

CSC11F: Advanced Data Structures and Algorithms

Segment Tree

Yutaka Watanobe (142-A, yutaka@u-aizu.ac.jp)

Problem to be solved

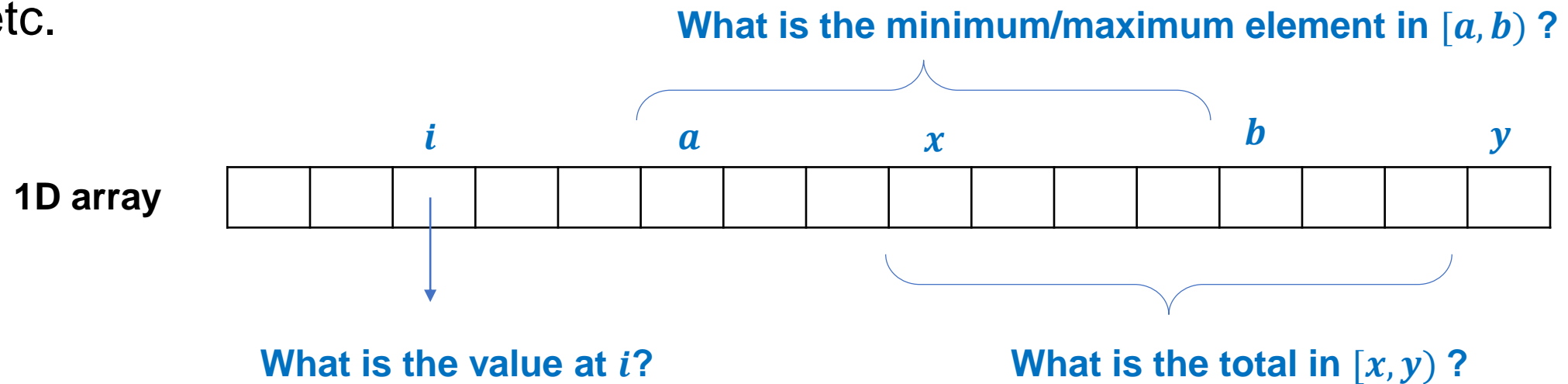
- Answering for queries to extract or update data is a fundamental operation to a data collection.
- As the terminology 'query' is used for database management system, it includes search operation (e.g. find the minimum element) and modifications (e.g. update a specified element).



- There are some possible queries for *elements* or *intervals* in an array as follows.

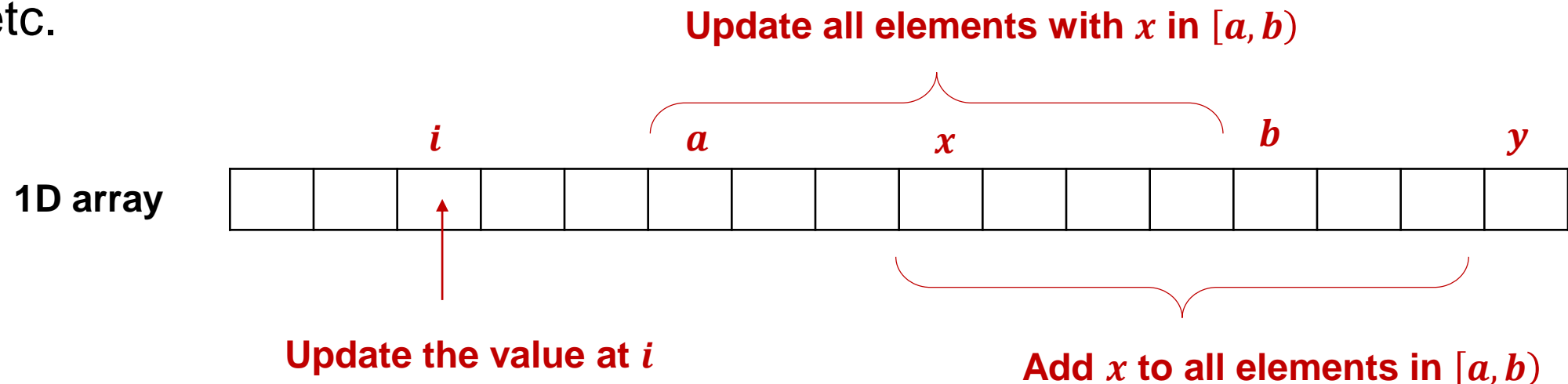
Queries to find the specified value in an array

- RMQ: Range Minimum Query. Find the minimum/maximum value in the specified range $[a, b)$.
- RSQ: Range Sum Query. Get the sum of values in the specified range $[a, b)$..
- SRQ: Single Read Query. Find the element at the specified position i .
- etc.



Queries to modify the specified elements in an array

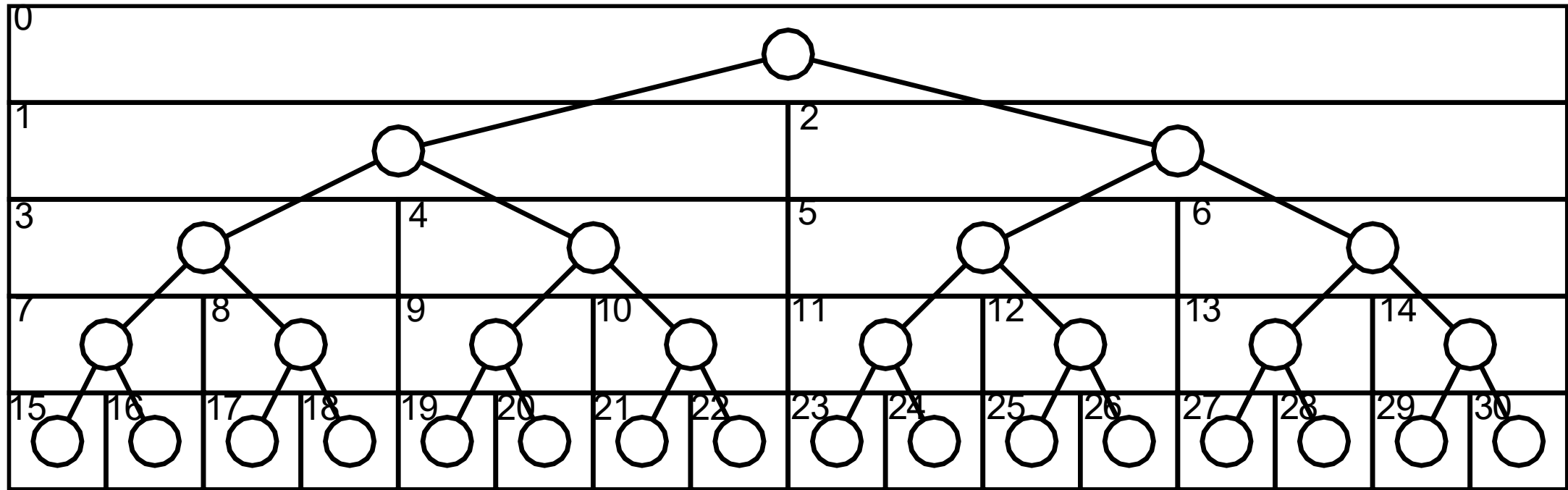
- RUQ: Range Update Query. Update the values in the specified range $[a, b)$ to x .
- RAQ: Range Add Query. Add x to elements in the specified range $[a, b)$.
- SUQ: Single Update Query. Update the value of the specified element i to x .
- etc.



