Assignment 1

Name : Hrishikesh Rajan
Email : hrishikeshrajan3@gmail.com
LinkedIn : https://www.linkedin.com/in/hrishikesh-rajan-96aa70165

Question 2)
You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad. Suppose you have a version and you want to find out the first bad one,which causes all the following ones to be bad. Also, talk about the time complexity of your code.

Test Cases:
Input: [0,0,0,1,1,1,1,1,1]
Output: 3

Explanation: 0 indicates a good version and 1 indicates a bad version. So, the index of the first 1 is at 3. Thus, the output is 3

*A Javascript implementation is provided

Answer :

```
function findBadVersion(arr,low,high,goodVersion,badVersion){

   if(low<=high){
    let mid =  Math.floor(low + ((high - low)/2));

    if((goodVersion === arr[mid-1] && badVersion === arr[mid]) || arr[mid]=== badVersion &&
arr.length ===1){
        return mid;
    }
    if(badVersion > arr[mid] ){
        return findBadVersion(arr,mid+1,high,goodVersion,badVersion);
    }
    if(badVersion <= arr[mid]){
        return findBadVersion(arr,low,mid-1,goodVersion,badVersion);
    }
   }
return -1;
}

const arr = [0,0,0,1,1,1,1,1,1];
const result = findBadVersion(arr,0,arr.length-1,0,1)
console.log(result)
```

Output : 3

Time Complexity :

The recurrence relation is formed by

T(n) = T(n/2) + c , c = constant. (1)

The reason for T(n/2) is that at a time our search space was limited to half of n, that is n/2, where n is the total length of the array.

Applying Master's Theorem

T(n) = aT(n/b) + f(n) , where f(n) = Θ ($n^{k} log^{p} n$) (2)

From the above recurrence relation (1)
a=1
b=2
k=0
p=0

Substituting values

log_{a}^{b} **=** log_{2}^{1} **=** 0

That means log_{a}^{b} **= k** and **P > -1,** This relation comes under case 2.

Then the equation becomes Θ ($n^{k} log^{p+1} n$)

= Θ ($n^{0} log^{0+1} n$)

= Θ ($1 * log^{1} n$)

= Θ ($log\ n$)

Time Complexity = O(logn)
Space Complexity = O(1)