Struktur Data

Binary Indexed Tree or Fenwick Tree

Range Minimum Query

Segment Tree

Problems

Diketahui sebuah arr[0 . . . n-1].

Lakukan:

1. Penjumlahan terhadap i elemen pertama (prefixsum).
2. Ubah nilai dari array ke-i, arr[i] = x, dimana 0 <= i <= n-1 (update).

arr	0	1	2	3	4	5	6	7	8	9
	20	50	30	10	40	60	70	100	80	90

Simple Solution 1

```
for(j=0;j<i;j++){
    sum+=arr[j];
}
```

→ O(n)

```
arr[0] = 25
```

→ O(1)

arr

0	1	2	3	4	5	6	7	8	9
20	50	30	10	40	60	70	100	80	90

arr

0	1	2	3	4	5	6	7	8	9
25	50	30	10	40	60	70	100	80	90

Simple Solution 2

arr

0	1	2	3	4	5	6	7	8	9
20	50	30	10	40	60	70	100	80	90

```
for(j=0;j<i;j++){
      sum+=arr[j];
      arr2[j]=sum;
}
```

arr2

0	1	2	3	4	5	6	7	8	9
20	70	100	110	150	210	280	380	460	550

1. Penjumlahan terhadap i elemen pertama. → O(1)
2. Ubah nilai dari array ke-i, arr[i] = x, dimana 0<= i <= n-1. → O(n)

   Nilai yang diubah tdk hanya array ke-i, tetapi juga array setelahnya

Apakah mungkin mendapatkan O(log n) untuk kedua operasi tersebut?

Gunakan data structure:

- Binary Indexed Tree / Fenwick Tree
  - Tinggi tree: log n
  - Query Prefix sum : O(log n)
  - Update : O(log n)

Binary Indexed Tree / Fenwick Tree

- Ide:
  - Setiap index akan menyimpan “partial sum” yang bervariasi, sehingga updatenya juga hanya akan mencakup sebagian saja, tidak semua.
  - Total sum didapat dengan menelusuri tree dari leaf ke root
  - Gunakan representasi biner :
  - Ex: sum(13)

