CCC2113 Data Structures and Algorithms Analysis Name:

Lab 3: Algorithm Complexity (25/4/2024) Group:

Question 1: Find the time complexity of the following code segments and write it in Big O notation:

1.	`public class Sort {`	1
2.		
3.	`    // Bubble sort algorithm`	
4.	`    public static void sort(int[] arr) {`	1
5.	`        int n = arr.length;`	
6.	`        for (int i = 0; i < n - 1; i++) {`	n
7.	`            for (int j = 0; j < n - i - 1; j++) {`	n(n-1)
8.	`                // Swap if the element found is greater than the next element`	
9.	`                if (arr[j] > arr[j + 1]) {`	n*
10.	`                    int temp = arr[j];`	
11.	`                    arr[j] = arr[j + 1];`	
12.	`                    arr[j + 1] = temp;`	
13.	`                } } } }`	
14.	`    public static void main(String[] args) {`	1
15.	`        int[] arr = {64, 34, 25, 12, 22, 11, 90}; // Example input`	1
16.	`        System.out.println("Array before sorting:");`	1
17.	`        printArray(arr);`	1
18.	`        sort(arr);`	1
19.	`        System.out.println("\nArray after sorting:");`	1
20.	`        printArray(arr);`	1
21.	`    }`	
22.	`    public static void printArray(int[] arr) {`	1
23.	`        for (int i = 0; i < arr.length; i++) {`	n+1
24.	`            System.out.print(arr[i] + " ");`	n
25.	`        }`	
26.	`        System.out.println();`	1
27.	`    }`	
28.	`}`	

Complexity in Big O: **O(n)**

Explain: _____

Question 2: Find the time complexity of the following code segments and write it in Big O notation:

1.	`public void complexity(int n) {`	1
2.	`    for (int j = 1; j <= n; j++) {`	n
3.	`        int i = n;`	n
4.	`        while (i > 1) {`	n.log(n)
5.	`            // Print the product of current row and column`	
6.	`            System.out.print(i * j + "\t");`	n.log(n)
7.	`            i = i / 2;`	n.log(n)
8.	`        }`	
9.	`        System.out.println();`	n
10.	`    } }`	

Complexity in Big O: **O(n.log(n))**

Explain: _____

Question 3:

The following code calculates the powerset of a given set of numbers. For example:

Input: arr = [1,2,3]

Output: subset = [[ ], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]

Code:

```
1. public class PowerSet {                                                    1
2.
3.   `   public static void generatePowerSet(int[] arr, int index, int[]
   currentSubset, ArrayList<ArrayList<Integer>> subsets) {                    1
4.         if (index == arr.length) {                                         n
5.             subsets.add(new ArrayList<>(Arrays.asList(currentSubset))); //
   Add current subset to final result                                         n
6.             return;                                                        n
7.         }
8.
9.         // Include current element in the subset
10.        currentSubset[index] = arr[index];                                 1
11.        generatePowerSet(arr, index + 1, currentSubset, subsets);          1
12.
13.        // Exclude current element from the subset
14.        currentSubset[index] = 0; // Assuming 0 represents absence in the
   subset
15.        generatePowerSet(arr, index + 1, currentSubset, subsets);
16.    }
17.
18.    public static void main(String[] args) {                               1
19.        int[] arr = {1, 2, 3};                                             1
20.        ArrayList<ArrayList<Integer>> subsets = new ArrayList<>();         1
21.        generatePowerSet(arr, 0, new int[arr.length], subsets);            1
22.        System.out.println("Power Set: " + subsets);                        1
23.    }
24.}
```

Find the time complexity of the above code in Big O notation: $O(2^n)$

Explain:

- Each element in the array has two choices: to be included in the subset or not. This results in a total of 2^n subsets for an array of length n.
- The `generatePowerSet` method is called recursively to explore each subset possibility, resulting in 2^n calls.