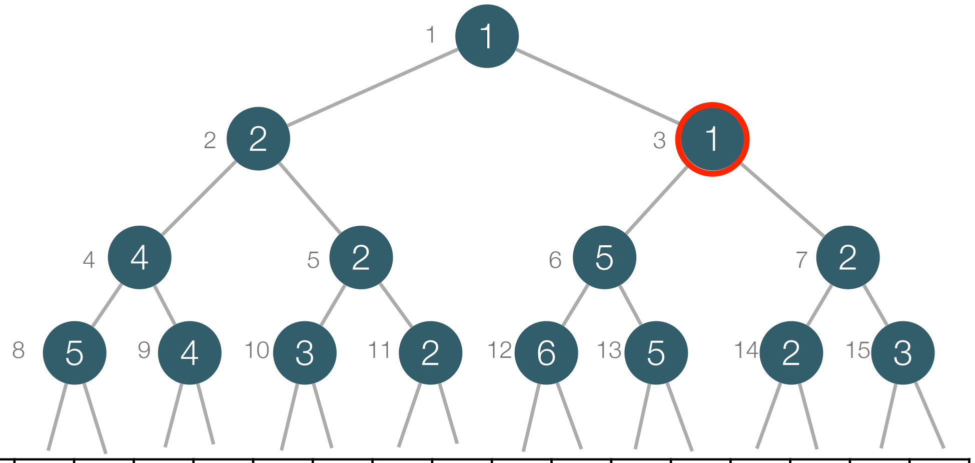


-	1	2	1	4	2	5	2	5	4	3	2	6	5	2	3	5	6	4	8	6	3	4	2	6	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = ⌊x/2⌋
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = ⌊x/2⌋
```

Segment trees

Add(8,5)

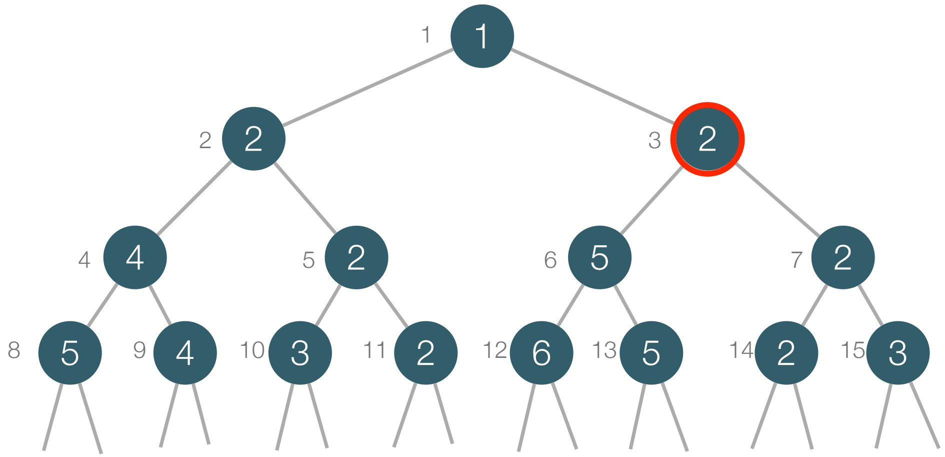


-	1	2	1	4	2	5	2	5	4	3	2	6	5	2	3	5	6	4	8	6	3	4	2	6	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = $\lfloor x/2 \rfloor$
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = $\lfloor x/2 \rfloor$
```

Segment trees

Add(8,5)

