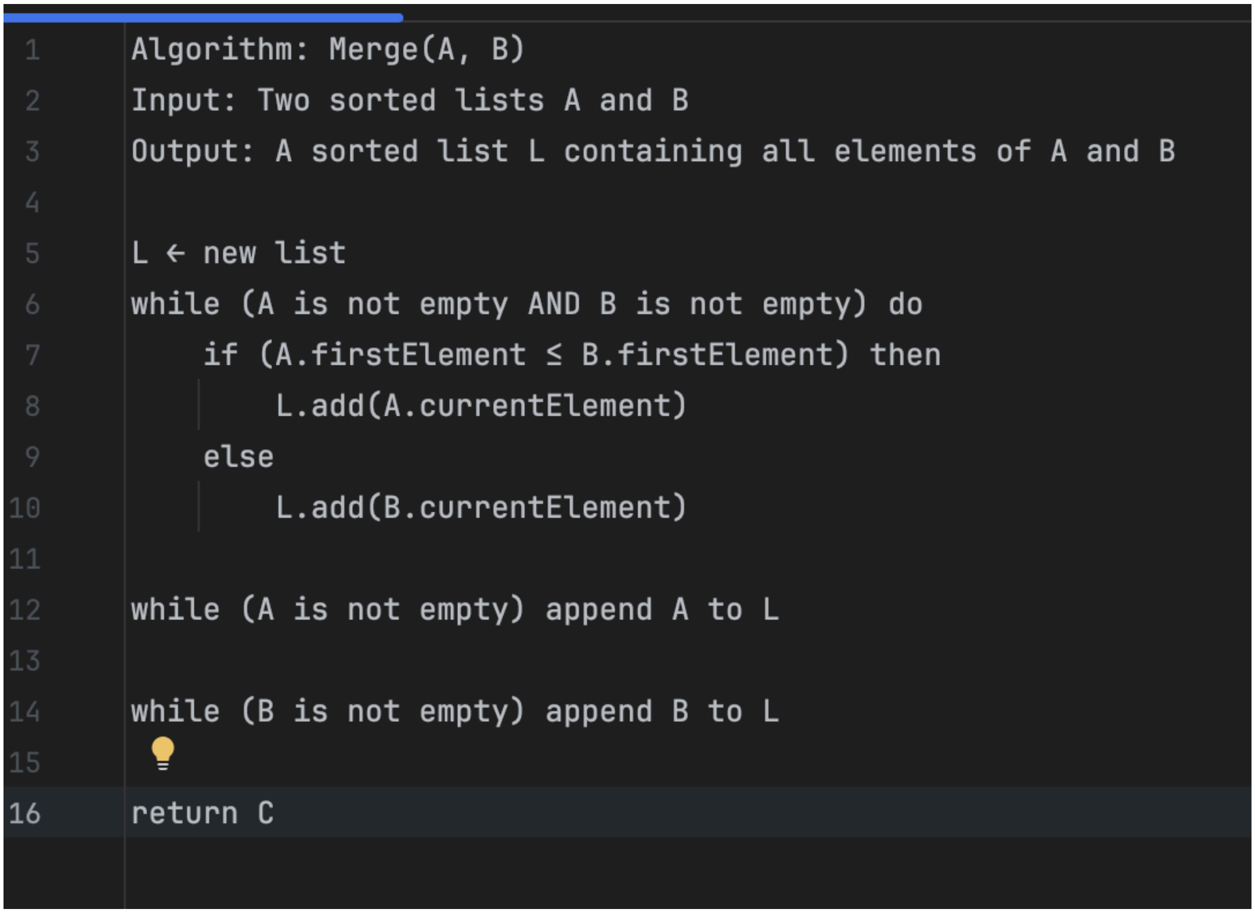
CS435DE - Lab 2

Saurab Ghimire

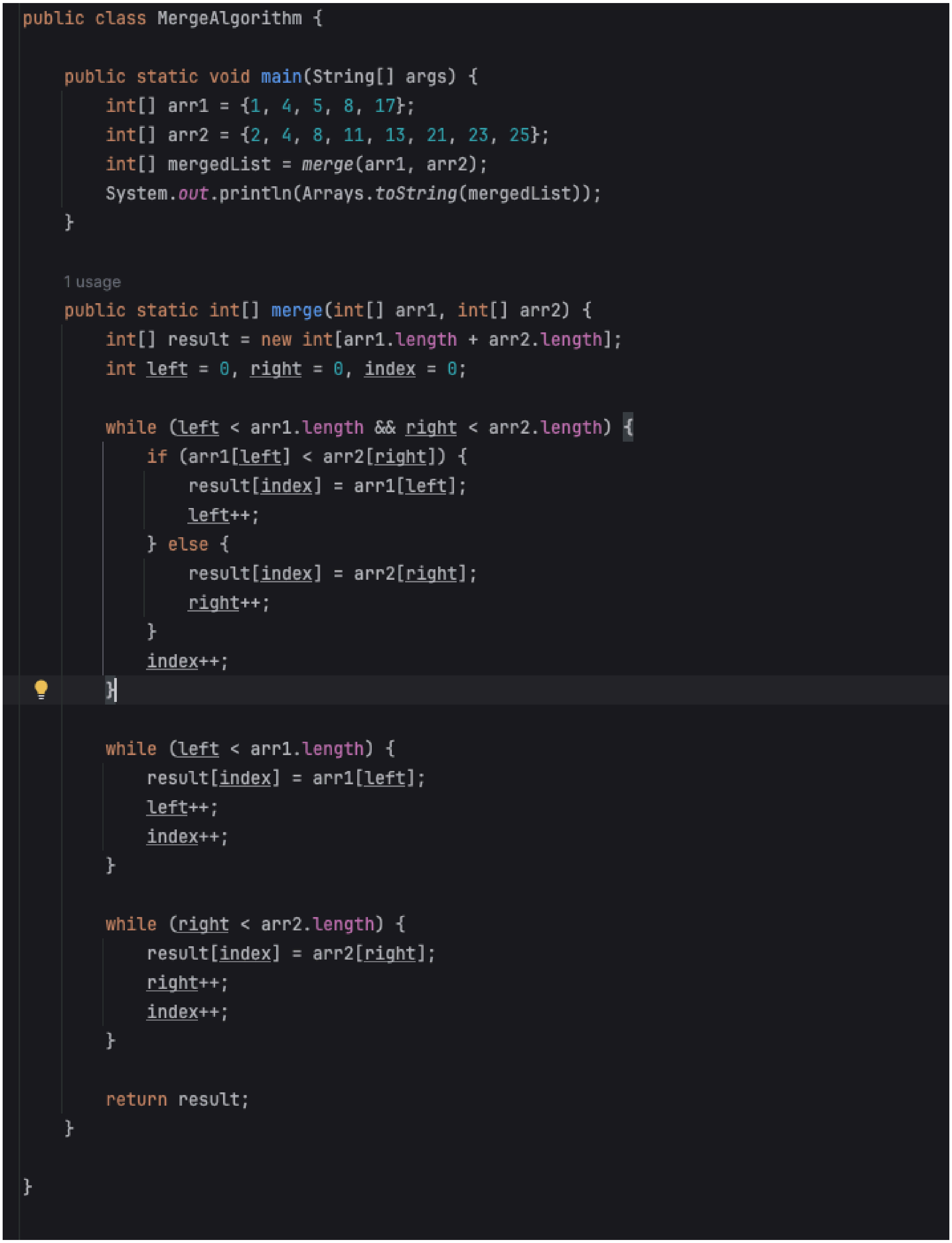
Problem 1: Solution
The asymptotic running time of the given procedure is O(n^2). The first for loop is O(n). The second for loop has a inner for loop so it is O(n.n). The O notation asymptotic running time of the procedure therefore is O(n^2).

