Bubble Up Part II

13 min

Now that we understand how to determine the relationship of elements with the internal heap, we're ready to finish .bubbleUp().

In a min-heap, we need to ensure that every child is greater in value than its parent. Let's add an element to the following heap.

```
console.log(minHeap.heap)
// returns [null, 10, 13, 21, 61, 22, 23, 99]

heap.add(12)
```

(new_element)

{ parent_element }

[null, 10, 13, 21, {61}, 22, 23, 99, (12)]

Oh no! We've violated the min-heap condition because 12's parent, 61, is greater than its child, 12.

To fix this, we will exchange the parent and the child elements.

before

[null, 10, 13, 21, {61}, 22, 23, 99, (12)]

SWAP 12 and 61

after

[null, 10, 13, 21, (12), 22, 23, 99, {61}]

12's parent is now 13 and it violates the min-heap condition. To fix this, we continue moving upwards swapping parent-child values.

before

[null, 10, {13}, 21, (12), 22, 23, 99, 61]

SWAP 12 and 13

after

[null, 10, (12), 21, {13}, 22, 23, 99, 61]

12's parent is now 10 and no longer violates the min-heap condition. We've restored the heap!

[null, {10}, (12), 21, 13, 22, 23, 99, 61]

The child, 12, is greater than the parent, 10!

Let’s recap our strategy for .bubbleUp():

Set the current element index to be the last index of heap

While current element is valid and its value is less than its parent's value

Swap the indexes

Update the current element index to be its parent index

Instructions

1. Checkpoint 1 Passed

1.

.bubbleUp() is called by .add() which appends an element to the internal heap property. Hence, we need to keep track of the added element.

Inside .bubbleUp(), declare a let current variable that will point to the added element’s index. Initialize current to the added element’s current location, which is the end of heap.

Hint

The size property tells us how many elements have been added to the heap array. It can also be used as an index to the last element of the array. For example:

let lastIndexOfArray = this.size;

Similarly, you can set current to this.size as well.

2. Checkpoint 2 Passed

2.

In .bubbleUp(), set up a while loop that will run as long as it meets these 2 conditions:

- There is a valid current index. A valid current index is anything greater than 1.
- The value at the current index is less than its parent’s value. This will violate the min-heap condition and will trigger swapping values. Use a helper method to derive the parent index.

Hint

Implement a while loop that meets this criteria:

while condition1 and condition2 are true

where

condition1 is current index greater than 1, and

condition2 is current value is less than its parent's value

Remember that current value can be derived from this.heap[current] and parent's value is determined by this.heap[getParent(current)].

3. Checkpoint 3 Passed

3.

Inside the while loop, swap the parent index and the current index using the helper method, .swap() that has been provided for you. Pass both current and parent indices to .swap().

Optionally, display:

- the content of the current heap and
- a message that shows swapping will occur between the current index and its parent before the actual swap.

Hint

To swap two elements with .swap() do the following:

```
this.swap(indexA, indexB);
console.log(indexA, indexB); // should return 7 5
```

To log meaningful messages before the swap, you can do something like this:

```
console.log('..', this.heap);
console.log(`.. swap index ${indexA} with ${indexB}`);
this.swap(indexA, indexB);
```

Fill in ${indexA} and ${indexB} with the current index and its parent index.

4. Checkpoint 4 Passed

4.

The last thing to do inside the while loop is to update the current index to be its parent's index, since we are progressing upwards, or *bubbling up*, the binary tree model of the min-heap.

Hint

To set an element's index to point to its parent, use one of the helper functions available. For example:

```
someIndex = getParent(someIndex);
```

5. Checkpoint 5 Passed

5.

Open **script.js** and run the code to enjoy the fruits of your labor! You should find the smallest value at the beginning of the heap at index 1.

MinHeap.js

```
class MinHeap {
  constructor() {
    this.heap = [ null ];
    this.size = 0;
  }

  add(value) {
    console.log(`.. adding ${value}`);
    this.heap.push(value);
    this.size++;
    this.bubbleUp();
    console.log(`added ${value} to heap`, this.heap);
  }

  bubbleUp() {
    let current = this.size;
    while (current > 1 && this.heap[current] < this.heap[getParent(current)]) {
      console.log('..', this.heap);
      console.log(`.. swap index ${current} with ${getParent(current)}`);
      this.swap(current, getParent(current));
      current = getParent(current);
    }
  }
```

```
  swap(a, b) {
    [this.heap[a], this.heap[b]] = [this.heap[b], this.heap[a]];
  }

}

const getParent = current => Math.floor((current / 2));
const getLeft = current => current * 2;
const getRight = current => current * 2 + 1;

module.exports = MinHeap;
```

script.js

```
// import MinHeap class
const MinHeap = require('./MinHeap');

// instantiate a MinHeap class
const minHeap = new MinHeap();

// helper function to return a random integer
function randomize() { return Math.floor(Math.random() * 40); }

// populate minHeap with random numbers
for (let i=0; i < 6; i++) {
  minHeap.add(randomize());
}

// display the bubbled up numbers in the heap
```

```
console.log('Bubbled Up', minHeap.heap);
```

>> Out

```
.. adding 10
added 10 to heap [ null, 10 ]
.. adding 15
added 15 to heap [ null, 10, 15 ]
.. adding 23
added 23 to heap [ null, 10, 15, 23 ]
.. adding 13
.. [ null, 10, 15, 23, 13 ]
.. swap index 4 with 2
added 13 to heap [ null, 10, 13, 23, 15 ]
.. adding 33
added 33 to heap [ null, 10, 13, 23, 15, 33 ]
.. adding 3
.. [ null, 10, 13, 23, 15, 33, 3 ]
.. swap index 6 with 3
.. [ null, 10, 13, 3, 15, 33, 23 ]
.. swap index 3 with 1
added 3 to heap [ null, 3, 13, 10, 15, 33, 23 ]
Bubbled Up [ null, 3, 13, 10, 15, 33, 23 ]
```