Queues - Assignment Solutions

1.

```
function printQueue(queue)
    {
        while (queue.length != 0)
        {
            document.write(queue[0] + " ");
            queue.shift();
        }
    }

    // Recursive function
    // to reverse the queue
    function reverseQueue(q)
    {
        // Base case
        if (q.length == 0)
            return;

        // Dequeue current
        // item (from front)
        let data = q[0];
        q.shift();

        // Reverse remaining queue
        reverseQueue(q);

        // Enqueue current
        // item (to rear)
        q.push(data);
    }

    let queue = [];
    queue.push(56);
    queue.push(27);
    queue.push(30);
    queue.push(45);
    queue.push(85);
    queue.push(92);
    queue.push(58);
    queue.push(80);
    queue.push(90);
    queue.push(100);
    reverseQueue(queue);
    printQueue(queue);
```

2.

```
function Queue(array) {
    this.array = [];
    if (array) this.array = array;
}
Queue.prototype.getBuffer = function() {
    return this.array.slice();
}

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

Queue.prototype.peek = function() {
    return this.array[0];
}

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

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

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

    var bufferStack = new Queue(bufferArray);

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

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

    while (!bufferQueue.isEmpty()) {
        if (bufferQueue.dequeue() == element) {
```

```
            return true;
        }
    }
    return false;
}

// Main Function
function QueueStack() {
    this.inbox = new Queue(); // first stack
}

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

QueueStack.prototype.pop = function() {
    var size = this.inbox.array.length - 1;
    var counter = 0;
    var bufferQueue = new Queue();

    while (++counter <= size) {
        bufferQueue.enqueue(this.inbox.dequeue());
    }
    var popped = this.inbox.dequeue();
    this.inbox = bufferQueue;
    return popped
};

var stack = new QueueStack();

stack.push(7);
stack.push(8);
stack.push(9);
stack.push(1);
stack.push(3);

console.log(stack.pop()); // 3
console.log(stack.pop()); // 1
console.log(stack.pop()); // 9
console.log(stack.pop()); // 8
console.log(stack.pop()); // 7
```

3.

```
function reverseQueueFirstKElements(k,Queue)
{
    if (Queue.length == 0 || k > Queue.length)
        return;
    if (k <= 0)
        return;

    let Stack = [];

    /* Push the first K elements
    into a Stack*/
```

```
    for (let i = 0; i < k; i++) {
        Stack.push(Queue.shift());
    }

    /* Enqueue the contents of stack
    at the back of the queue*/
    while (Stack.length > 0) {
        Queue.push(Stack.pop());
    }

    /* Remove the remaining elements and
    enqueue them at the end of the Queue*/
    for (let i = 0; i < Queue.length - k; i++) {
        Queue.push(Queue.shift());
    }
}

function Print(Queue)
{
    while (Queue.length > 0) {
        document.write(Queue.shift()," ");
    }
}

let Queue = [];
Queue.push(10);
Queue.push(20);
Queue.push(30);
Queue.push(40);
Queue.push(50);
Queue.push(60);
Queue.push(70);
Queue.push(80);
Queue.push(90);
Queue.push(100);

let k = 5;
reverseQueueFirstKElements(k, Queue);
Print(Queue);
```

4.

```
// A petrol pump has petrol and distance to next petrol pump
    class petrolPump {
        constructor(petrol, distance) {
            this.petrol = petrol;
            this.distance = distance;
        }
    };
    // The function returns starting point if there is a possible solution,
    // otherwise returns -1
    const printTour = (arr, n) => {
        // Consider first petrol pump as a starting point
        let start = 0;
        let end = 1;
```

```
        let curr_petrol = arr[start].petrol - arr[start].distance;

        /* Run a loop while all petrol pumps are not visited.
        And we have reached first petrol pump again with 0 or more petrol */
        while (end != start || curr_petrol < 0) {
            // If current amount of petrol in truck becomes less than 0,
then
            // remove the starting petrol pump from tour
            while (curr_petrol < 0 && start != end) {
                // Remove starting petrol pump. Change start
                curr_petrol -= arr[start].petrol - arr[start].distance;
                start = (start + 1) % n;

                // If 0 is being considered as start again, then there is no
                // possible solution
                if (start == 0)
                    return -1;
            }

            // Add a petrol pump to current tour
            curr_petrol += arr[end].petrol - arr[end].distance;

            end = (end + 1) % n;
        }

        // Return starting point
        return start;
    }
    let arr = [new petrolPump(6, 4), new petrolPump(3, 6), new petrolPump(7,
3)];
    let n = arr.length;
    let start = printTour(arr, n);

    (start == -1) ? document.write("No solution") : document.write(`Start =
${start}`);
```