# LECTURE 8

## SEGMENT TREE

CS200

3/25 2022

BOSTON UNIVERSITY

# Segment Tree

# Naive

- Consider the following:
- Given an array [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
- Find the sum between index 2 and index 5

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

- Every query - O(n)

# Prefix Sum

- Given an arr = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
- cumulative[0] = arr[0]
- cumulative[1] = cumulative[0] + arr[1] ...
- cumulative = [0,1,3,6,10,15,21,28,36,45,55,66,78,91,105,120]
- Sum between index 2 and 5: 2 + 3 + 4 + 5 = 14
- cumulative[5] - cumlative[2-1] = 14

$$0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120$$

- Pre-compute O(n), Every query O(1)

# Update: Prefix Sum

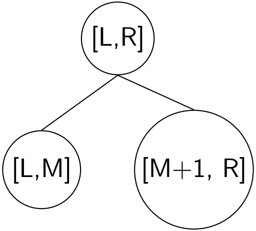
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

0, 1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 18

- Recompute the array: O(n)

# Segment Tree (Interval Tree)

Query "optimal" answer in some interval in logarithmic time.



# Perfect Balanced Tree

Let A be our array and its length is a power of 2.

- A = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]

<table>
<tr><td colspan="16">1: [0, 16)</td></tr>
<tr><td colspan="8">2: [0, 8)</td><td colspan="8">3: [8, 16)</td></tr>
<tr><td colspan="4">4: [0, 4)</td><td colspan="4">5: [4, 8)</td><td colspan="4">6: [8, 12)</td><td colspan="4">7: [12, 16)</td></tr>
<tr><td colspan="2">8:<br>[0, 2)</td><td colspan="2">9:<br>[2, 4)</td><td colspan="2">10:<br>[4, 6)</td><td colspan="2">11:<br>[6, 8)</td><td colspan="2">12:<br>[8, 10)</td><td colspan="2">13:<br>[10, 12)</td><td colspan="2">14:<br>[12, 14)</td><td colspan="2">15:<br>[14, 16)</td></tr>
<tr><td>16:<br>0</td><td>17:<br>1</td><td>18:<br>2</td><td>19:<br>3</td><td>20:<br>4</td><td>21:<br>5</td><td>22:<br>6</td><td>23:<br>7</td><td>24:<br>8</td><td>25:<br>9</td><td>26:<br>10</td><td>27:<br>11</td><td>28:<br>12</td><td>29:<br>13</td><td>30:<br>14</td><td>31:<br>15</td></tr>
</table>

- $BLOCK_i$ is made from a combination of $BLOCK_{2i}$ and $BLOCK_{2i+1}$

Compute the sum between interval [3,11)

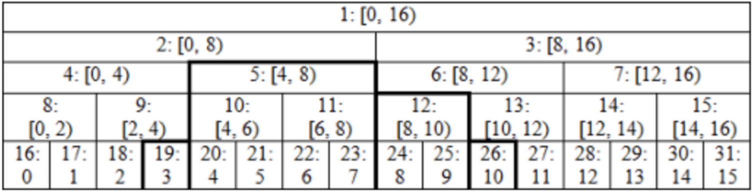
Start with x,y = [0,16)

- Does [3,11) cover the entire [x,y)
- No → drill down to the left side and the right side
- Yes → take the answer in the entire block

