**Assignment17**

**1.**

function firstUniqChar(s) {

const charMap = {};

for (let i = 0; i < s.length; i++) {

const char = s[i];

charMap[char] = charMap[char] ? charMap[char] + 1 : 1;

}

for (let i = 0; i < s.length; i++) {

const char = s[i];

if (charMap[char] === 1) {

return i;

}

}

return -1;

}

Example:

console.log(firstUniqChar("leetcode"));

Output: 0

console.log(firstUniqChar("loveleetcode"));

Output: 2

console.log(firstUniqChar("aabb"));

Output: -1

**2.**

function maxSubarraySumCircular(nums) {

let maxSum = nums[0];

let minSum = nums[0];

let totalSum = nums[0];

let currentMax = nums[0];

let currentMin = nums[0];

for (let i = 1; i < nums.length; i++) {

currentMax = Math.max(nums[i], currentMax + nums[i]);

maxSum = Math.max(maxSum, currentMax);

currentMin = Math.min(nums[i], currentMin + nums[i]);

minSum = Math.min(minSum, currentMin);

totalSum += nums[i];

}

if (maxSum < 0) {

return maxSum;

}

let circularMaxSum = totalSum - minSum;

return Math.max(maxSum, circularMaxSum);

}

Example:

console.log(maxSubarraySumCircular([1, -2, 3, -2]));

Output: 3

console.log(maxSubarraySumCircular([5, -3, 5]));

Output: 10

console.log(maxSubarraySumCircular([-3, -2, -3]));

Output: -2

**3.**

function countStudents(students, sandwiches) {

const n = students.length;

let count = 0;

while (count < n) {

if (students[0] === sandwiches[0]) {

students.shift();

sandwiches.shift();

count = 0;

} else {

students.push(students.shift());

count++;

}

}

return students.length;

}

Example:

console.log(countStudents([1, 1, 0, 0], [0, 1, 0, 1]));

Output: 0

console.log(countStudents([1, 1, 1, 0, 0, 1], [1, 0, 0, 0, 1, 1]));

Output: 3

**4.**

class RecentCounter {

constructor() {

this.requests = [];

}

ping(t) {

this.requests.push(t);

while (this.requests[0] < t - 3000) {

this.requests.shift();

}

return this.requests.length;

}

}

Example:

const recentCounter = new RecentCounter();

console.log(recentCounter.ping(1));

Output: 1

console.log(recentCounter.ping(100));

Output: 2

console.log(recentCounter.ping(3001));

Output: 3

console.log(recentCounter.ping(3002));

Output: 3

**5.**

class ListNode {

constructor(val) {

this.val = val;

this.next = null;

}

}

function findTheWinner(n, k) {

let head = new ListNode(1);

let prev = head;

for (let i = 2; i <= n; i++) {

let node = new ListNode(i);

prev.next = node;

prev = node;

}

prev.next = head;

let current = head;

while (current.next !== current) {

for (let i = 1; i < k; i++) {

current = current.next;

}

current.next = current.next.next;

}

return current.val;

}

// Example:

console.log(findTheWinner(5, 2));

Output: 3

console.log(findTheWinner(6, 5));

Output: 1

**6.**

function deckRevealedIncreasing(deck) {

const n = deck.length;

deck.sort((a, b) => a - b); // Sort the deck in increasing order

const queue = [];

for (let i = 0; i < n; i++) {

queue.push(i); // Initialize the queue with indices

}

const result = [];

while (queue.length > 0) {

result.push(queue.shift()); // Reveal the top card

if (queue.length > 0) {

queue.push(queue.shift()); // Move the next top card to the bottom

}

}

const orderedDeck = [];

for (const idx of result) {

orderedDeck[idx] = deck.shift(); // Map indices to corresponding cards

}

return orderedDeck;

}

Example:

console.log(deckRevealedIncreasing([17, 13, 11, 2, 3, 5, 7]));

Output: [2, 13, 3, 11, 5, 17, 7]

console.log(deckRevealedIncreasing([1, 1000]));

Output: [1, 1000]

**7.**

class FrontMiddleBackQueue {

constructor() {

this.front = [];

this.back = [];

}

pushFront(val) {

this.front.unshift(val);

this.balance();

}

pushMiddle(val) {

if (this.front.length > this.back.length) {

this.back.unshift(this.front.pop());

}

this.front.push(val);

}

pushBack(val) {

this.back.push(val);

this.balance();

}

popFront() {

if (this.front.length > 0) {

return this.front.shift();

} else if (this.back.length > 0) {

return this.back.shift();

}

return -1;

}

popMiddle() {

if (this.front.length > this.back.length) {

return this.front.pop();

} else if (this.back.length > 0) {

return this.back.shift();

}

return -1;

}

popBack() {

if (this.back.length > 0) {

return this.back.pop();

} else if (this.front.length > 0) {

return this.front.pop();

}

return -1;

}

balance() {

if (this.front.length > this.back.length + 1) {

this.back.unshift(this.front.pop());

} else if (this.back.length > this.front.length) {

this.front.push(this.back.shift());

}

}

}

**8.**

class DataStream {

constructor(value, k) {

this.value = value;

this.k = k;

this.stream = [];

}

consec(num) {

this.stream.push(num);

if (this.stream.length < this.k) {

return false;

}

const lastKIntegers = this.stream.slice(-this.k);

return lastKIntegers.every((integer) => integer === this.value);

}

}

const dataStream = new DataStream(4, 3);

console.log(dataStream.consec(4));

false

console.log(dataStream.consec(4));

false

console.log(dataStream.consec(4));

true

console.log(dataStream.consec(3));

false