Naive Algorithms

- It is easy to design and implement naive algorithms for such queries.
 - We can design $O(1)$ algorithms for reading/updating a single element, and $O(N)$ algorithms for calculating/updating values of elements in the specified range where N is the number of elements.
- On the other hand, if you should deal with many such queries (where Q is the number of queries), you should consider more efficient algorithms.
 - The algorithm should not be $O(QN)$.
- Fortunately, we can implement efficient algorithms by means of the advanced data structure called **segment trees**.

Structure of Segment Tree

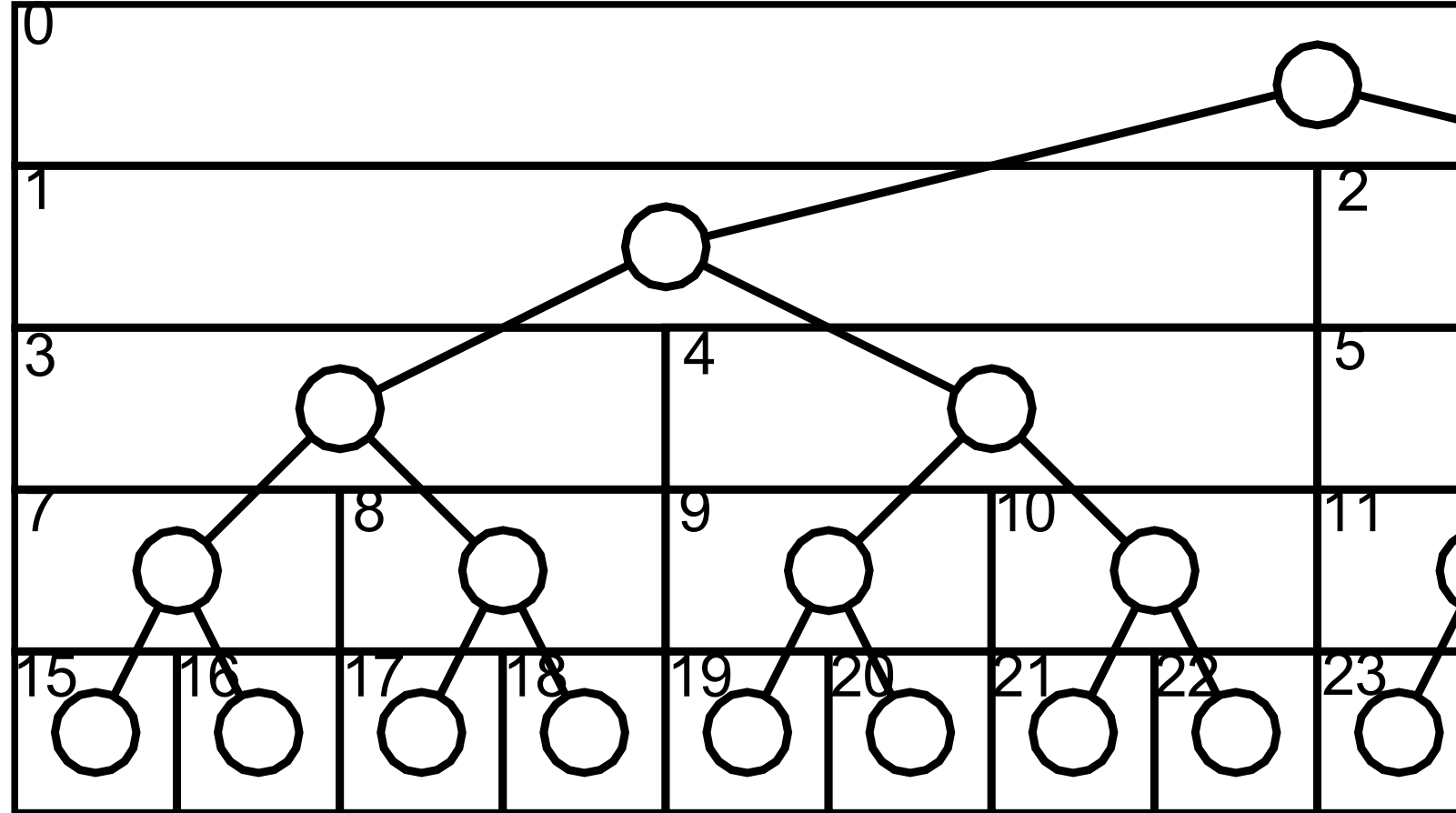


A segment tree T is a **complete binary** tree which each node represents an interval on an array.

