**Calculation of size**

The segment tree is a **complete binary tree**. The number of leaves must be a power of 2, even if it means padding with zeros.

* Length of original data, n = 6.
* We need the smallest power of 2 that is **greater than or equal to n**.
* Let's list powers of 2: 1, 2, 4, 8...
* 4 < 6? Yes.
* 8 >= 6? Yes.
* Therefore, the correct size is **8**.  
  The total tree array size is 2 \* size = 16. The leaves will be at indices from 8 to 15 (inclusive). Since we only have 6 data points, leaves at indices 14 and 15 will be padding (value 0).  
  **So, your original statement was correct: size = 8.** My initial doubt was misplaced. Let's proceed with the correct build.

**1. Correctly Building the Tree (size = 8)**

**Original Array:** data = [1, 2, 3, 4, 5, 6] (indices 0 to 5)  
**Tree Array Indices (self.tree):**

* Internal Nodes: indices 1 to 7
* Leaves: indices 8 to 15  
  **Step 1: Populate the Leaves**  
  We copy the data into the leaf segment. The remaining leaves are padded with zero.

Index: 08, 09, 10, 11, 12, 13, 14, 15  
Value: 1, 2, 3, 4, 5, 6, 0, 0

**Step 2: Build Internal Nodes (Bottom-Up)**  
We calculate the value for each internal node (i from 7 down to 1) as the sum of its two children (2\*i and 2\*i + 1).

* i = 7: tree[7] = tree[14] + tree[15] = 0 + 0 = 0
* i = 6: tree[6] = tree[12] + tree[13] = 5 + 6 = 11
* i = 5: tree[5] = tree[10] + tree[11] = 3 + 4 = 7
* i = 4: tree[4] = tree[8] + tree[9] = 1 + 2 = 3
* i = 3: tree[3] = tree[6] + tree[7] = 11 + 0 = 11
* i = 2: tree[2] = tree[4] + tree[5] = 3 + 7 = 10
* i = 1: tree[1] = tree[2] + tree[3] = 10 + 11 = 21 **(Root)**  
  **Final Tree Array:**

Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  
Value: [0], 21, 10, 11, 3, 7, 11, 0, 1, 2, 3, 4, 5, 6, 0, 0

*(Index 0 is unused)*  
This tree is now built correctly. The visual representation of the tree structure is as follows:

[1:21]  
 / \  
 [2:10] [3:11]  
 / \ / \  
 [4:3] [5:7] [6:11] [7:0]  
 / \ / \ / \ / \  
[8:1][9:2]... ... ... ... ...

**2. Querying Range [2, 5) (Get sum of elements at positions 2, 3, 4)**

We want the sum of data[2] + data[3] + data[4] = 3 + 4 + 5 = 12.  
Let's execute the query function step by step with the **correctly built tree**:  
**Initialization:**

l = 2 # start index (inclusive)  
r = 5 # end index (exclusive)  
l += self.size # l = 2 + 8 = 10  
r += self.size # r = 5 + 8 = 13  
res = 0

**Now we enter the while l < r loop (10 < 13 is true):**

**Iteration 1: l = 10, r = 13**

* if l & 1: → 10 & 1 → 0 (False). l is even. **Do nothing.**
* if r & 1: → 13 & 1 → 1 (True). r is odd.
  + r -= 1 → r = 12
  + res += self.tree[12] → res = 0 + 5 = 5 // Added leaf node 12 (value 5)
* l //= 2 → l = 10 // 2 = 5
* r //= 2 → r = 12 // 2 = 6
* **State:** l=5, r=6, res=5

**Iteration 2: l = 5, r = 6 (5 < 6 is true)**

* if l & 1: → 5 & 1 → 1 (True). l is odd.
  + res += self.tree[5] → res = 5 + 7 = 12 // Added internal node 5 (value 7, which is 3+4)
  + l += 1 → l = 6
* if r & 1: → 6 & 1 → 0 (False). r is even. **Do nothing.**
* l //= 2 → l = 6 // 2 = 3
* r //= 2 → r = 6 // 2 = 3
* **State:** l=3, r=3, res=12