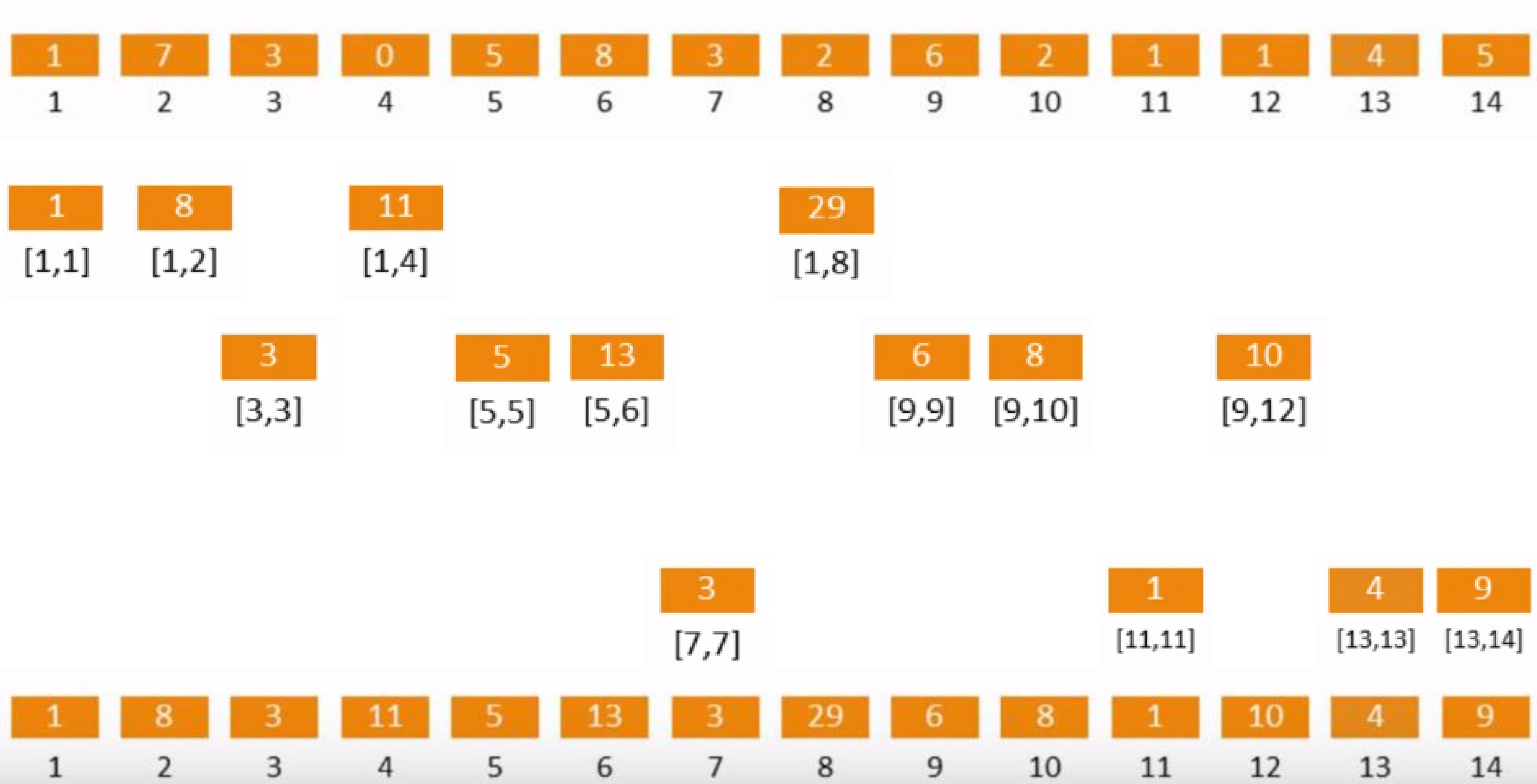
$$13 = 2^3 + 2^2 + 2^0$$

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

sum(13) = range(1,8) + range(9,12) + range(13,13)

