**FULL STACK DEVELOPER ASSESSMENT**

**PART III (PLEASE ATTACH A SCREENSHOT OR PICTURE OF YOUR ANSWER ON PAGE 4)**

1. Write a JavaScript program to sort a list of elements using Quick sort. Quick sort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation (formally, a total order) is defined. Note: do not use the array native function “sort”
2. Write a JavaScript program to sort a list of elements using Insertion sort. Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. Note: do not use the array native function “sort”
3. Write a JavaScript program to sort a list of elements using Merge sort. According to Wikipedia "Merge sort (also commonly spelled mergesort) is an O (n log n) comparison-based sorting algorithm. Most implementations produce a stable sort, which means that the implementation preserves the input order of equal elements in the sorted output." Note: do not use the array native function “sort”
4. Write a JavaScript program to compare two objects to determine if the first one contains equivalent property values to the second one.
5. Write a JavaScript program to measure the time taken by a function to execute
6. Write a JavaScript program to get the volume of a Cylinder with four decimal places using object classes. Volume of a cylinder : V = πr2h

Where r is the radius and h is the height of the cylinder.

1. Write a JavaScript program to demonstrate a class inheritance.
2. Write a ReactJS code snippet on implementing props vs state. Put a comment accordingly
3. Write a ReactJS code snippet on implementing hooks. Put a comment accordingly
4. Write a ReactJS code snippet sample of stateless vs stateful component. Put a comment accordingly
5. Having this sample document

|  |
| --- |
| {  "\_id": MongoId("56902f7f31de51cdcfc03f07"),  "post": "Hello World",  "likes": 10,  "timestamp": 1572266114118 } |

Create an aggregation query to get the average ‘like’ of post per day

|  |
| --- |
| {  "\_id": {"month": 10, "day": 20, "year": 2019},  "totalLikes": 100  } |

1. Having this sample document

|  |
| --- |
| {  "\_id": MongoId("56902f7f31de51cdcfc03f07"),  "sms": "Привет, мир",  "recipient": "+74457771234",  "timestamp": 1572266114118 } |

Create an aggregate query that will count the total number of SMS by 132 bytes per count

|  |
| --- |
| {  "\_id": null,  "smsCount": 100 } |
|  |
|  |

1. Having this document, create an update query to increase the count of an item(Apple) inside an array

|  |
| --- |
| {  "\_id": MongoId("56902f7f31de51cdcfc03f07"),  "items": [  {  "code": 9643372659,  "name": "Apple",  "count": 100  },  {  "code": 7969928269,  "name": "Orange",  "count": 100  }  ] } |
|  |

1. Design a sample JSON document that will represent a One-to-Many relationship in a non relational database such as MongoDB.
2. Design a sample JSON document that is capable of efficient querying. Meaning that if the whole collection reach more than 10M, its execution time is still less than 1sec. Example a query needs to fetch the last 100 added documents in the collection. Note: also include the index json and the index type, and assume that the collection is non capped collection type

ANSWER SHEET:

# Quicksort

function selectPivot(numList) {

    let i = Math.floor((numList.length - 1) / 2);

    return i;

}

function swapPivotToEnd(numList, pivotIndex) {

    // swaps the locations of pivot to the end of the array

    [numList[pivotIndex], numList[numList.length - 1]] = [numList[numList.length - 1], numList[pivotIndex]];

}

function locateFromLeft(numList, pivot) {

    // returns index when item greater than pivot has been found

    for(let i = 0; i < numList.length - 1; i++){

        if(numList[i] > pivot){

            return i;

        }

    }

    return numList.length - 1;

}

function locateFromRight(numList, pivot) {

    // returns index when item lesser than pivot has been found

    // start at numlist.length - 2 since numlist.length - 1 is the pivot itself

    for(let i = numList.length - 2; i >= 0; i--){

        if(numList[i] < pivot){

            return i;

        }

    }

    return null;

}

function isSorted(numList){

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

        if(numList[i - 1] > numList[i]){

            return false;

        }

    }

    return true;

}

function quickSort(numList) {

    // checks to end recursion

    if(isSorted(numList)){

        return numList;

    } else if(numList.length == 2){ // edge case for only 2 numbers

        [numList[0], numList[1]] = [numList[1], numList[0]];

        return numList

    }

    let pivotIndex = selectPivot(numList);              // 1) select pivot index at center

    swapPivotToEnd(numList, pivotIndex);                // 2) swap index of pivot to end of array

    // pivot is located at the end here

    pivotIndex = numList.length - 1;            // 3) take new index of pivot

    let pivot = numList[pivotIndex];

    let leftItemIndex = 0;

    let rightItemIndex = 1;

    // swaps number from left index with number from right index

    // ends when left index > right index

    while (leftItemIndex < rightItemIndex){             // 4) sort

        leftItemIndex = locateFromLeft(numList, pivot);

        rightItemIndex = locateFromRight(numList, pivot);