Structure of Segment Tree

Each node of the complete binary tree is indexed as follows:

- Index of the root is 0.
- Indices of the left child and the right child of node k is $2k + 1$ and $2k + 2$ respectively.
- Index of the parent of node k is $\frac{(k-1)}{2}$



Structure of Segment Tree

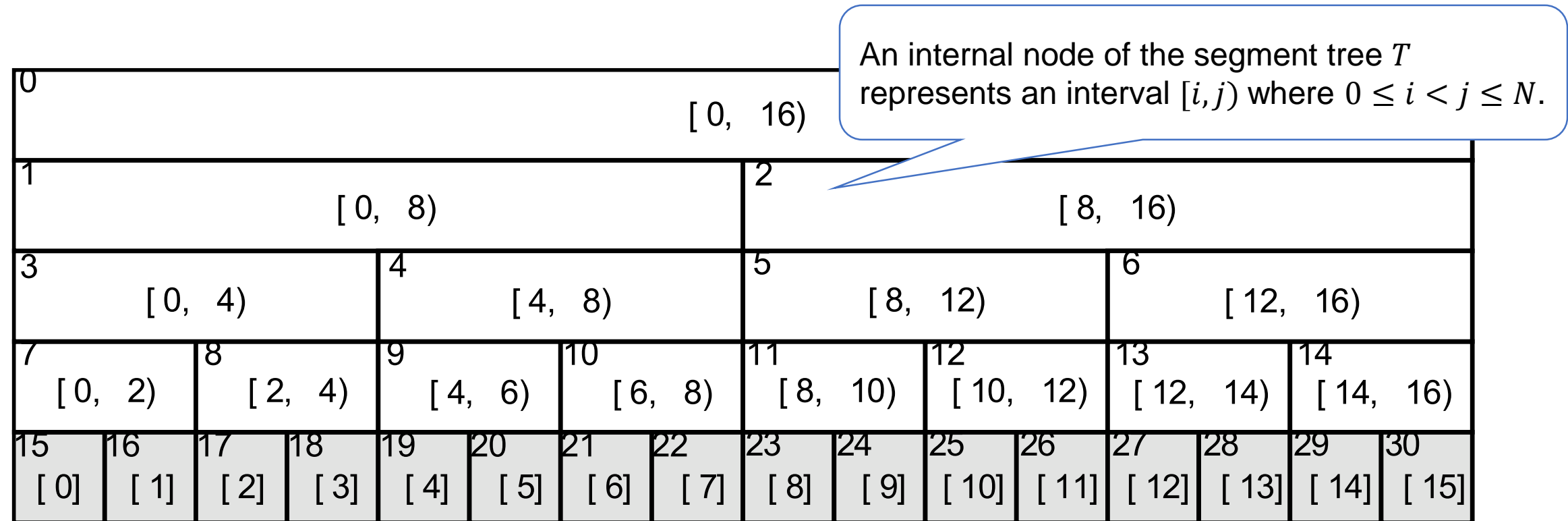
Each node of the segment tree T is to represent a single element or a sequence of elements in an interval of the array.

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

A 0-origin array A of size N is managed by a segment tree T of size $2N - 1$.

To employ a segment tree, we suppose the length of the array is a power of 2.

Structure of Segment Tree



Each leaf of the segment tree T corresponds to an element i of the array such that $0 \leq i < N$.

Structure of Segment

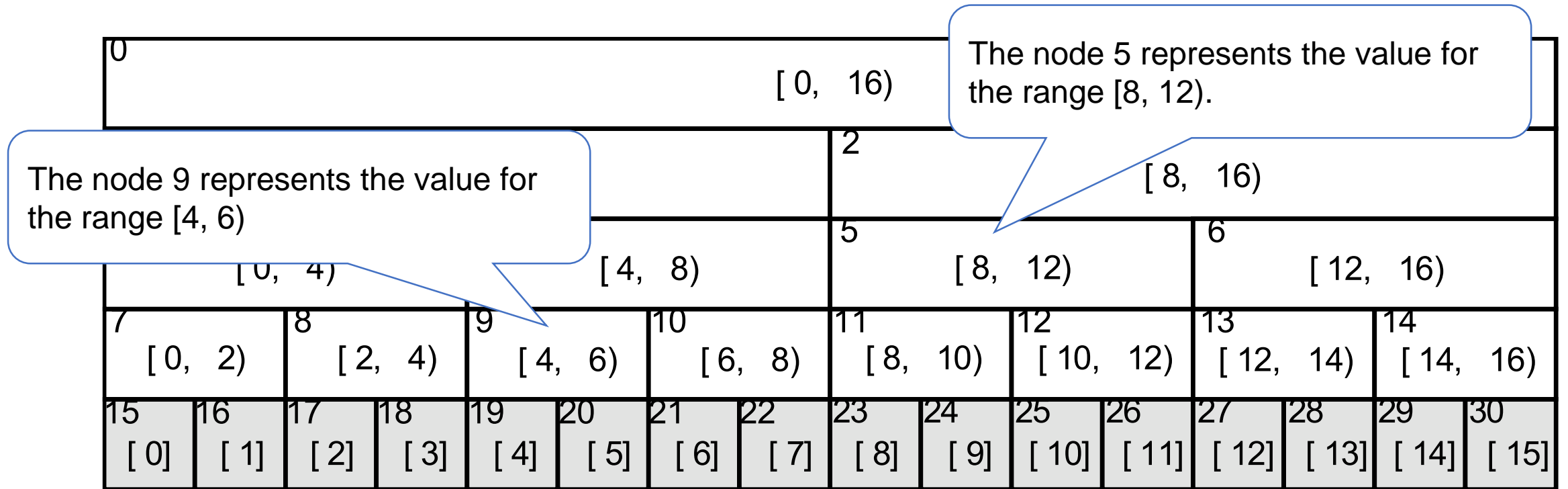
Generally, the range $[l, r)$ represented by a node is broken down into the half segments $[l, \frac{l+r}{2})$ and $[\frac{l+r}{2}, r)$ at its children respectively.

The root of the segment tree T represents the whole range $[0, N)$ of the array. The range $[0, N)$ represented by the root is broken down into two half segments $[0, \frac{0+N}{2})$ and $[\frac{0+N}{2}, N)$ at its children respectively.

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

The index of node in the segment tree for i -th element of the array is $i + N - 1$.