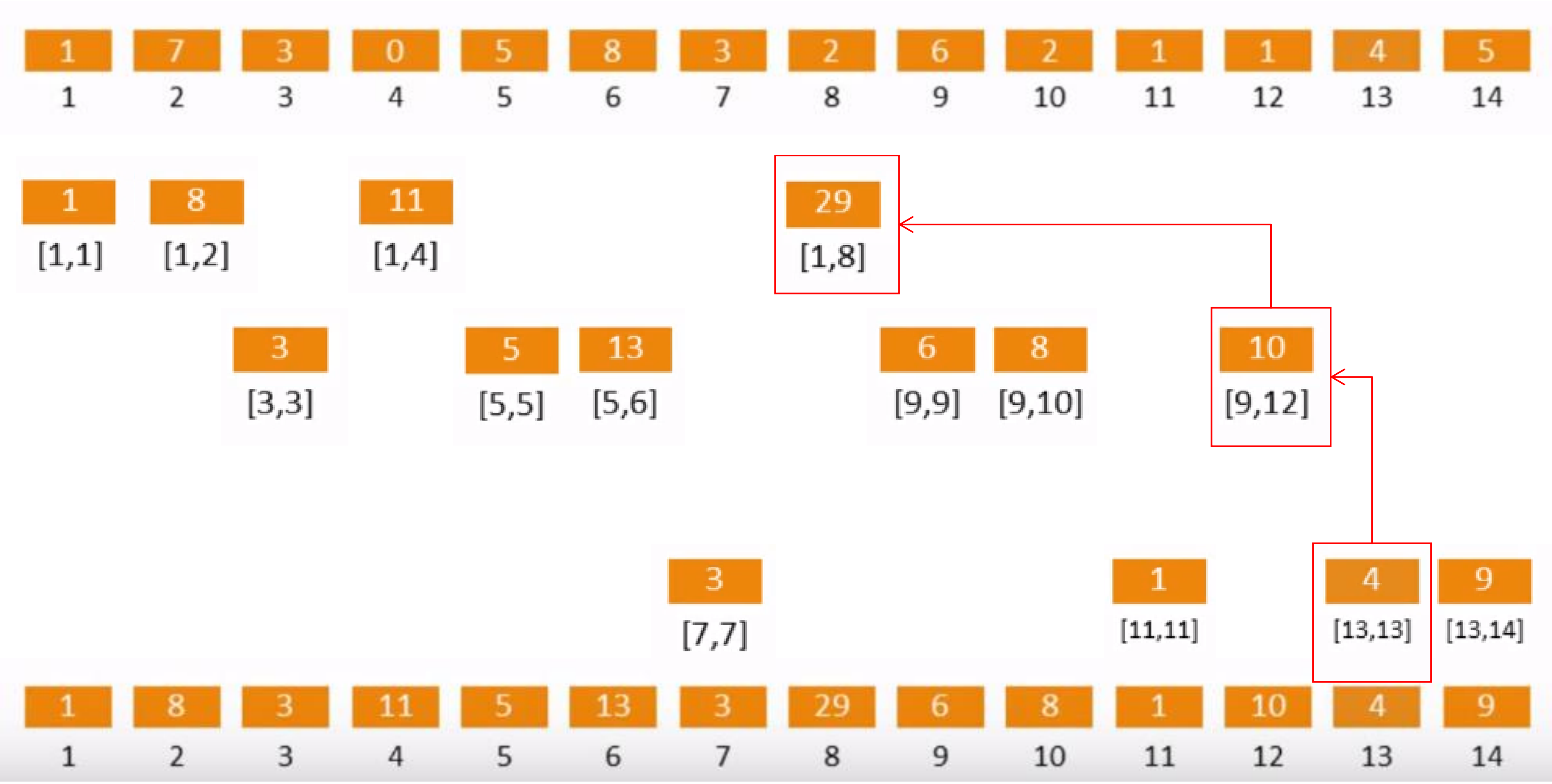
= 29 + 10 + 4 = 43

Binary Indexed Tree Construction



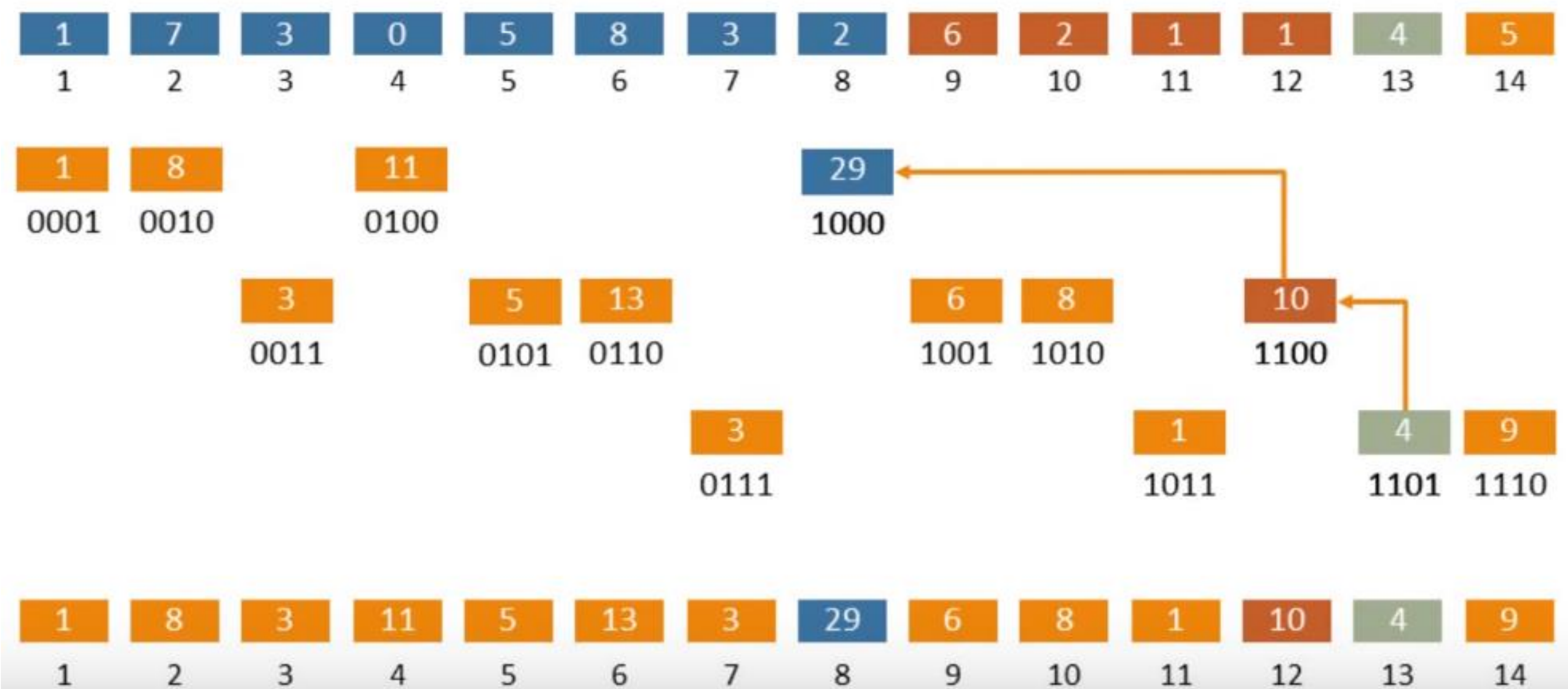
Binary Indexed Tree PrefixSum

sum(13) = range(1,8) + range(9,12) + range(13,13)



