Microservices: it is a controller for distributed systems, where a architect develops a application in order handle the different servers.

***a service-oriented application component that is tightly scoped, strongly encapsulated, loosely coupled, independently deployable and independently scalable***

Microservices architectures, or simply Microservices is an SDLC approach based on which larger applications are built as a collection of small functional modules. These functional modules are independently deployable, scalable, target specific business goals, and communicate with each other over standard protocols like HTTP request/response with resource APIs and lightweight asynchronous messaging.

***The project I have worked is great example for microservice, it is large scale data storage application built on small functional modules (like creating pools, volumes , hosts, and volume groups ) , each module is scalable , independently deployable, functional, target specific goals and communicate with each other with HTTP.***

## ***Time complexity:***

Constant Time Complexity: O(1) – when only one statement executes in program

**Linear Time Complexity: O(n) – when programs iterates through loop n times -** more efficient and faster

**Quadratic Time Complexity: O(n²) -** Nested **For Loops** run on quadratic time – less efficient, and their performance can degrade quickly as the input size grows

**Exponential Time Complexity: O(2^n) - Brute-Force algorithms – loop iterate growth rate increases.**

**Logarithmic Time Complexity: O(log n) -** inversely proportional to the input “n” or “Divide and Conquer” – **Two pointer algorithms**

## **Data Structure**

**STRINGS, ARRAYS ,LIST, MAP, SET**

Maps – doesn’t allows the duplicate keys, to get sorted map use Treemap – no null treemap

Set - doesn’t allows the duplicate values, , to get sorted set use Treeset- no null in treemap

### **Set** -JAVA

Interface Set interface extends Collection interface. In a set, no duplicates are allowed. Every element in a set must be unique.

* HashSet does not guarantee any order
* LinkedHashSet maintain insertion order
* TreeSet maintain sorting order
* HashSet and LinkedHashSet allows only one null
* TreeSet does not allow null
* All three are not thread-safe

### Maps: JS

Based on the concept of key and value pair.

Properties:

* Keyes can’t be duplicated, has to be unique.
* INSERT is not possible if the key already presents.
* UPDATE value for a key
* DELETE based on key
* Position doesn’t matter

Obj[“key”] =17;

Object.keys(obj);

**JAVA- put(), remove,clear(),containsKey(),containsValue(), get()**

**for(int i: arra){}**

typeOf(variable\_ref); > returns the data type of variable

parseInt(variable\_ref); > returns the NUMBER data type ;

Floor>Math.floor(1.2) > 1

Floor means closet integer lesser than the given number.

Ceil> Math.ceil(3.5) > 4

Ceil means closet integer greater than the given number.

class Car {

constructor(name, year) {

this.name = name;

this.year = year;

}

}

const myCar1 = new Car("Ford", 2014);

const myCar2 = new Car("Audi", 2019);

<script>

const person = {

firstName : "John",

lastName : "Doe",

age : 50,

eyeColor : "blue"

};

document.getElementById("demo").innerHTML = person.firstName + " " + person.lastName;

</script>

Array:

Homogeneous Data

Constant size, go for it when the known quantity of items.

Accessed and stored data using index.

Int array[] = new int[]; || var array[] = new Array();

Length is property of Array class.

Push – unshift -adding an element pop-shift -to delete an element.

## **Brute Force:**

Two loops, quadratic TC > O(n\*2)

 int result=1;

        int count=0;

        for(int i=1;i<n;i++){*//7 4 10 9 6 1 8 2 5 3*

            for(int j=i-1;j>=0;j--){

                if(nums[i]!=nums[j]){

                    count++;

                }

            }

            if(count == i){

                result ++;

            }

            count =0;

        }

        return result;

## Two pointer approach

Two pointers, one loop , logarithmic TC > O(log N)

## Optimized approach

Sort the data structure + loop ,O(nlogn)

# **Binary Search:**

In binary search:

* There is one primary pointer (usually called "mid" or "pivot") that divides the search space into two halves.
* It repeatedly adjusts this primary pointer based on a comparison between the target value and the value at the midpoint.
* The search space is divided into two halves, and one of them is eliminated at each iteration.

You can efficiently find the target value in a sorted array of distinct integers with a binary search algorithm, which has a time complexity of O(log N) logarithmic time complexity for sorted arrays. Here's a Java example of how to do this:

int left = 0;

        int right = nums.length - 1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            if (nums[mid] == target) {

                return mid; *// Target found, return its index.*

            }

            if (nums[mid] < target) {

                left = mid + 1; *// Target is on the right side.*

            } else {

                right = mid - 1; *// Target is on the left side.*

            }

        }

        return -1; *// Target not found in the array.*

    }

### Example: HashMap and ArrayList program

TO find intersection elements of two arrays

 static List<Integer> intersectionOfTwoArraysBasic(int n, List<Integer>a, int m, List<Integer>b){

    List<Integer> result = new ArrayList<>();

*// Time complexity - O(n+m)*

*//use a for each loop and Hashmap(duplicate keys not allowed)*

*//check the value of a[i] not a key in map*

*//if yes put new key in map, else update the exiting key with old+1;*

    Map<Integer,Integer> freq = new HashMap<Integer,Integer>();

    int count =0;

    for(int i: a){

        if(!freq.containsKey(i)){

            freq.put(i,1);

        }

        else{

        freq.put(i,freq.get(i)+1);

        }

    }

*//iterating through the b[] using for each loop*

*// check the element of b being as key in map or not*

*// if yes check the accurance count >0*

*//if yes decrement the its value and add to the result list*

    for(int i: b){

        if(freq.containsKey(i)){

            if(freq.get(i)>0){

            result.add(i);

            freq.put(i,freq.get(i)-1);

            }

        }

    }

*//sort the sort list and return*

        Collections.sort(result);

        return result;

    }