Queues - Practice Problems

Do the given problems using JavaScript.

1. Write a Program to find the first non-repeating character using a queue.
2. Write a Program to sort the queue.
3. Write a Program to design a queue using only your knowledge of stacks.
4. Write a Program to connect the n ropes in minimum cost such that the cost to connect two ropes is equal to the sum of their lengths.

   Input: arr[] = {1, 2, 3} , N = 3

   Output: 9

   Explanation: First, connect ropes of lengths 1 and 2. Now we have two ropes of lengths 3 and 3. Finally connect the two ropes and all ropes are connected.

Solutions

1.

```
const MAX_CHAR = 26;

    function firstNonRepeating(str) {
      // count array of size 26(assuming
      // only lower case characters are present)
      var charCount = new Array(MAX_CHAR).fill(0);

      // Queue to store Characters
      var q = [];

      // traverse whole stream
      for (var i = 0; i < str.length; i++) {
        var ch = str[i];

        // push each character in queue
        q.push(ch);

        // increment the frequency count
        charCount[ch.charCodeAt(0) - "a".charCodeAt(0)]++;

        // check for the non repeating character
        while (q.length > 0) {
          if (charCount[q[0].charCodeAt(0) -
          "a".charCodeAt(0)] > 1) {
            q.shift();
          } else {
            document.write(q[0] + " ");
            break;
          }
        }
        if (q.length == 0) {
          document.write(-1 + " ");
        }
      }
      document.write("<br>");
    }
    var str = "aabc";
    firstNonRepeating(str);
```

2.

```
function minIndex(q, sortedIndex)
{
    let min_index = -1
    let min_val = 999999999999
    let n = q.length
    for(let i = 0; i < n; i++)
    {
        let curr = q.shift()
```

```
        // q.get() / This is dequeue() in C++ STL

        // / we add the condition i <= sortedIndex
        // / because we don't want to traverse
        // / on the sorted part of the queue,
        // / which is the right part.
        if (curr <= min_val && i <= sortedIndex){
            min_index = i
            min_val = curr
        }
        q.push(curr) // This is enqueue() in
                    // / C++ STL
    }
    return min_index
}

// / Moves given minimum element to
// / rear of queue
function insertMinToRear(q, min_index){
    let min_val = 0
    let n = q.length
    for(let i=0;i<n;i++){
        let curr = q.shift()
        if (i != min_index)
            q.push(curr)
        else
            min_val = curr
    }
    q.push(min_val)
}

function sortQueue(q){
    for(let i=1;i<q.length+1;i++){
        let min_index = minIndex(q, q.length - i)
        insertMinToRear(q, min_index)
    }
}
let q = []
q.push(30)
q.push(11)
q.push(15)
q.push(4)

// / Sort queue
sortQueue(q)

// / Print sorted queue
while (q.length > 0){
    document.write(q.shift()," ")
}
```

3.

```
function Stack(array) {
    this.array = [];
    if (array) this.array = array;
}

Stack.prototype.getBuffer = function() {
    return this.array.slice();
}

Stack.prototype.isEmpty = function() {
    return this.array.length == 0;
}

Stack.prototype.peek = function() {
    return this.array[this.array.length - 1];
}

// Insertion:
// Time Complexity:     O(1)
// Space Complexity:    O(1)
Stack.prototype.push = function(value) {
    this.array.push(value);
}

// Deletion:
// Time Complexity:     O(1)
// Space Complexity:    O(1)
Stack.prototype.pop = function() {
    return this.array.pop();
};

// Access:
// Time Complexity:     O(n)
// Space Complexity:    O(n)
Stack.prototype.access = function(n) {
    var bufferArray = this.getBuffer();
    if (n <= 0) throw 'error'

    var bufferStack = new Stack(bufferArray);

    while (--n !== 0) {
        bufferStack.pop();
    }
    return bufferStack.pop();
};

// Search:
// Time Complexity:     O(n)
// Space Complexity:    O(n)
Stack.prototype.search = function(element) {
    var bufferArray = this.getBuffer();
    var bufferStack = new Stack(bufferArray);

    while (!bufferStack.isEmpty()) {
```

```
        if (bufferStack.pop() == element) {
            return true;
        }
    }

    return false;
};

// Main Function
function StackQueue() {
    this.inbox = new Stack(); // first stack
    this.outbox = new Stack();
}

StackQueue.prototype.enqueue = function(val) {
    this.inbox.push(val);
};

StackQueue.prototype.dequeue = function() {
    if (this.outbox.isEmpty()) {
        while (!this.inbox.isEmpty()) {
            this.outbox.push(this.inbox.pop());
        }
    }
    return this.outbox.pop();

};

var queue = new StackQueue();

queue.enqueue(7);
queue.enqueue(8);
queue.enqueue(9);
queue.enqueue(2);
queue.enqueue(1);

console.log(queue.dequeue()); // 7
console.log(queue.dequeue()); // 8
console.log(queue.dequeue()); // 9
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 1
```

4.

```
function minCost(arr,n)
{
        let pq = [];

        // Adding items to the pQueue
        for (let i = 0; i < n; i++) {
            pq.push(arr[i]);
        }

        pq.sort(function(a,b){return a-b;});
```

```
        let res = 0;

        while (pq.length > 1) {
            // Extract shortest two ropes from pq
            let first = pq.shift();
            let second = pq.shift();

            // Connect the ropes: update result
            // and insert the new rope to pq
            res += first + second;
            pq.push(first + second);
            pq.sort(function(a,b){return a-b;});
        }
        return res;
}
let len = [4, 3, 2, 6];
let size = len.length;
document.write("Total cost for connecting" + " ropes is " + minCost(len,
size));
```