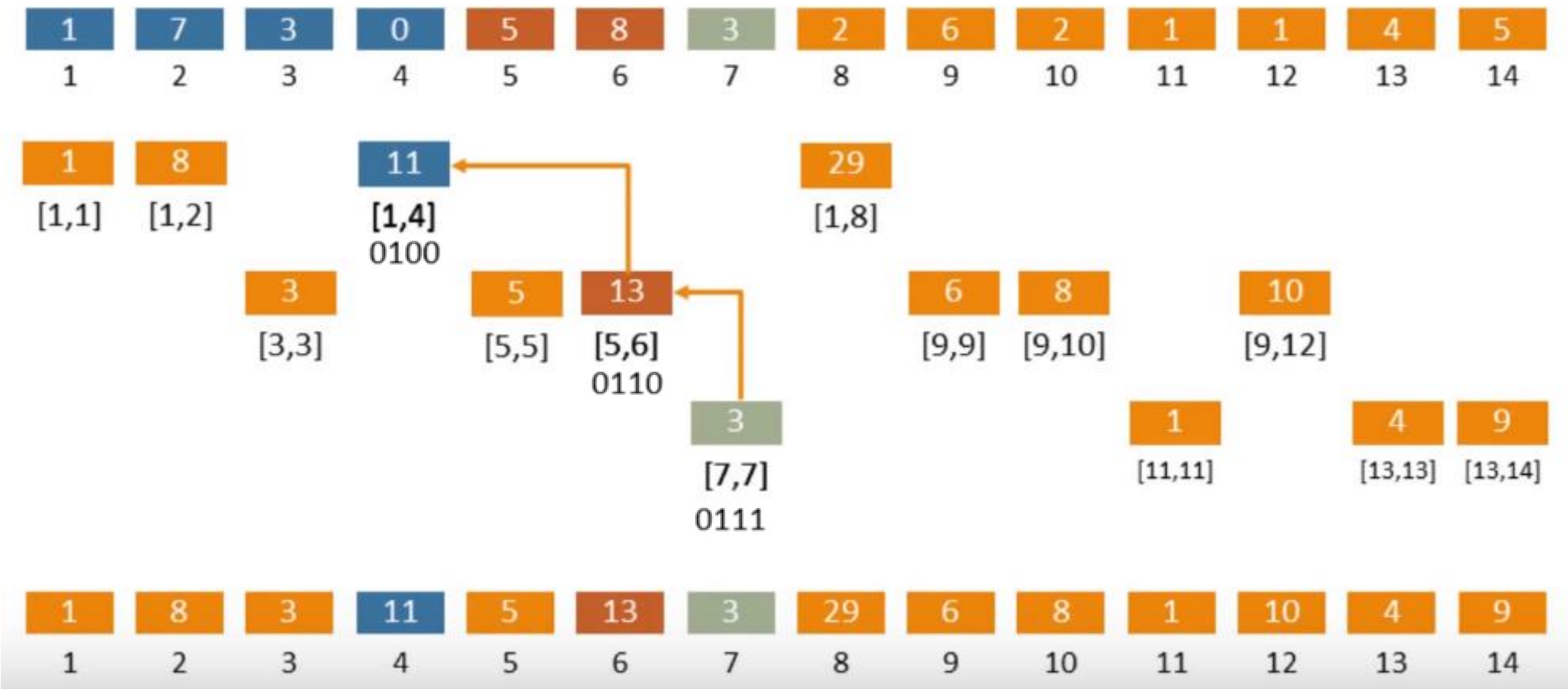
Binary Indexed Tree PrefixSum

sum(13) = range(1,8) + range(9,12) + range(13,13)



Binary Indexed Tree PrefixSum

sum(7) = range(1,4) + range(5,6) + range(7,7)



Binary Indexed Tree GetParents