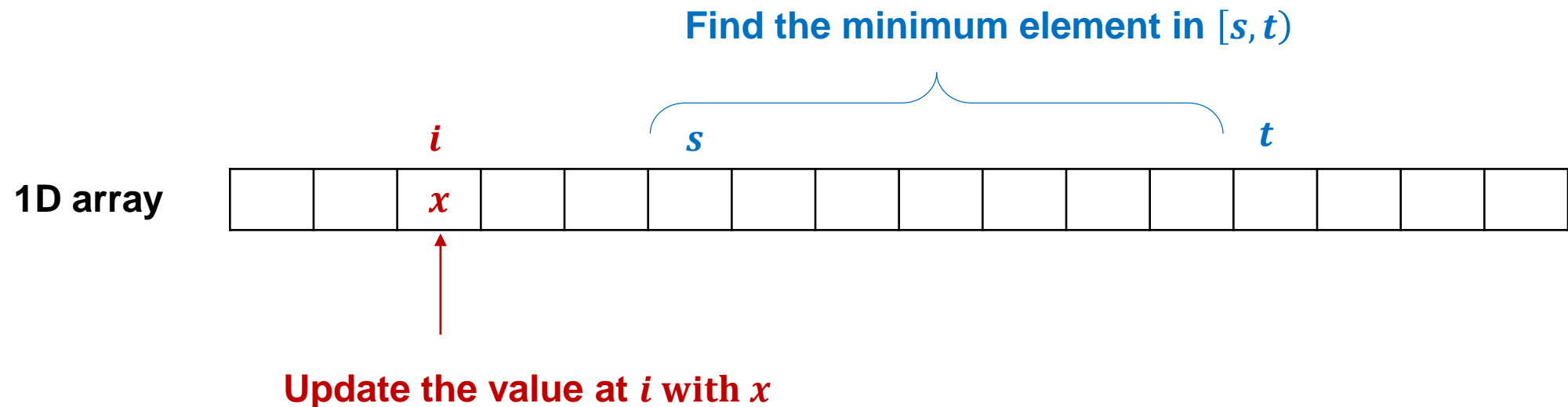
Structure of Segment Tree



RMQ and SUQ

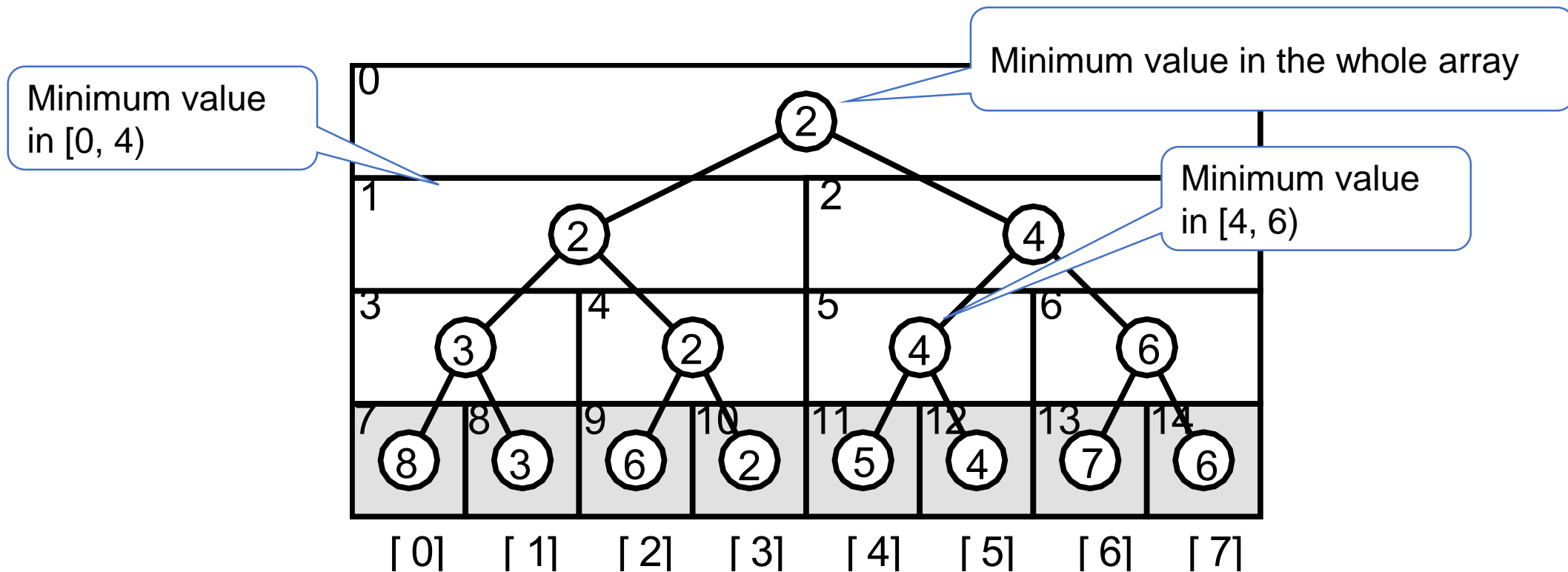
Let's consider a problem to answer for both RMQ and SUQ for a sequence $A = \{a_0, a_1, \dots, a_{n-1}\}$ with the following operations:

- $\text{find}(s, t)$: report the minimum element in a_s, a_{s+1}, \dots, a_t
- $\text{update}(i, x)$: change a_i to x .