Problem 2: Solution
The pseudo code to merge two sorted arrays together is as follows:



```
Algorithm: Merge(A, B)
Input: Two sorted lists A and B
Output: A sorted list L containing all elements of A and B

L ← new list
while (A is not empty AND B is not empty) do
    if (A.firstElement ≤ B.firstElement) then
        L.add(A.currentElement)
    else
        L.add(B.currentElement)

while (A is not empty) append A to L

while (B is not empty) append B to L

return C
```

The asymptotic running time is O(n+m).
Implemented code:



```java
public class MergeAlgorithm {

    public static void main(String[] args) {
        int[] arr1 = {1, 4, 5, 8, 17};
        int[] arr2 = {2, 4, 8, 11, 13, 21, 23, 25};
        int[] mergedList = merge(arr1, arr2);
        System.out.println(Arrays.toString(mergedList));
    }

    1 usage
    public static int[] merge(int[] arr1, int[] arr2) {
        int[] result = new int[arr1.length + arr2.length];
        int left = 0, right = 0, index = 0;

        while (left < arr1.length && right < arr2.length) {
            if (arr1[left] < arr2[right]) {
                result[index] = arr1[left];
                left++;
            } else {
                result[index] = arr2[right];
                right++;
            }
            index++;
        }

        while (left < arr1.length) {
            result[index] = arr1[left];
            left++;
            index++;
        }

        while (right < arr2.length) {
            result[index] = arr2[right];
            right++;
            index++;
        }

        return result;
    }

}
```

Problem 3: Solution

(3)

(A) $1 + 4n^2$ is $O(n^2)$

Let 'c' be a constant, such that

$$1+4n^2 \le c\cdot n^2 \quad \forall_n \ge 0$$

Dividing by n^2,

$$\frac{1}{n^2} + 4 \le c$$

Here, for $n \to \infty$, $\frac{1}{n^2} \simeq 0$,

For 'n' larger, c can be equal to 5.

Hence $1+4n^2 \rightarrow O(n^2)$

(B) $n^2 - 2n$ is not $O(n)$

If n^2-2n is $O(n)$, there must be 'c' such that

$$n^2 - 2n \le c\cdot n \quad \forall_n \ge n_0$$

Dividing by n, we get

$$n - 1 \le c$$

Here, 'c' is a constant, but as 'n' can approaches ∞,

Thus $n^2 - n$ is not $O(n)$

(C) $\log(n)$ is $o(n)$

As n goes to infinity

$$\lim_{n\to\infty} \frac{\log(n)}{n} = 0$$

Using $\displaystyle\lim_{n\to\infty} \frac{\log(n)}{n} = \lim_{n\to\infty} \frac{1/n}{1} = \lim_{n\to\infty} \frac{1}{n} = 0$