- Extract last set bit: $x\ \&\ (-x)$
- Remove it: $x - (x\ \&\ (-x))$

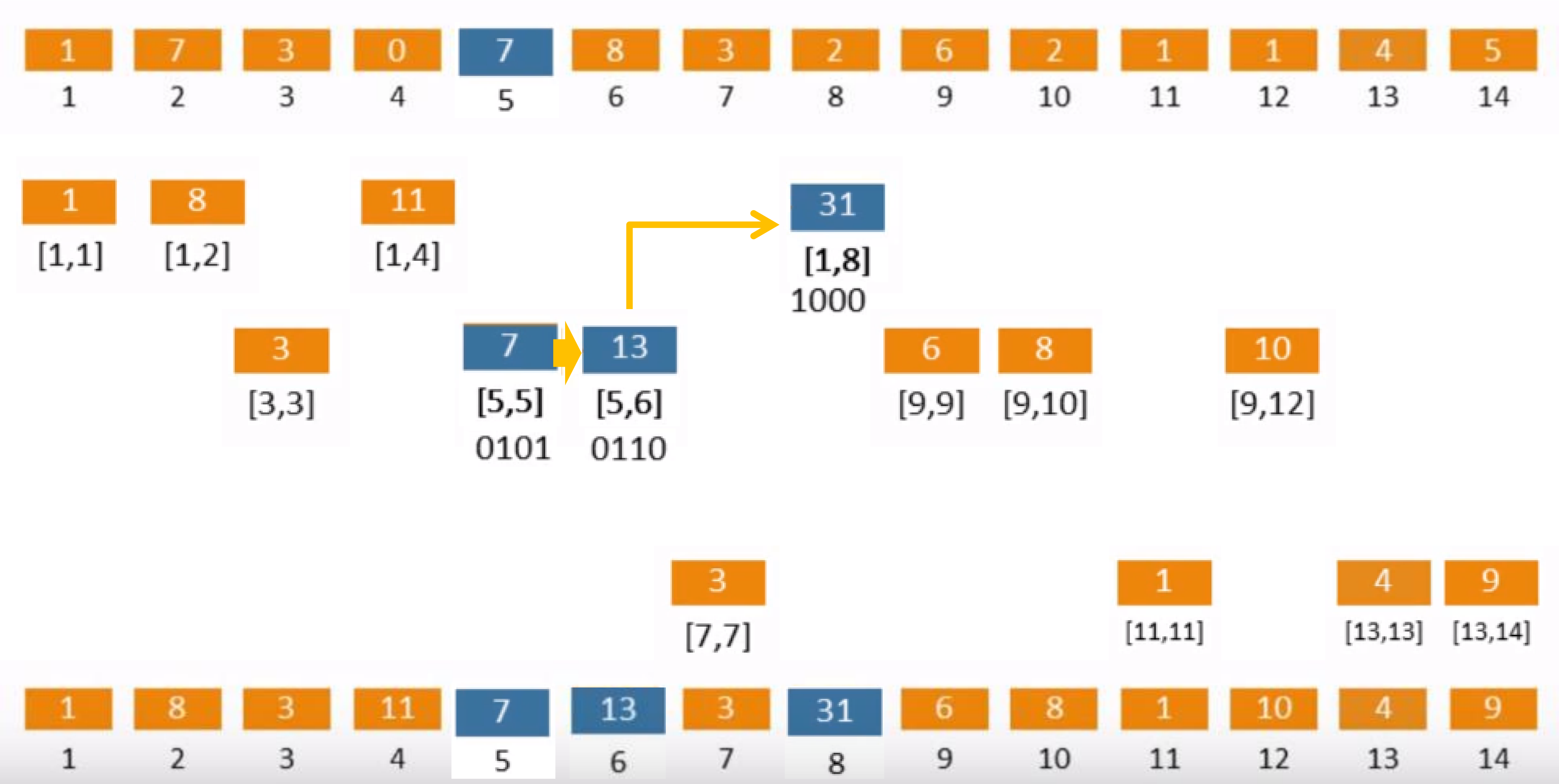
$$x \ = \ 13 \ = (00001101)_2$$
$$-x = -13 = (11110011)_2$$
$$x\ \&\ (-x) = (00000001)_2$$
$$x \ - \ \big(x\ \&\ (-x)\big) = (00001100)_2$$