        // swap left item and pivot location

        if(leftItemIndex < rightItemIndex){

            [numList[leftItemIndex], numList[rightItemIndex]] = [numList[rightItemIndex], numList[leftItemIndex]];

        }

    }

                                                        // 5) when left index overtakes right, swap pivot with item in left index

    [numList[leftItemIndex], numList[pivotIndex]] = [numList[pivotIndex], numList[leftItemIndex]];

    // To track the location of the pivot after the swap

    let pivotCorrectIndex = leftItemIndex;

    // 6) recursively call function until all are sorted

    let preArray = numList.slice(0, pivotCorrectIndex);

    let postArray = numList.slice(pivotCorrectIndex+ 1, numList.length);

    let outputArray = quickSort(preArray).concat(

        [pivot],

        quickSort(postArray)

        );

    return outputArray;

}

function main() {

    let inputList = [2, 6, 5, 3, 8, 7, 1, 0];           // given

    let inputList2 = [4, 4, 7, 9, 1, 3, 0];

    let output = quickSort(inputList2);

    console.log(output);

}

# Insertion Sort

function shimmyLeft(numList, numlistIndex){

    // i > 0 since code should stop when last item has been compared

    for(let i = numlistIndex; i > 0; i--){

        if(numList[i] < numList[i - 1]){

            [numList[i], numList[i - 1]] = [numList[i - 1], numList[i]];

        }

    }

}

function quickSort(numList){

    // i = 1 since quicksort compares with left index. Prevents null pointer exception

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

        if(numList[i] < numList[i - 1]){

            shimmyLeft(numList, i);

        }

    }

}

function main(){

    let input = [4, 2, 7, 9, 1, 3];

    let input2 = [13, 24, 4, 7, 36, 2, 1]

    quickSort(input2);

    console.log(input2);

}

main();

# Mergesort

function merge(arr1, arr2, arrOrig){

    // indeces of arr1 and arr2

    let i = 0, i1 = 0, i2 = 0;

    while(i1 < arr1.length && i2 < arr2.length){

        if(arr1[i1] < arr2[i2]){

            arrOrig[i] = arr1[i1];

            i++;

            i1++;

        } else {

            arrOrig[i] = arr2[i2];

            i++;

            i2++;

        }

    }

    // edge cases when the first while loop exits and there are still items in the arrays

    while(i1 < arr1.length){

        arrOrig[i] = arr1[i1];

        i++;

        i1++;

    }

    while(i2 < arr2.length){

        arrOrig[i] = arr2[i2];

        i++;

        i2++;

    }

    return arrOrig;

}

function mergeSort(arrOrig){

    // returns when list length == 1

    if(arrOrig.length == 1){

        return arrOrig

    }

    let arr1 = arrOrig.slice(0, Math.floor(arrOrig.length/2));

    let arr2 = arrOrig.slice(Math.floor(arrOrig.length/2), arrOrig.length);

    arr1 = mergeSort(arr1);

    arr2 = mergeSort(arr2);

    return merge(arr1, arr2, arrOrig);

}

function main(){

    let input = [4, 2, 7, 9, 1, 3];

    let input2 = [13, 24, 24, 7, 36, 2, 1];

    let sortedList = mergeSort(input2);

    console.log (sortedList);

}

main();

# Object Equivalence

function isEqual(obj1, obj2){

    let obj1Properties = Object.keys(obj1);

    let obj2Properties = Object.keys(obj2);

    for(let property in obj1){

        if(obj1[property] == obj2[property]){

            return true;

        }

    }

    return false;

}

# Performance

function hello(){

    console.log("Hello World");

}

function performanceCheck(f){

    let start = performance.now();

    f();

    let end = performance.now();

    return end - start;

}

let perf = performanceCheck(hello);

console.log(perf);

# Cylinder

function getCylinderVolume(cylinder){

    let radius = cylinder['radius'];

    let height = cylinder['height'];

    return (Math.PI \* radius \* 2 \* height).toFixed(4);

}

let cylinder = {

    radius: 24,

    height: 50

}

# Inheritance

class Hero {

    constructor(name, race){

        this.name = name;

        this.race = race;

    }

    attack(){

        console.log("I, " + this.name + ", attack!");

    }

}

class Warrior extends Hero {

    constructor(name, race){

        super(name, race);

        this.attackPower = 50;

        this.defense = 100;

    }

    attack(){

        super.attack();

        this.swingSword();

    }

    swingSword(){

        console.log("... with my special move Sword Spin!");

    }

}

class Wizard extends Hero {

    constructor(name, race){

        super(name, race);

        this.attackPower = 100;

        this.defense = 50;

    }

    attack(){

        super.attack();

        this.castMagic();

    }

    castMagic(){

        console.log("... using my magic! Avada Kedavra!");

    }

}

ANSWERS: