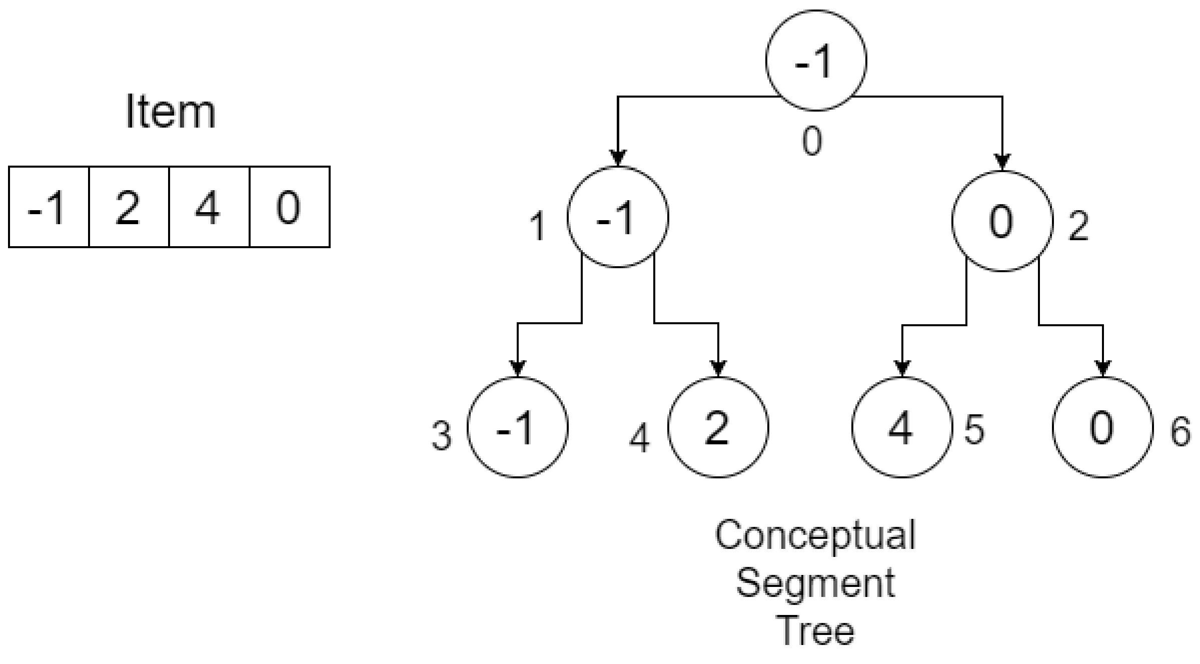
Performing a Range Minimum Query

Example

The procedure to perform a RMQ is already shown in introduction. The pseudo-code for checking Range Minimum Query will be:

```
Procedure RangeMinimumQuery(SegmentTree, qLow, qHigh, low, high, position):
if qLow <= low and qHigh >= high        //Total Overlap
    Return SegmentTree[position]
else if qLow > high || qHigh < low      //No Overlap
    Return infinity
else                                    //Partial Overlap
    mid := (low+high)/2
    Return min(RangeMinimumQuery(SegmentTree, qLow, qHigh, low, mid, 2*position+1),
               RangeMinimumQuery(SegmentTree, qLow, qHigh, mid+1, high, 2*position+2))
end if
```

Here, `qLow = The lower range of our query`, `qHigh = The upper range of our query`. `low = starting index of Item array`, `high = Finishing index of Item array`, `position = root = 0`. Now let's try to understand the procedure using the example we created before:



Our **SegmentTree** array:

```
   0     1     2     3     4     5     6
+-----+-----+-----+-----+-----+-----+-----+
|  -1 |  -1 |  0  |  -1 |  2  |  4  |  0  |
+-----+-----+-----+-----+-----+-----+-----+
```

We want to find the minimum in range **[1,3]**.

Since this is a recursive procedure, we'll see the operation of the `RangeMinimumQuery` using a recursion table that keeps track of `qLow`, `qHigh`, `low`, `high`, `position`, `mid` and `calling line`. At first, we call **RangeMinimumQuery(SegmentTree, 1, 3, 0, 3, 0**. Here, the first two conditions are not met(partial overlap). We'll get a `mid`. The `calling line` indicates which `RangeMinimumQuery` is called after this statement. We denote the `RangeMinimumQuery` calls inside the procedure as **1** and **2** respectively. Our table will look like:

```
+------+-------+-----+------+----------+-----+--------------+
| qLow | qHigh | low | high | position | mid | calling line |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  0  |   3  |     0    |  1  |       1      |
+------+-------+-----+------+----------+-----+--------------+
```

