**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:

ANSWERS:

**1.**

function quickSort(arr) {

if (arr.length < 2) {

return arr;

} else {

const pivot = arr[0];

const lesser = arr.slice(1).filter((el) => el <= pivot);

const greater = arr.slice(1).filter((el) => el > pivot);

return quickSort(lesser).concat(pivot, quickSort(greater));

}

}

const unsortedArray = [3, 5, 2, 1, 4];

console.log(quickSort(unsortedArray));

**2.**

function insertionSort(arr) {

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

let current = arr[i];

let j = i - 1;

while (j >= 0 && arr[j] > current) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = current;

}

return arr;

}

let unsortedArray = [3, 5, 2, 1, 4];

console.log(insertionSort(unsortedArray));

**3.**

function mergeSort(arr) {

if (arr.length <= 1) return arr;

const mid = Math.floor(arr.length / 2);

const left = arr.slice(0, mid);

const right = arr.slice(mid);

const sortedLeft = mergeSort(left);

const sortedRight = mergeSort(right);

const merged = [];

let leftIndex = 0;

let rightIndex = 0;

while (leftIndex < sortedLeft.length && rightIndex < sortedRight.length) {

if (sortedLeft[leftIndex] < sortedRight[rightIndex]) {

merged.push(sortedLeft[leftIndex]);

leftIndex++;

} else {

merged.push(sortedRight[rightIndex]);

rightIndex++;

}

}

return merged.concat(sortedLeft.slice(leftIndex), sortedRight.slice(rightIndex));

}

const unsortedArray = [3, 5, 2, 1, 4];

console.log(mergeSort(unsortedArray));

**4.**

function objectsAreEqual(obj1, obj2) {

const obj1Keys = Object.keys(obj1);

const obj2Keys = Object.keys(obj2);

if (obj1Keys.length !== obj2Keys.length) return false;

for (let key of obj1Keys) {

if (!obj2.hasOwnProperty(key)) return false;

if (obj1[key] !== obj2[key]) return false;

}

return true;

}

const theObject = { x: 1, y: 2, z: 3 };

const objectSameAsTheFirstObject = { x: 1, y: 2, z: 3 };

const entirelyDifferentObject = { a: 1, b: 2, c: 3 };

console.log(objectsAreEqual(theObject, objectSameAsTheFirstObject));

console.log(objectsAreEqual(theObject, entirelyDifferentObject));

5.

function timeTheFunction(fn) {

const start = performance.now();

fn()

const end = performance.now();

console.log(`Function ${fn.name} took ${end - start} milliseconds to execute.`);

}

const helloWorld = () => console.log("hello world");

timeTheFunction(helloWorld);

6.

class Cylinder {

constructor(radius, height) {

this.radius = radius;

this.height = height;

}

getVolume = () => (Math.PI \* this.radius \*\* 2 \* this.height).toFixed(4);

}

console.log(new Cylinder(2, 5).getVolume());

7.

class Animal {

speakString = "says";

constructor(name, noise) {

this.name = name;

this.noise = noise;

}

speak = () => console.log(`${this.name} ${this.speakString}: ${this.noise}! ${this.noise}!`);

}

class Dog extends Animal {

speakString = "barks"

}

class Cat extends Animal {

speakString = "meows"

}

const dog = new Dog("Denji", "Arf");

const cat = new Cat("Power", "Meow");

dog.speak();

cat.speak();

8.

import React, { useState } from 'react';

function HelloOrHiWorld(props) {

const [isHello, setIsHello] = useState(true);

const toggleHello = () => {

setIsHello(!isHello);

}

return (

<div>

{/\* this is the prop, passed, but we are not mutating it here, we are simply displaying it

if there are changes needed, It needs to be mutated in the parent and passed here to this component \*/}

<h1>{props.title}</h1>

{/\* this is the state, we can mutate it here \*/}

<p>{isHello ? "Hello" : "Hi"} World!</p>

{/\* in this line we are toggling isHello to show either hello or hi \*/}

<button onClick={toggleHello}>click me</button>

</div>

);

}

export default HelloOrHiWorld;

9. Note: im using the same snippet that I submitted in number 8 because Its also using hooks anyway :).

import React, { useState } from 'react';

function HelloOrHiWorld(props) {

const [isHello, setIsHello] = useState(true);

const toggleHello = () => {

setIsHello(!isHello);

}

return (

<div>

<h1>{props.title}</h1>

{/\* we are mutating isHello using useState hook \*/}

<p>{isHello ? "Hello" : "Hi"} World!</p>

{/\* in this line we are toggling isHello to show either hello or hi \*/}

<button onClick={toggleHello}>click me</button>

</div>

);

}

export default HelloOrHiWorld;

10.

import React, { useState } from 'react';

// STATELESS

// in this example we cannot change the "Hello world" string because it's stateless

function HelloWorld() {

return (

<div>

<p>Hello World!</p>

</div>

);

}

export default HelloWorld;

//STATEFUL

// in this example we can show either "Hello world" or "Hi world" by clicking the button

function HelloOrHiWorld() {

const [isHello, setIsHello] = useState(true);

const toggleHello = () => {

setIsHello(!isHello);

}

return (

<div>

{/\* this is the state, we can mutate it here \*/}

<p>{isHello ? "Hello" : "Hi"} World!</p>

{/\* in this line we are toggling isHello to show either hello or hi \*/}

<button onClick={toggleHello}>click me</button>

</div>

);

}

export default HelloOrHiWorld;

11.

db.posts.aggregate([

{

$project: {

year: { $year: { $toDate: "$timestamp" } },

month: { $month: { $toDate: "$timestamp" } },

day: { $dayOfMonth: { $toDate: "$timestamp" } },

likes: "$likes"

}

},

{

$group: {

\_id: { year: "$year", month: "$month", day: "$day" },

totalLikes: { $sum: "$likes" }

}

},

{

$project: {

\_id: 1,

totalLikes: 1

}

}

]);

12.

db.sms.aggregate([

{

$project: {

smsLength: { $strLenCP: "$sms" }

}

},

{

$group: {

\_id: null,

smsCount: { $sum: { $ceil: { $divide: [ "$smsLength", 132 ] } } }

}

},

{

$project: {

\_id: 0,

smsCount: 1

}

}

]);

13

db.inventory.updateOne(

{ "items.name": "Apple" },

{ $inc: { "items.$.count": 1 } }

);

14

{

"\_id": ObjectId("615f74cf771d2eb28565fae2"),

"name": "John Doe",

"email": "jerome@example.com",

"comments": [

{

"\_id": ObjectId("615f74cf771d2eb28565fae3"),

"text": "sauce?",

"date": ISODate("2022-02-17T12:00:00Z")

},

{

"\_id": ObjectId("615f74cf771d2eb28565fae4"),

"text": "What’s the sauce?",

"date": ISODate("2022-02-18T10:30:00Z")

}

]

}

15.  
{

"\_id": ObjectId("61e9f7d43f54d55eeb183d29"),

"title": "Document",

“indexWhenInserted”: 10000000,

"created\_at": ISODate("2022-01-20T12:00:00Z")

}

Then we can query them by “indexWhenInserted” in descending order, this way we don’t have to sort the entire collection.  
I know this is a bit hacky but I just wanted to show the basic concept of what I’m thinking as simple as possible.