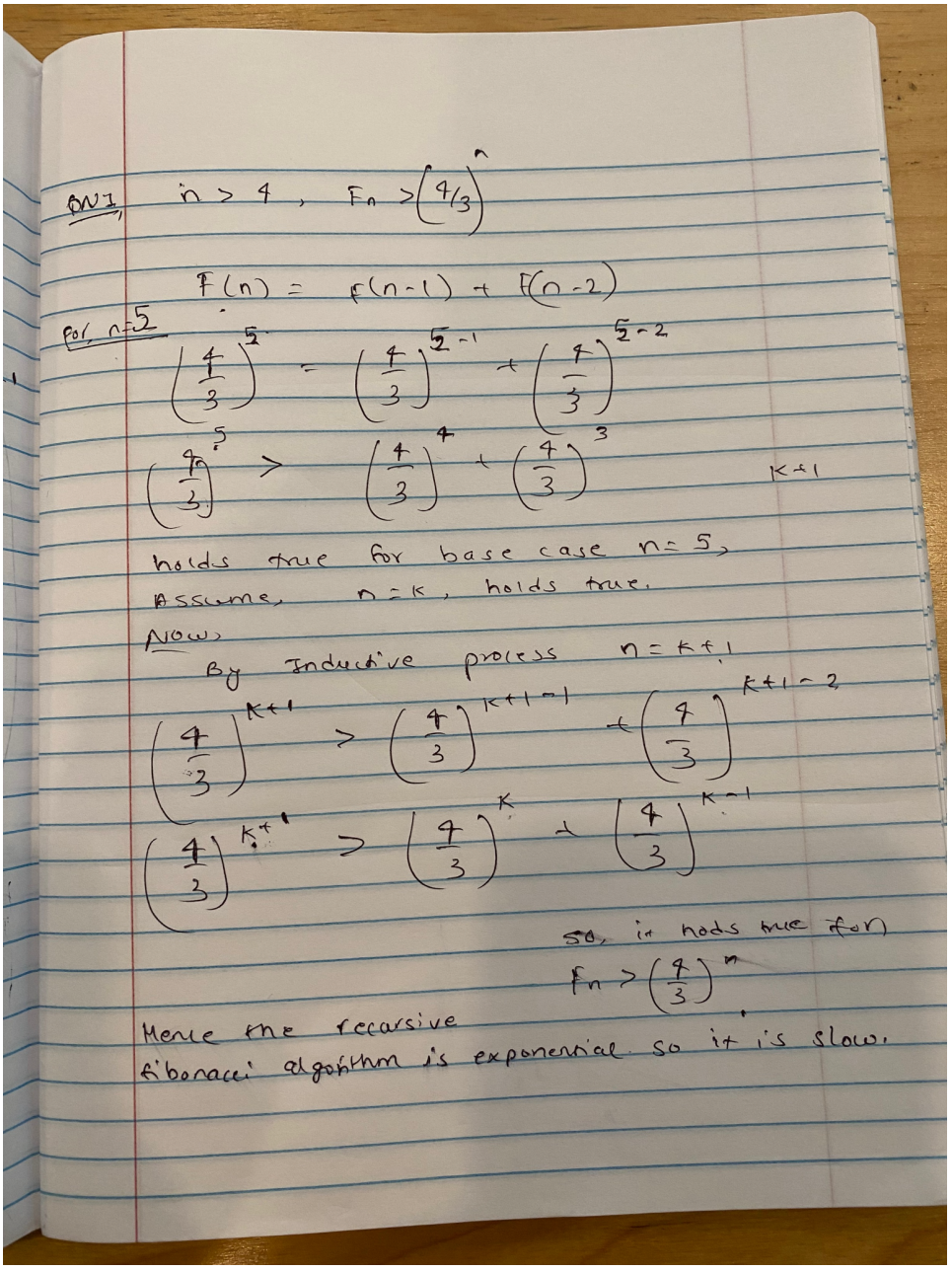
Lab 2 Continue

Q1.



Q2.

A. 4^n is O(2^n) : **FALSE**

$$\lim_{n \to \infty} \frac{4^n}{2^n}$$

$$\lim_{n \to \infty} \frac{2^{2n}}{2^n}$$

$$\lim_{n \to \infty} 2^{2n-n}$$

$$\lim_{n \to \infty} 2^n$$

$$c.2^{\infty}$$

$$\infty$$

B. $logn$ is $\Theta\ log_3 n$: **TRUE**

For, f(n) / g(n)

$$\lim_{n \to \infty} \frac{\log n}{\log_3(n)}$$

$$\lim_{n \to \infty} \frac{\frac{1}{n}}{\frac{1}{n \log_3}}$$

$$\lim_{n \to \infty} \frac{n \log 3}{n}$$

$$\lim_{n \to \infty} \log 3$$

log3 > 0

For, g(n)/f(n)

$$\lim_{n \to \infty} \frac{\log_3(n)}{\log n}$$

$$\lim_{n \to \infty} \frac{\frac{1}{n \log_3}}{\frac{1}{n}}$$

$$\lim_{n \to \infty} \frac{n}{n \log 3}$$

$$\lim_{n \to \infty} \frac{1}{\log 3}$$

1/log3 > 0

C. n/2 log(n/2) is Θ(nlogn) : **TRUE**

For, f(n) / g(n)

