QUIZ

The helper method `.canSwap()` determines if the current element breaks the heap condition of:

```
The current element has to be less than both its children
```

by comparing it with either of its children. Fill in the code with the correct conditional.

```
canSwap(current, leftChild, rightChild) {
    // Check that one of the possible swap conditions exists
    return (
      this.exists(leftChild) && this.heap[current] > this.heap[leftChild]
      || this.exists(rightChild) && this.heap[current] > this.heap[rightChild]
    );
  }
```

You got it!

Finish the `MinHeap .heapify()` method in order to prevent it from running infinitely.

```
heapify() {
    let current = 1;
    let leftChild = getLeft(current);
    let rightChild = getRight(current);

    while (this.canSwap(current, leftChild, rightChild)) {
      if (this.exists(leftChild) && this.exists(rightChild)) {
        if (this.heap[leftChild] < this.heap[rightChild]) {
          this.swap(current, leftChild);
          current = leftChild;
        } else {
          this.swap(current, rightChild);
          current = rightChild;
        }
      } else {
        this.swap(current, leftChild);
        current = leftChild;
      }
      leftChild = getLeft( current );
      rightChild = getRight( current );

    }
  }
```

You got it!

The Javascript `MinHeap` class method, `.push(value)`, works in conjunction with `.bubbleUp()`. The purpose of `.bubbleUp()` is to restore the min-heap condition when an element is pushed to the heap:

```
the parent element has to be less in value than the child element
```

Fill in the code with the correct swap condition in `.bubbleUp()`.

```
class MinHeap {
  // ...
  add(value) {
    this.heap.push(value);
    this.size++;
    this.bubbleUp();
  }

  bubbleUp() {
    let current = this.size;

    while (current > 1 && this.heap[ getParent(current) ] > this.heap[ current ]) {

      this.swap(current, getParent(current));
      current = getParent(current);
    }
  }
}
```

You got it!

The `MinHeap` class method `.heapify()` restores the minimum element in the heap after it has been removed in `.popMin()`. Fill in the correct condition in the code that enables the minimum element at index `1` to swap with the appropriate child element.

```
class MinHeap {
  // ...
  heapify() {
    let current = 1;
    let leftChild = getLeft(current);
    let rightChild = getRight(current);

    while (this.canSwap(current, leftChild, rightChild)) {
      if (this.exists(leftChild) && this.exists(rightChild)) {
        // Make sure to swap with the smaller of the two children
        if ( this.heap[leftChild] < this.heap[rightChild] ) {
          this.swap(current, leftChild);
          current = leftChild;
        } else {
          this.swap(current, rightChild);
          current = rightChild;
        }
      } else {
        this.swap(current, leftChild);
        current = leftChild;
      }
      leftChild = getLeft(current);
      rightChild = getRight(current);
    }
  }
}
```

You got it!

This `.popMin()` `MinHeap` method is missing a few steps. Complete it so that it removes the minimum value and returns it, or returns `null` if the heap is empty.

```
popMin() {
  if (this.size === 0) {
    return null ;
  }
  this.swap(1, this.size) ;
  const min = this.heap.pop();
  this.size-- ;
  this.heapify();
  return min ;
}
```

You got it!

What would be the correct restored state of this heap:

```
[ null, 14, 15, 36, 28, 38 ]
```

after adding `30` to it? Hint: Locate the parent index of the appended element and swap if necessary.

```
[ null, 14, 15, 36, 30, 38, 28 ]
```

```
[ null, 14, 15, 30, 28, 38, 36 ]
```

👏 Excellent!

What are the parent index, left child index and right child index of an element at index 10?

Parent: 5, Left Child: 21, Right Child: 20

Parent: 5, Left Child: 20, Right Child: 21

👏 Excellent!

Parent: 21, Left Child: 20, Right Child: 5

Parent: 20, Left Child: 5, Right Child: 21

Which of these is not an operation of a Min-Heap data structure?

Retrieve the minimum element from the heap.

Restore the heap after a removal of the minimum element.

Add an element to the heap.

Insert an element at the root of the heap.

👏 This is not a valid operation, as an element is always appended at the end, and not at the root.