<table>
<tr><td colspan="16">1: [0, 16)</td></tr>
<tr><td colspan="8">2: [0, 8)</td><td colspan="8">3: [8, 16)</td></tr>
<tr><td colspan="4">4: [0, 4)</td><td colspan="4">5: [4, 8)</td><td colspan="4">6: [8, 12)</td><td colspan="4">7: [12, 16)</td></tr>
<tr><td colspan="2">8:<br>[0, 2)</td><td colspan="2">9:<br>[2, 4)</td><td colspan="2">10:<br>[4, 6)</td><td colspan="2">11:<br>[6, 8)</td><td colspan="2">12:<br>[8, 10)</td><td colspan="2">13:<br>[10, 12)</td><td colspan="2">14:<br>[12, 14)</td><td colspan="2">15:<br>[14, 16)</td></tr>
<tr><td>16:<br>0</td><td>17:<br>1</td><td>18:<br>2</td><td>19:<br>3</td><td>20:<br>4</td><td>21:<br>5</td><td>22:<br>6</td><td>23:<br>7</td><td>24:<br>8</td><td>25:<br>9</td><td>26:<br>10</td><td>27:<br>11</td><td>28:<br>12</td><td>29:<br>13</td><td>30:<br>14</td><td>31:<br>15</td></tr>
</table>

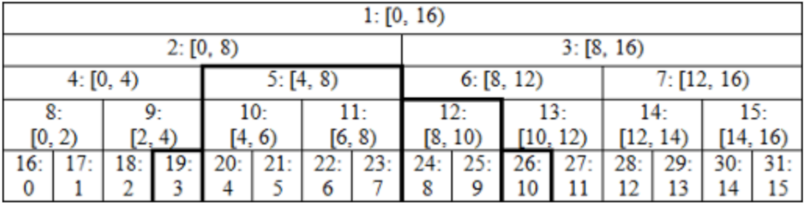
# Analysis

- Build - O(n)
  - Original array starts n+0
  - Parents are stored between [0,n)
- Modify - O(log(n))
  - We only need to modify the parents of the current node. Located at p/2, follow the ancestor to the top level.



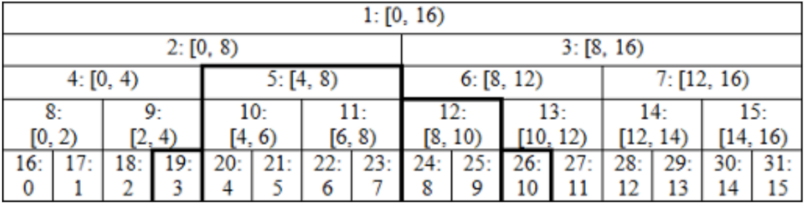
# Query

- query(l,r) - O(log(n))
  - Let l be the left boundary: If l is odd then it has to be the right child of its parent. (We include l but not its parent)
  - Move to the right of l's parent If l is even then it's left child
  - Move to parent



# Query

- The right boundary has similar properties
  - If r is odd then it's right child
  - If r is even then it's left child
- Terminate when $l \geq r$, the boundaries meet

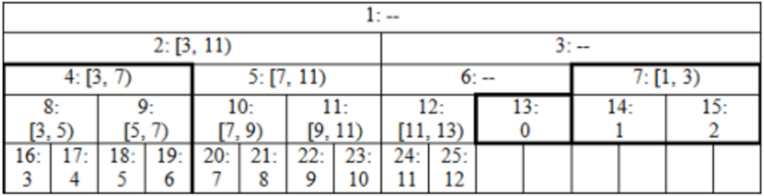


# Efficient Implementation

Now we will implement segment tree efficiently using binary logic and without recursion

# Arbitrary sized array

- What if the length of the array is not a power of 2?
- 0,1,2,3,4,5,6,7,8,0,10,11,12




- Reduction: Arbitrary sized tree → A set of multiple perfect binary tree
- Same implementation works

## Non-Commutative Function

- Addition / Min / Max are commutative
  - $a + b = b + a$
  - $\min(a,b) = \min(b,a)$
- What if our "function" is complex and non-commutative?

# Non-Commutative Example

- Problem Statement: Find the minimum and the number of elements equal to the minimum in a segment.
- Two types of operations:
  **1**: Update index i to value v
  **2**: Query (l,r) for minimum and the number of elements equal to the min.
- Example:
  5 5
  3 4 3 5 2
  2 0 3
  1 1 2
  2 0 3
  1 0 2
  2 0 5