Lab 2

Q1. Runtime complexity of given program

1st Single for Loop: O(n)
2nd Nested for Loop: O(n^2)

I.e O(n) + O(n^2)
Considering only higher order terms, O(n^2)

Q2. Merging two sorted arrays

A. *Algorithm description using the pseudo-code*

- Create final array of size of sum (arr1 length + arr 2 length)
- Initialize pointers for arr1 and arr2, with initial value at 0. And initial starting pointer for final array as index 0.
- Begin
    - Validate If both the pointers are less than the respective array length
        - Compare arr1 pointer value with arr2 pointer value
        - If arr1 pointer value is < then arr 2 pointer value, then take arr1 value and put it into finalArr, and increment arr1 pointer.
        - Else, take arr2 pointer value, put in finalArr, and increment arr2 pointer.
    - After adding element, from either arr1 or arr2. Increment finalArr index by 1.
- Once either pointer arr1 or arr2 is fully checked
    - Check if arr1 pointer is not fully completed, then Begin
        - Copying all remaining arr1 values to final array.
    - Else check if arr2 pointer is not fully checked, then Begin
        - copying all remaining arr2 values to final array.
- Return the final Array.

B. *Running time complexity :* **O(m + n)**

I.e m as arr1 and n as arr2

C. *Program in Java*

```
public static void main(String[] args) {
```

```
    int[] resultArr = mergeArray(new int[]{1, 4, 5, 8, 17}, new int[]{2,
4, 8, 11, 13, 21, 23, 25});
    System.out.println(Arrays.toString(resultArr));
}

public static int[] mergeArray(int[] a, int[] b) {
    int finalArr[] = new int[a.length + b.length];

    int aPointer = 0;
    int bPointer = 0;
    int index = 0;
    while (aPointer < a.length && bPointer < b.length) { // O(min(a,b))
        if (a[aPointer] < b[bPointer]) {
            finalArr[index] = a[aPointer];
            aPointer++;
        } else {
            finalArr[index] = b[bPointer];
            bPointer++;
        }
        index++;
    }
    while (aPointer < a.length) {
        finalArr[index] = a[aPointer];
        aPointer++;
        index++;
    }
    while (bPointer < b.length) {
        finalArr[index] = b[bPointer];
        bPointer++;
        index++;
    }
    return finalArr;
}
```

Q3. Using O(f(n)) and limit facts, decide true or false for given case

A. 1 + 4n^2 is O(n2) : True

$$\lim_{n \to \infty} \frac{1 + 4n^2}{n^2}$$

$$\frac{0 + 8n}{2n}$$

$$\frac{8n}{2n}$$

$$4$$

As 4 > 0, $1 + 4n^2$ *is big* $O(n^2)$. True

B. n 2 - 2n is not O(n) : True

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

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

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

$$\frac{0}{2}$$

$$0$$

As f(n) is not <= g(n) , $n^2 - 2n$ *is not big* $O(n)$
So, this is True.

C. log(n) is o(n) : True

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

$$\frac{\frac{1}{n}}{1}$$

$$\frac{1}{n}$$

$$\frac{1}{\infty}$$

$$0$$

As 0 <= 0, $logn\ is\ a\ little\ o(n)$. True

D. n is not o(n) : True

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

$$1$$

As 1 > 0, $n\ is\ a\ big\ O(n)$

So, is not little o(n). True

Q4.

```
public static void main(String[] args) {
   List<Integer> items=new ArrayList<>();
   items.addAll(Arrays.asList(1,2,3));
   System.out.println(powerSet(items));
}


static <T>List powerSet(List<T> items){
   List<Set<T>> powerSetList=new CopyOnWriteArrayList<>();
   Set<T> emptySet=new HashSet();
   powerSetList.add(emptySet);
   while(!items.isEmpty()){
        T rem=items.remove(0);
        for(Set<T>  it : powerSetList){
            Set<T> ns=new HashSet();
            ns.addAll(it);
            ns.add(rem);
            powerSetList.add(ns);
```

```
        }
    }
    return powerSetList;
}
```