$$\lim_{n \to \infty} \frac{\frac{n}{2} \log\left(\frac{n}{2}\right)}{n \log n}$$

$$\lim_{n \to \infty} \frac{\frac{1}{2} \cdot \frac{1}{2} \frac{1}{\left(\frac{n}{2}\right)}}{\frac{1}{n}}$$

$$\frac{1}{2}$$

1/2 > 0

For, g(n)/f(n)

$$\lim_{n \to \infty} \frac{n \log n}{\frac{n}{2} \log\left(\frac{n}{2}\right)}$$

$$\lim_{n \to \infty} \frac{\frac{1}{n}}{\frac{1}{2} \cdot \frac{1}{2} \frac{1}{\left(\frac{n}{2}\right)}}$$

$$\frac{\frac{1}{n}}{\frac{2}{2n} \cdot \frac{1}{2}}$$

$$\frac{1}{2}$$

1/2 > 0

Therefore, n/2 log(n/2) is Θ(nlogn)

Q3.

A. Guessing Method

T(n) = 5 + T(n-1);
T(1) = 5;
T(2) = 5+T(2-1) = 5 + T(1) = 5 + (5);
T(3) = 5+T(3-1) = 5 + T(2) = 5 + (5+5); => 5+ (3-1)x5
T(4) = 5+T(4-1) = 5 + T(3) = 5 + (5+5+5); => 5+ (4-1)x5
...
T(m) = 5+ (m-1)5;
T(n) = 5+ (n-1)5 = 5n; which is O(n)

B. Proof:

From the above equation,
f(n) = 5n is a solution of recurrence T(n) = 5+T(n-1);
Therefore we must show,
=> f(1) = 5 and
=> f(n) = f(n-1)+5;
We have,

f(n) = 5n

= 5n - 5 + 5;
= 5(n-1) + 5;
= f(n-1) + 5; as f(n) = 5(n);

Which is required and proved.

Q4.

```
static List<Integer> items=new ArrayList<>(Arrays.asList(0,1));

static List<Integer> fib(int k){
   for(int i=2;i<k;i++){
       items.add(items.get(i-1)+items.get(i-2));
   }
   return items;
}
```

The given algorithm iterates kth times where k is the input. So the running complexity is O(n).

Q5.
We have T(n) = T(n/2) +n; T(1) =1
Using Master Formula equation, we get
a = 1, b =2, k =1 and b^k = 2

Here we have a < b^k, Therefore, Master Formula says, T(n) = $\Theta(n)$;

Q6.

```
public class ZerosAndOnesInArray {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int[] result = zerosAndOnes(new int[] {0,0,0,1,1,1,1});
        System.out.println(Arrays.toString(result));

        int[] binaryResult = findZerosAndOnes(new int[] {0,0,0,1,1,1,1},0,7);
        System.out.println(Arrays.toString(binaryResult));

    }
    //Binary Search to reduce complexity
    static int[] findZerosAndOnes(int[] A,int lower, int upper) {

        int[] result = new int[2];
        int mid = (upper + lower)/2;
        if(A[mid] == 0 && A[mid+1] == 1) {
            result[0] = mid +1;
            result[1] = A.length - (mid+1);
                return result;
        }
        if(A[mid] == 0 && A[mid+1] == 0) {
            return findZerosAndOnes(A, mid+1, upper);
        }
        if (A[mid] == 1) {
            return findZerosAndOnes(A, lower, mid -1);
        }
        return null;
    }

    //Time Complexity is O(logn) as Binary Search algorithm is used. Hence algorithm runs in small o(n) time;


    static int[] zerosAndOnes(int[] sortedArray) {
        int[] countsOfZerosAndOnes = new int[2];
        for(int i =0 ; i < sortedArray.length;i++ ) {
            if(sortedArray[i] == 0) {
```

```
public class ZerosAndOnesInArray {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int[] result = zerosAndOnes(new int[] {0,0,0,1,1,1,1});
		System.out.println(Arrays.toString(result));

		int[] binaryResult = findZerosAndOnes(new int[] {0,0,0,1,1,1,1},0,7);
		System.out.println(Arrays.toString(binaryResult));

	}
	//Binary Search to reduce complexity
	static int[] findZerosAndOnes(int[] A,int lower, int upper) {

		int[] result = new int[2];
		int mid = (upper + lower)/2;
		if(A[mid] == 0 && A[mid+1] == 1) {
			result[0] = mid +1;
			result[1] = A.length - (mid+1);
				return result;
		}
		if(A[mid] == 0 && A[mid+1] == 0) {
			return findZerosAndOnes(A, mid+1, upper);
		}
		if (A[mid] == 1) {
			return findZerosAndOnes(A, lower, mid -1);
		}
		return null;
```

```
	}

	//Time Complexity is O(logn) as Binary Search algorithm is used. Hence
algorithm runs in small o(n) time;


	static int[] zerosAndOnes(int[] sortedArray) {
		int[] countsOfZerosAndOnes = new int[2];
		for(int i =0 ; i < sortedArray.length;i++ ) {
			if(sortedArray[i] == 0) {
				++countsOfZerosAndOnes[0];
			}
			else {
				++countsOfZerosAndOnes[1];
			}
		}
		return countsOfZerosAndOnes;
	}
	//Time Complexity is O(n) where n: length of array since it iterates through
length of array

}
```