**Iteration 3: l = 3, r = 3 (3 < 3 is false)**

* Exit the loop.  
  **Return res = 12** ✓ (Which is correct: 3 + 4 + 5 = 12)

**Summary of Nodes Used:**

The query for range [2, 5) was answered by combining these two nodes from the tree:

1. **tree[5] (Value = 7)**: This internal node represents the range [2, 3] (elements 3, 4).
2. **tree[12] (Value = 5)**: This leaf node represents the single element at index 4 (element 5).  
   This is the most efficient way to cover the range [2, 5), using only 2 segments instead of 3 individual elements. The algorithm successfully found these segments by analyzing the odd/even nature of the indices as it traversed from the leaves toward the root.

The correct construction and querying of the segment tree can be presented in an organized, step-by-step format for clarity and quick reference.

**Segment Tree: Correct Size Calculation**

When building a segment tree, the number of leaves is always the next power of 2 greater than or equal to the data length.

* **Original Data Length**: $ n = 6 $
* **Smallest Power of 2 ≥ n**: List is 1, 2, 4, 8...
  + $4 < 6 $ — not enough
  + — correct
* **Correct Size** (**size**): **8**
* **Total Array Size**: (indices 0–15)

**Tree Array Construction**

**Leaf Assignment**

Indices 8–15 are allocated for leaves. The data is placed at the beginning, and remaining leaves are padded with zero:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Value | 1 | 2 | 3 | 4 | 5 | 6 | 0 | 0 |

**Internal Nodes**

Work bottom-up (indices 7 down to 1):

* 7: $ tree = tree + tree = 0 + 0 = 0 $
* 6: $ tree = tree + tree = 5 + 6 = 11 $
* 5: $ tree = tree + tree = 3 + 4 = 7 $
* 4: $ tree = tree + tree = 1 + 2 = 3 $
* 3: $ tree = tree + tree = 11 + 0 = 11 $
* 2: $ tree = tree + tree = 3 + 7 = 10 $
* 1: $ tree = tree + tree = 10 + 11 = 21 $ (**Root**)

**Full Array Layout**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Value | 0 | 21 | 10 | 11 | 3 | 7 | 11 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 0 | 0 |

(The root is at index 1; index 0 is not used.)

**Visual Representation**

[1:21]  
 / \  
 [2:10] [3:11]  
 / \ / \  
 [4:3] [5:7] [6:11] [7:0]  
 / \ / \ / \ / \  
[8:1][9:2][10:3][11:4][12:5][13:6][14:0][15:0]

**Range Query Example: [2, 5)**

To find the sum for indices 2 (inclusive) to 5 (exclusive):

* **Initialization**:
  + $ l = 2, r = 5 $
  + $ l = l + size = 10 $
  + $ r = r + size = 13 $
  + $ res = 0 $

**Step-by-Step Execution**

1. **Iteration 1** (l=10, r=13):
   * l even → do nothing.
   * r odd → $ r -= 1; r = 12 $; $ res += tree; res = 5 $
   * l=5, r=6, res=5
2. **Iteration 2** (l=5, r=6):
   * l odd → $ res += tree; res = 12 $; $ l += 1, l = 6 $
   * r even → do nothing.
   * l=3, r=3, res=12
3. **Exit** (l=3, r=3 — not less than)
   * **Final Result**: $ res = 12 $ (matches $ 3 + 4 + 5 $)

**Efficient Coverage**

The query smartly combines the following tree nodes:

* **tree** (value 7): covers (values 3,4)
* **tree** (value 5): covers (value 5)

**Conclusion**

* **Correct size**: 8.
* **Correct tree structure**: padded with zeros for unused leaves.
* **Efficient query logic**: combines minimal segments, efficiently calculates ranges.