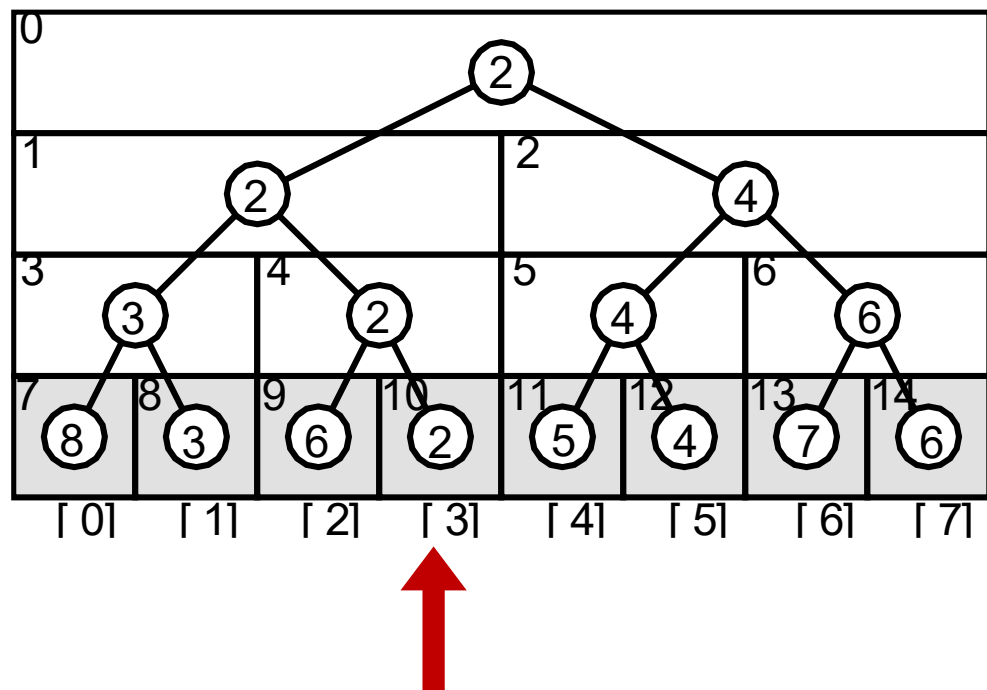
RMQ and SUQ

We should answer for RMQ, so each node of the segment tree should maintain the minimum value between the corresponding interval.



Operation for SUQ

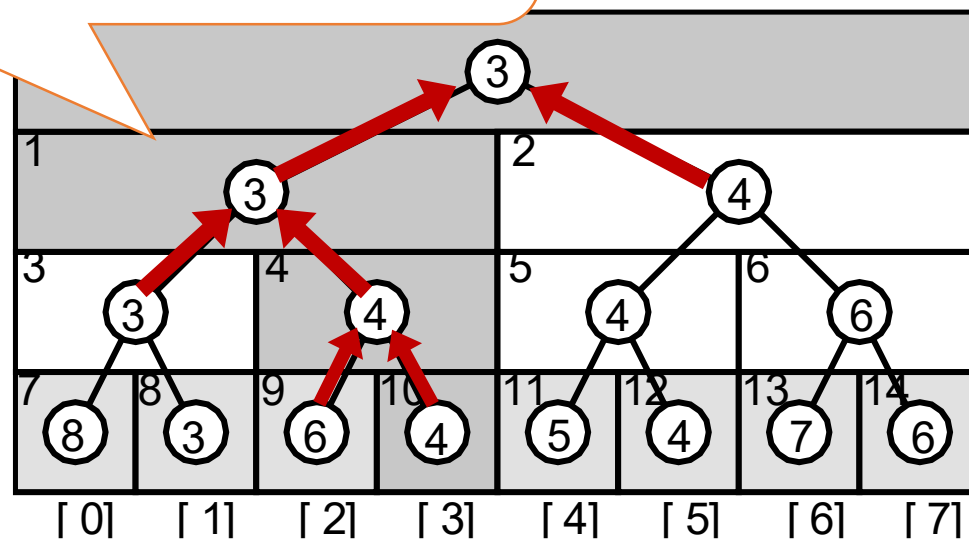
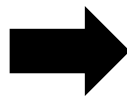
$\text{update}(i, x)$: replace $A[i]$ with x .



2

At node with index k , we take pairs of nodes with indices $(2k + 1, 2k + 2)$ and combine their values (for RMQ, select the minimum element).

$\text{update}(3, 4)$

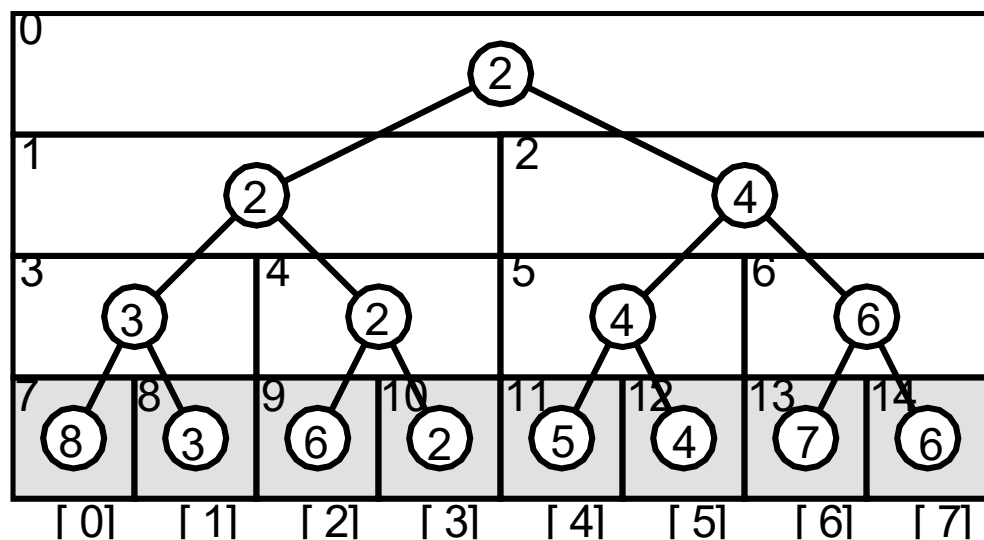


1

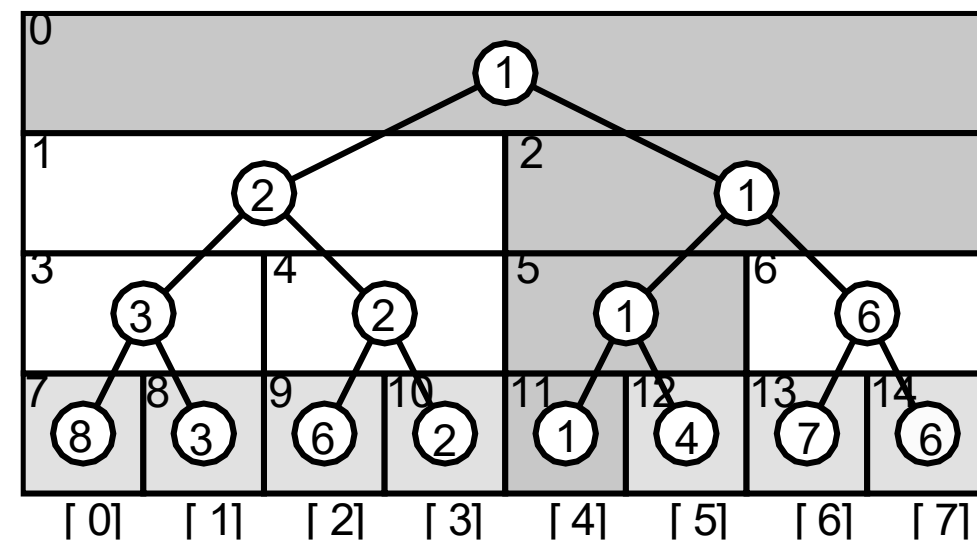
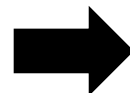
The update operation starts with the modification of the array element (a leaf of the segment tree) and it goes up the segment tree.

Operation for SUQ

Another example

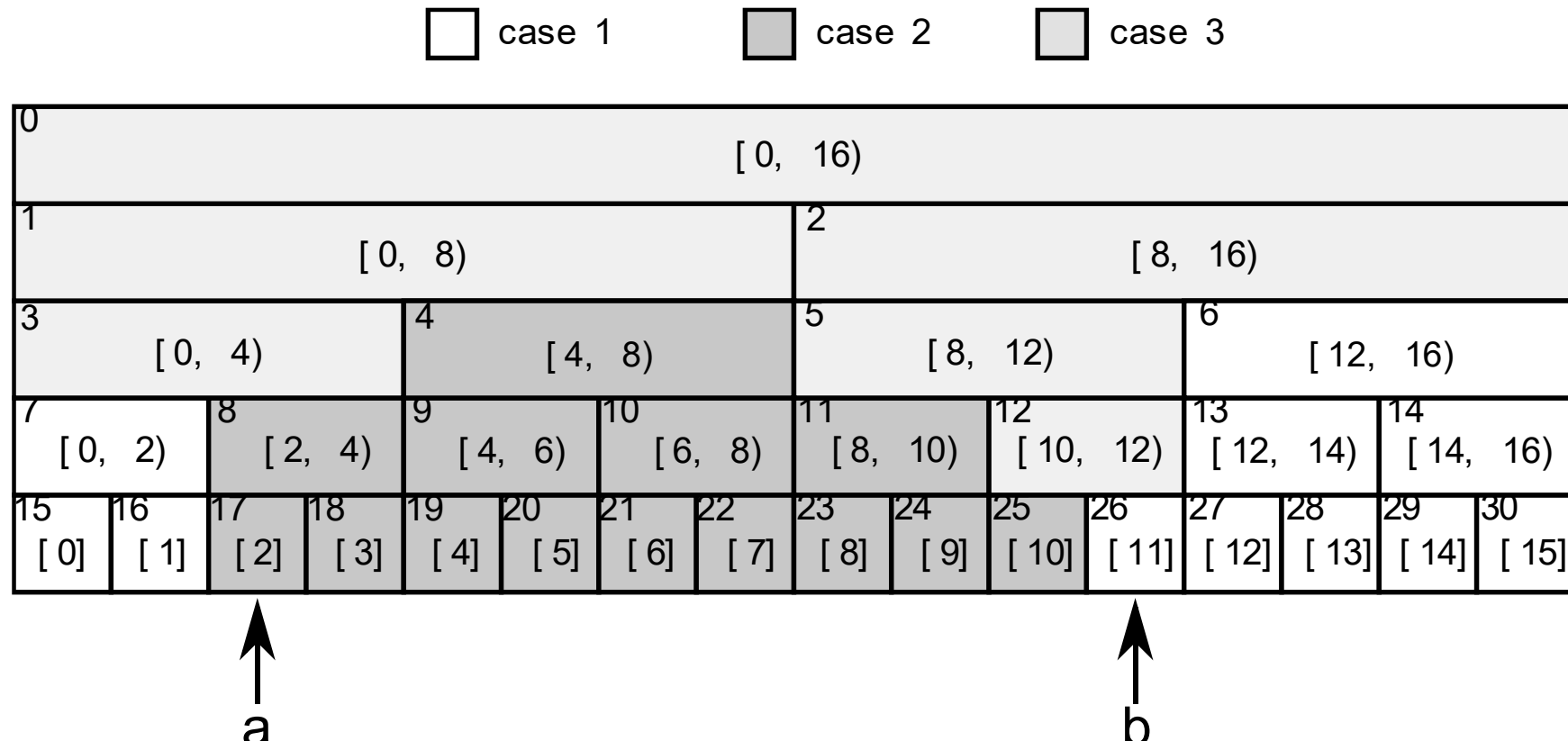


update(4, 1)



Anyway, we can perform the single update query with at most $\log N$ (the height of the tree) steps where N is the number of nodes in the segment tree.