So when we call `RangeMinimumQuery-1`, we pass: `low = 0`, `high = mid = 1`, `position = 2*position+1 = 1`. One thing you can notice, that is `2*position+1` is the left child of a **node**. That means we're checking the left child of **root**. Since **[1,3]** partially overlaps **[0,1]**, the first two conditions are not met, we get a `mid`. Our table:

```
+------+-------+-----+------+----------+-----+--------------+
| qLow | qHigh | low | high | position | mid | calling line |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  0  |   3  |     0    |  1  |       1      |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  0  |   1  |     1    |  0  |       1      |
+------+-------+-----+------+----------+-----+--------------+
```

In the next recursive call, we pass `low = 0`, `high = mid = 0`, `position = 2*position+1 = 3`. We reach the leftmost leaf of our tree. Since **[1,3]** doesn't overlap with **[0,0]**, we return `infinity` to our calling function. Recursion table:

```
+------+-------+-----+------+----------+-----+--------------+
| qLow | qHigh | low | high | position | mid | calling line |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  0  |   3  |     0    |  1  |       1      |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  0  |   1  |     1    |  0  |       1      |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  0  |   0  |     3    |     |              |
+------+-------+-----+------+----------+-----+--------------+
```

Since this recursive call is complete, we go back to the previous row of our recursion table. We get:

```
+------+-------+-----+------+----------+-----+--------------+
| qLow | qHigh | low | high | position | mid | calling line |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  0  |   3  |     0    |  1  |       1      |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  0  |   1  |     1    |  0  |       1      |
+------+-------+-----+------+----------+-----+--------------+
```

In our procedure, we execute `RangeMinimumQuery-2` call. This time, we pass `low = mid+1 = 1`, `high = 1` and `position = 2*position+2 = 4`. Our `calling line changes to **2**`. We get:

```
+------+-------+-----+------+----------+-----+--------------+
| qLow | qHigh | low | high | position | mid | calling line |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  0  |   3  |     0    |  1  |       1      |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  0  |   1  |     1    |  0  |       2      |
+------+-------+-----+------+----------+-----+--------------+
|   1  |   3   |  1  |   1  |     4    |     |              |
+------+-------+-----+------+----------+-----+--------------+
```

So we are going to the right child of previous node. This time there is a total overlap. We return the value `SegmentTree[position] = SegmentTree[4] = 2`.

```
+------+-------+-----+------+----------+-----+-------------+
| qLow | qHigh | low | high | position | mid | calling line |
+------+-------+-----+------+----------+-----+-------------+
|   1  |   3   |  0  |   3  |     0    |  1  |       1      |
+------+-------+-----+------+----------+-----+-------------+
```

Back at the calling function, we are checking what is the minimum of the two returned values of two calling functions. Obviously the minimum is **2**, we return **2** to the calling function. Our recursion table looks like:

```
+------+-------+-----+------+----------+-----+-------------+
| qLow | qHigh | low | high | position | mid | calling line |
+------+-------+-----+------+----------+-----+-------------+
|   1  |   3   |  0  |   3  |     0    |  1  |       1      |
+------+-------+-----+------+----------+-----+-------------+
```

We're done looking at the left portion of our segment tree. Now we'll call `RangeMinimumQuery-2` from here. We'll pass `low = mid+1 = 1+1 =2`, `high = 3` and `position = 2*position+2 = 2`. Our table:

```
+------+-------+-----+------+----------+-----+-------------+
| qLow | qHigh | low | high | position | mid | calling line |
+------+-------+-----+------+----------+-----+-------------+
|   1  |   3   |  0  |   3  |     0    |  1  |       1      |
+------+-------+-----+------+----------+-----+-------------+
|   1  |   3   |  2  |   3  |     2    |     |              |
+------+-------+-----+------+----------+-----+-------------+
```

There is a total overlap. So we return the value: `SegmentTree[position] = SegmentTree[2] = 0`. We come back to the recursion from where these two children were called and get the minimum of **(4,0)**, that is **0** and return the value.

After executing the procedure, we get **0**, which is the minimum from index-**1** to index-**3**.

The runtime complexity for this procedure is `O(logn)` where **n** is the number of elements in the **Items** array. To perform a Range Maximum Query, we need to replace the line:

```
Return min(RangeMinimumQuery(SegmentTree, qLow, qHigh, low, mid, 2*position+1),
               RangeMinimumQuery(SegmentTree, qLow, qHigh, mid+1, high, 2*position+2))
```

with:

```
Return max(RangeMinimumQuery(SegmentTree, qLow, qHigh, low, mid, 2*position+1),
               RangeMinimumQuery(SegmentTree, qLow, qHigh, mid+1, high, 2*position+2))
```

‹ Previous Next ›

PDF - Download **data-structures** for free