Binary Indexed Tree Sum

```
def sum(bit_arr, idx):
    result = 0
    while idx:
        result += bit_arr[idx]
        idx -= idx & -idx
    return result
```

Binary Indexed Tree ADD

ADD (5,2)



Binary Indexed Tree ADD

- Extract last set bit: $x \,\&\, (-x)$
- Adding it: $x + (x \,\&\, (-x))$

```
def add(bit_arr, idx, val):
    while idx < len(bit_arr):
        bit_arr[idx] += val
        idx += idx & -idx
```

Segment Tree

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

- Min (2,4) → 0
- Min (0,3) → -1
- Max (1,5) → 4
- Max (0,3) → 4
- Sum (0,2) → 6
- Sum (4,5) → 3

Range Minimum Query (RMQ)

Segment Tree

0	1	2	3	4	5	6	7
1	5	3	7	3	2	5	7

Minimum in range [1,7) :

1	2	3	4	5	6
5	3	7	3	2	5

→ 2

Update value at index 5 with 6:

0	1	2	3	4	5	6	7
1	5	3	7	3	6	5	7

Minimum in range [3,8) :

3	4	5	6	7
7	3	6	5	7

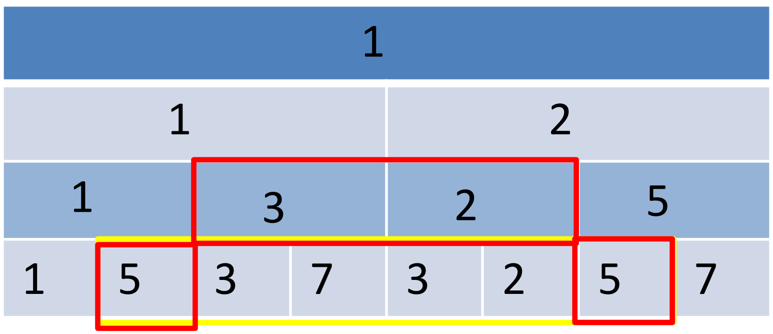
→ 3

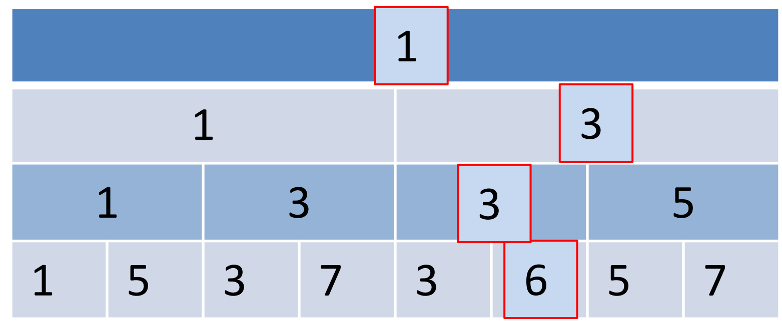
Compute MRQ : O(n)
Compute update : O(1)

→ Segment Tree →

Compute MRQ : O(log n)
Compute update : O(log n)