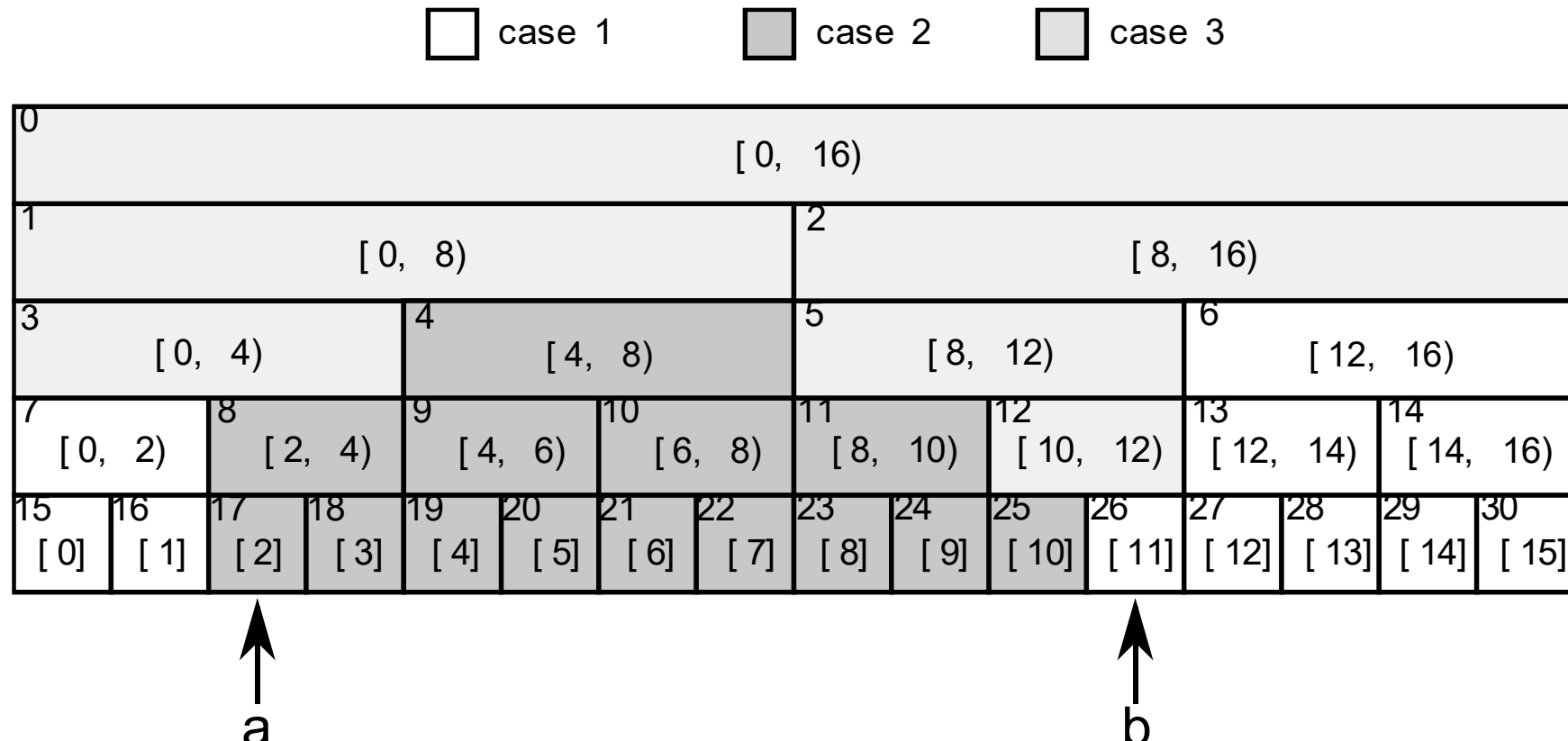
Operation for RMQ



An example of the traversal area for a range query.

For a range query, we do not need to traverse all elements of in specified interval $[a, b)$ on the array, but scan narrow downed nodes of the segment tree.

Operation for RMQ



The range query finds the solution by the ***post-order tree traversal*** starting with the root.

Suppose the range of the query is $[a, b)$ and the range of the current node during the traversal is $[l, r)$.

Operation for RMQ

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

☐ case 1 : the two ranges $[a, b)$ and $[l, r)$ do not intersect, say $r \leq a$ or $b \leq l$

we should just return a value which does not affect the solution for the query. For RMQ, we should return INF (e.g. a large integer).

Operation for RMQ


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

 case 2 : $[a, b)$ includes $[l, r)$, say $a \leq l$ and $r \leq b$

we have the solution for this range, so we can immediately return the value of the node.

Operation for RMQ

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

 case 3 : others

we can not decide the solution in the range because there are possibilities to have the solution in both children of the current node. So, we should visit the children recursively to obtain the solutions, then select and return the optimal solution for the range. For RMQ, we should select the minimum value.

Operation for RMQ

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



In this way, we can perform a range query in $O(\log N)$ where N is the number of nodes in the segment tree.

Implementation

For implementing a segment tree for RMQ, we can declare the following variables:

`n:` the size of the target array

`D[i]:` minimum values for the range represented by
node `i` of the segment tree

Implementation

```
initRMQ(n_) // input the required array size
    n = 1
    // the size of array is power of 2
    while n < n_:
        n *= 2
    // initialize all node in the segment tree
    for i = 0 to 2*n-1-1:
        D[i] = INF
```

Implementation

```
update(k, x)
    k += n - 1 // convert array index to tree index
    D[k] = x
    while k > 0:
        k = (k - 1) / 2
        D[k] = min(D[k*2+1], D[k*2+2])

// [a, b]
findMin(a, b)
    return query(a, b+1, 0, 0, n)
```


Implementation

```
// [a, b)
query(a, b, k, l, r)
    if r <= a || b <= l: // case 1
        return INF
    if a <= l && r <= b: // case 2
        return D[k]
    // case 3
    vl = query(a, b, k*2 + 1, l, (l + r)/2)
    vr = query(a, b, k*2 + 2, (l + r)/2, r)
    return min(vl, vr)
```

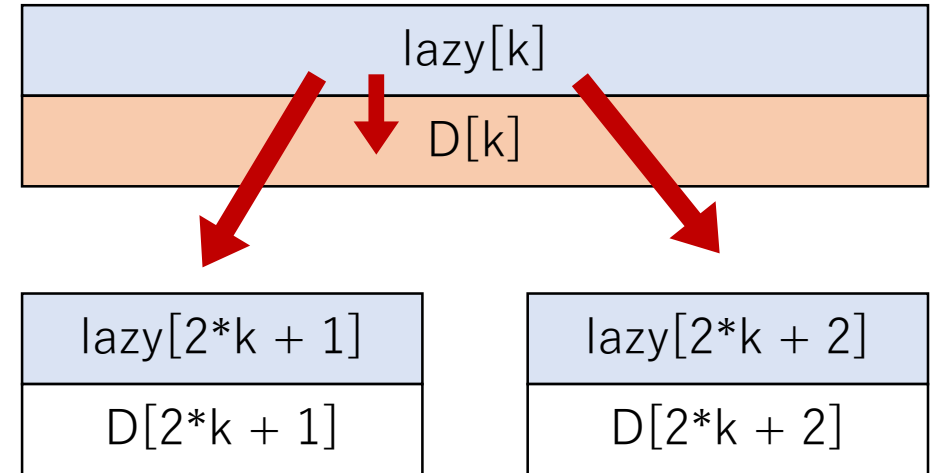
Lazy Evaluation

- To implement an efficient segment tree, for some cases (e.g. RMQ + RUQ), we need a technique called lazy evaluation.
- In addition to $D[i]$, let $lazy[i]$ be a value which is equally updated (or added in case of range add) for the corresponding segment but has not been reflected to $D[i]$.
- At the appropriate time, $lazy[k]$ is expanded.