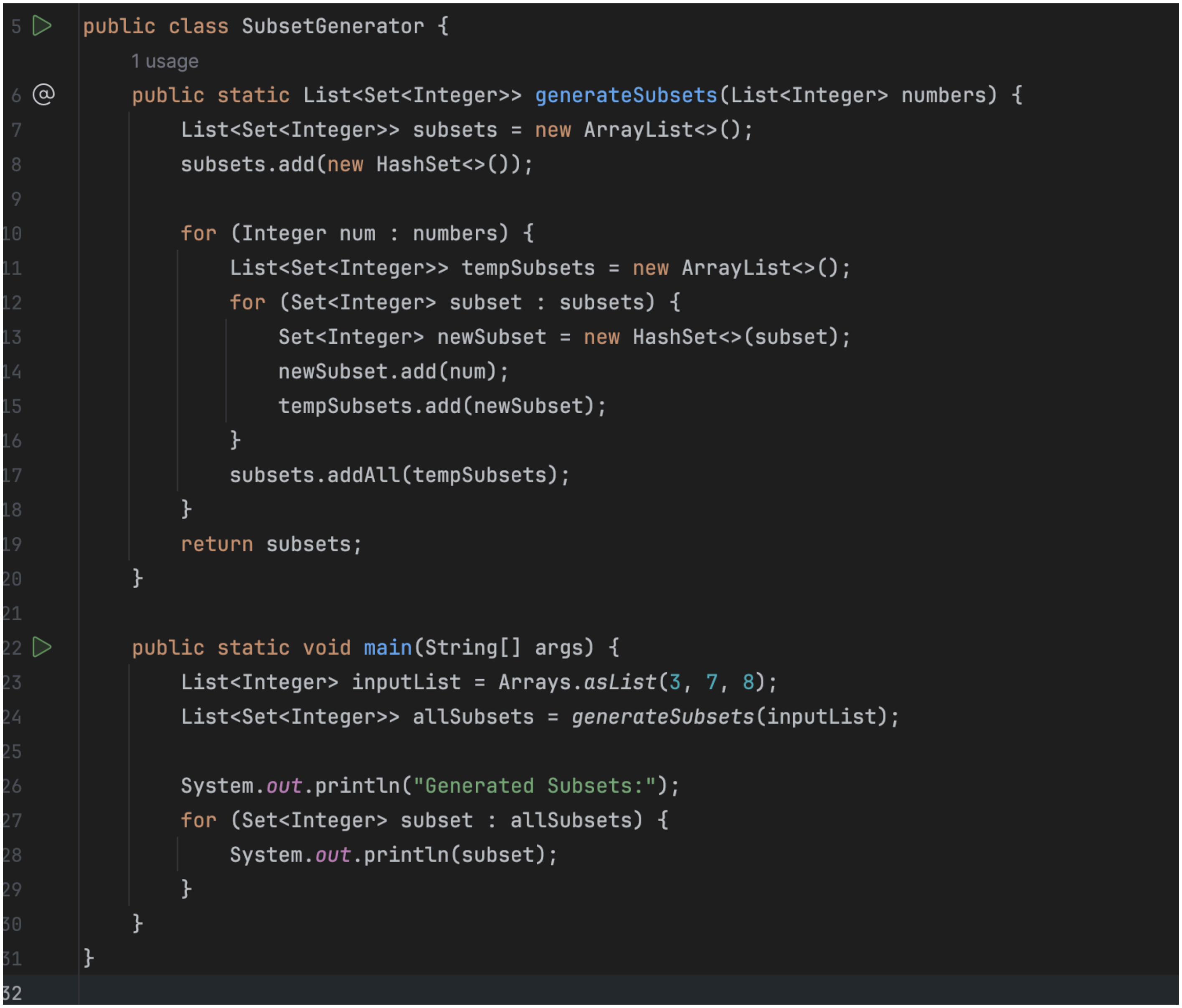
Thus, $\log(n)$ is $o(n)$.

(D) n is not $o(n)$

For this $\Rightarrow$ $\lim_{n \to \infty} \frac{n}{n} = 1$

Here, n is not $o(n)$ because the limit is not 0.

Problem 4: Solution



```java
public class SubsetGenerator {
    public static List<Set<Integer>> generateSubsets(List<Integer> numbers) {
        List<Set<Integer>> subsets = new ArrayList<>();
        subsets.add(new HashSet<>());

        for (Integer num : numbers) {
            List<Set<Integer>> tempSubsets = new ArrayList<>();
            for (Set<Integer> subset : subsets) {
                Set<Integer> newSubset = new HashSet<>(subset);
                newSubset.add(num);
                tempSubsets.add(newSubset);
            }
            subsets.addAll(tempSubsets);
        }
        return subsets;
    }

    public static void main(String[] args) {
        List<Integer> inputList = Arrays.asList(3, 7, 8);
        List<Set<Integer>> allSubsets = generateSubsets(inputList);

        System.out.println("Generated Subsets:");
        for (Set<Integer> subset : allSubsets) {
            System.out.println(subset);
        }
    }
}
```