Segment Tree



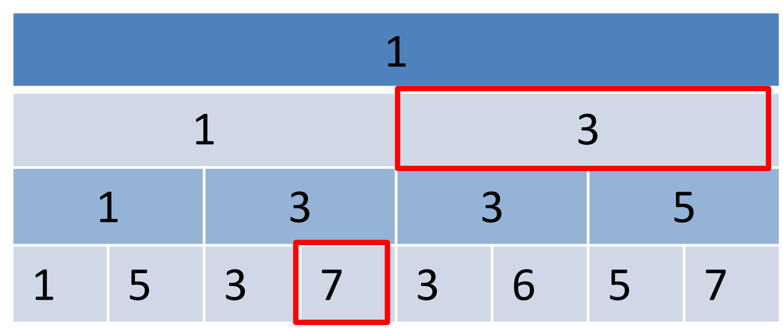


Secara eksplisit menyimpan informasi minimum pada range tertentu

Minimum in range [1,7) : 2

Update value at index 5 with 6

Minimum in range [3,8) : 3



Segmen Tree Construction

<table>
<tr><td colspan="8">1: [0, 8)</td></tr>
<tr><td colspan="4">2: [0, 4)</td><td colspan="4">3: [4, 8)</td></tr>
<tr><td colspan="2">4: [0, 2)</td><td colspan="2">5: [2, 4)</td><td colspan="2">6: [4, 6)</td><td colspan="2">7: [6, 8)</td></tr>
<tr><td>8: 0</td><td>9: 1</td><td>10: 2</td><td>11: 3</td><td>12: 4</td><td>13: 5</td><td>14: 6</td><td>15: 7</td></tr>
</table>

- Index root start at 1
- The child nodes of nodes idx:
  - 2*idx (left child) and 2*idx+1 (right child)
- The parent of the node at idx is idx/2

Algorithm 1 Construction of Segment-Tree

```
procedure CONSTRUCTION(arr)
    n ← length of arr
    data ← array of length 2 · n
    copy arr to second half of data
    for idx = n − 1 . . . 1 do
        data[idx] ← min(data[2 · idx], data[2 · idx + 1])
```

Segmen Tree Update

<table>
<tr><td colspan="8">1: $[0, 8)$</td></tr>
<tr><td colspan="4">2: $[0, 4)$</td><td colspan="4">3: $[4, 8)$</td></tr>
<tr><td colspan="2">4: $[0, 2)$</td><td colspan="2">5: $[2, 4)$</td><td colspan="2">6: $[4, 6)$</td><td colspan="2">7: $[6, 8)$</td></tr>
<tr><td>8: 0</td><td>9: 1</td><td>10: 2</td><td>11: 3</td><td>12: 4</td><td>13: 5</td><td>14: 6</td><td>15: 7</td></tr>
</table>

Algorithm 2 Update of Segment-Tree

procedure $\textsc{Update}(idx,\ value)$

  $idx \leftarrow idx + n$

  $data[idx] \leftarrow value$

  while $idx > 1$ **do**

    $idx \leftarrow idx/2$

    $data[idx] \leftarrow \min(data[2 \cdot idx], data[2 \cdot idx + 1])$

Segment Tree MRQ

1: $[0,8)$							
2: $[0,4)$				3: $[4,8)$			
4: $[0,2)$		5: $[2,4)$		6: $[4,6)$		7: $[6,8)$	
8: 0	9: 1	10: 2	11: 3	12: 4	13: 5	14: 6	15: 7

Algorithm 3 Compute minimum of range $[left, right)$

procedure MINIMUM($left$, $right$)
  $left \leftarrow left + n$, $right \leftarrow right + n$
  $minimum \leftarrow \infty$
  while $left < right$ **do**
    if $left$ is odd **then**
      $minimum \leftarrow \min(minimum, data[left])$
      $left \leftarrow left + 1$
    if $right$ is odd **then**
      $right \leftarrow right - 1$
      $minimum \leftarrow \min(minimum, data[right])$
    $left \leftarrow left/2$, $right \leftarrow right/2$
  return $minimum$

1							
1				3			
1		3		3		5	
1	5	3	7	3	6	5	7

Minimum in range [3,8) : ?