```
class MinHeap {

    constructor () {
        /* Initialing the array heap and adding a dummy element at index 0 */
        this.heap = [null]
    }

    getMin () {
        /* Accessing the min element at index 1 in the heap array */
        return this.heap[1]
    }

    insert (node) {

        /* Inserting the new node at the end of the heap array */
        this.heap.push(node)

        /* Finding the correct position for the new node */

        if (this.heap.length > 1) {
            let current = this.heap.length - 1

            /* Traversing up the parent node until the current node (current) is greater than the parent (current/2)*/
            while (current > 1 && this.heap[Math.floor(current/2)] > this.heap[current]) {

                /* Swapping the two nodes by using the ES6 destructuring syntax*/
                [this.heap[Math.floor(current/2)], this.heap[current]] = [this.heap[current], this.heap[Math.floor(current/2)]]
                current = Math.floor(current/2)
            }
        }
    }

    remove() {
        /* Smallest element is at the index 1 in the heap array */
        let smallest = this.heap[1]

        /* When there are more than two elements in the array, we put the right most element at the first position
            and start comparing nodes with the child nodes
        */
        if (this.heap.length > 2) {
            this.heap[1] = this.heap[this.heap.length-1]
            this.heap.splice(this.heap.length - 1)

            if (this.heap.length === 3) {
                if (this.heap[1] > this.heap[2]) {
                    [this.heap[1], this.heap[2]] = [this.heap[2], this.heap[1]]
                }
                return smallest
            }

            let current = 1
            let leftChildIndex = current * 2
            let rightChildIndex = current * 2 + 1

            while (this.heap[leftChildIndex] &&
                    this.heap[rightChildIndex] &&
                    (this.heap[current] > this.heap[leftChildIndex] ||
                        this.heap[current] > this.heap[rightChildIndex])) {
                if (this.heap[leftChildIndex] < this.heap[rightChildIndex]) {
                    [this.heap[current], this.heap[leftChildIndex]] = [this.heap[leftChildIndex], this.heap[current]]
                    current = leftChildIndex
                } else {
                    [this.heap[current], this.heap[rightChildIndex]] = [this.heap[rightChildIndex], this.heap[current]]
                    current = rightChildIndex
                }

                leftChildIndex = current * 2
                rightChildIndex = current * 2 + 1
            }
        }

        /* If there are only two elements in the array, we directly splice out the first element */

        else if (this.heap.length === 2) {
            this.heap.splice(1, 1)
        } else {
            return null
        }

        return smallest
    }
}

let heapOne = new MinHeap();

heapOne.insert(5);
heapOne.insert(7);
heapOne.insert(2);

console.log(heapOne.getMin());
console.log(heapOne.heap.indexOf(7));
console.log(heapOne.heap.indexOf(5));
console.log(heapOne.heap);

console.log(heapOne.heap[0]);
console.log(heapOne.heap[1]);
console.log(heapOne.heap[2]);
console.log(heapOne.heap[3]);

console.log("length before: "+heapOne.heap.length);

heapOne.remove(2);

console.log("length after: "+ heapOne.heap.length);
```