Lazy Evaluation

```
lazy_evaluate(k)
    if lazy[k] == NUL return
    D[k] = lazy[k]
    if k < n - 1: // not leaf
        lazy[2*k + 1] = lazy[k]
        lazy[2*k + 2] = lazy[k]
    lazy[k] = NUL
```

(local view)

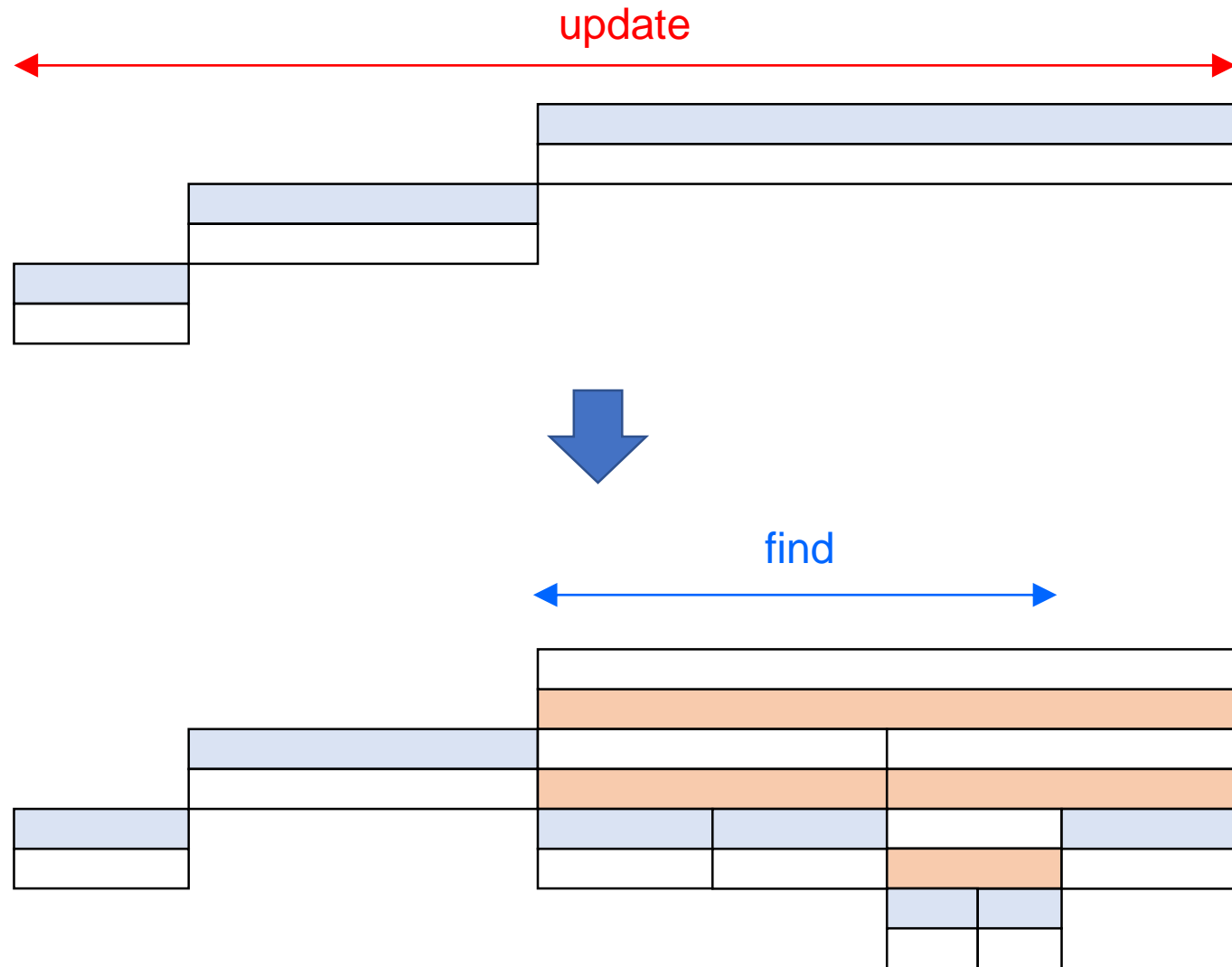


`lazy_evaluate(k)` should be activated before the corresponding segment which is represented by k , is updated or queried.

Lazy Evaluation

(global view)

lazy
D



Lazy Evaluation

You can implement the query function with the lazy evaluation as follows (for RMQ).

```
query(a, b, k, l, r)
    if r <= a || b <= l: // case 1
        return INF

    lazy_evaluate(k)
    if a <= l && r <= b: // case 2
        return D[k]
    // case 3
    vl = query(a, b, k*2 + 1, l, (l + r)/2)
    vr = query(a, b, k*2 + 2, (l + r)/2, r)
    return min(vl, vr)
```

Lazy Evaluation

You can implement the update function with the lazy evaluation as follows (for RMQ).

```
update(a, b, k, l, r, x)
    lazy_evaluate(k)
    if r <= a || b <= l:
        return
    if a <= l && r <= b:
        lazy[k] = x
    else
        update(a, b, k*2+1, l, (l+r)/2, x)
        update(a, b, k*2+2, (l+r)/2, r, x)
        D[k] = min(D[k*2+1], D[k*2+2])
    lazy_evaluate(k)
```

Lazy Evaluation

- In this implementation, you should perform the lazy evaluation at both the beginning and the end of the update function.
- The first one is to propagate the lazy evaluation. On the other hand, we need the second one to use reflected values of $D[k*2+1]$ and $D[k*2+2]$.
- We can perform both RMQ and RUQ in $O(\log N)$.