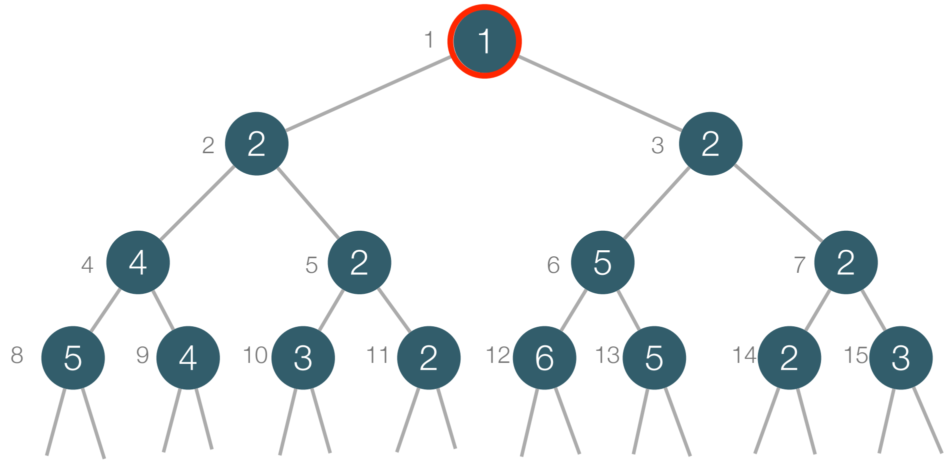


-	1	2	2	4	2	5	2	5	4	3	2	6	5	2	3	5	6	4	8	6	3	4	2	6	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = ⌊x/2⌋
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = ⌊x/2⌋
```

Segment trees

Add(8,5)

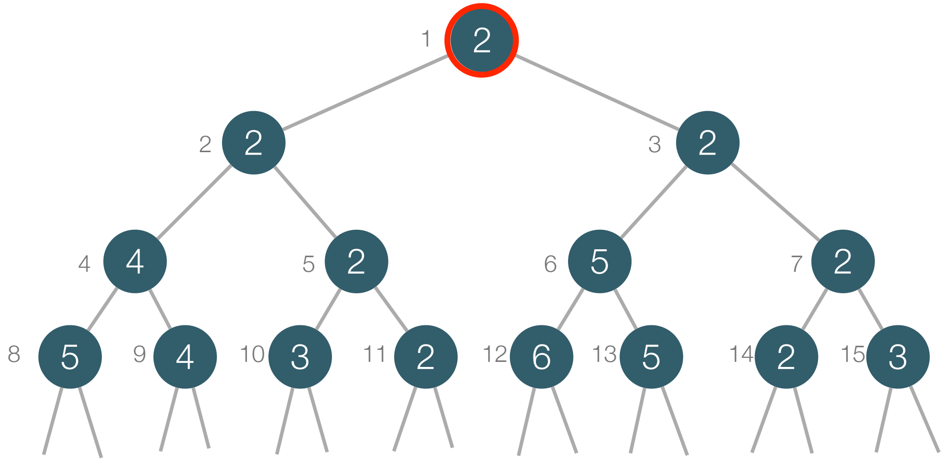


-	1	2	2	4	2	5	2	5	4	3	2	6	5	2	3	5	6	4	8	6	3	4	2	6	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = ⌊x/2⌋
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = ⌊x/2⌋
```

Segment trees

Add(8,5)

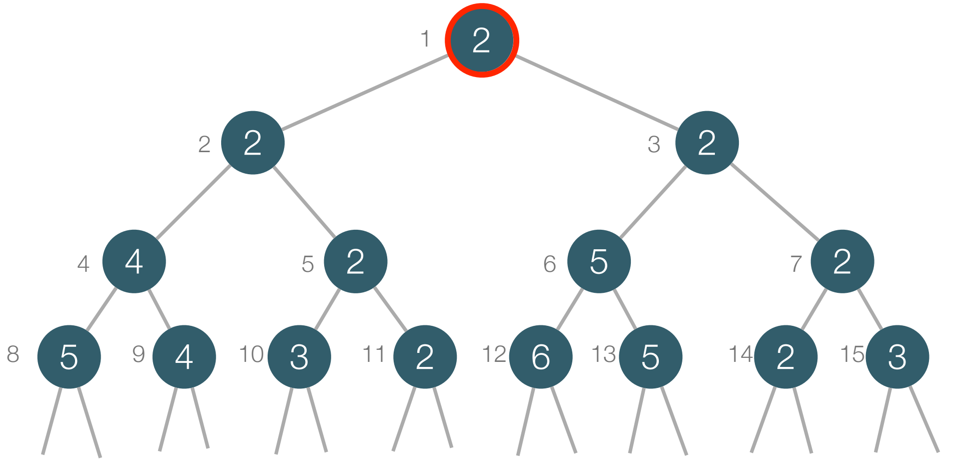


-	2	2	2	4	2	5	2	5	4	3	2	6	5	2	3	5	6	4	8	6	3	4	2	6	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = ⌊x/2⌋
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = ⌊x/2⌋
```

Segment trees

Add(8,5)



-	2	2	2	4	2	5	2	5	4	3	2	6	5	2	3	5	6	4	8	6	3	4	2	6	9	7	5	4	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

```
Add(i, k):
   x = i + n
   T[x] = k
   x = ⌊x/2⌋
   while (x ≥ 1):
      T[x] = min(T[2x], T[2x+1])
      x = ⌊x/2⌋
```