Lab assignment solution

Problem 1:

The running time is the sum of the running time of the components. For the first for loop the running time is $\Theta(n)$ and for the second loop, it's the product of the upper loop and the inner loop, which is $\Theta(n^2)$. So the asymptotic running time is $\Theta(n^2)$.

Problem 2:

A.

Algorithm merge(A, B)
Input: sorted array A of n integers, sorted array B of m integers
Output: sorted array formed from A and B
C is empty array of length n+m;
$i \leftarrow 0, j \leftarrow 0, k \leftarrow 0$
While($i < n$ && $j < m$)
If($A[i] < B[j]$)
$C[k] \leftarrow A[i]$;
i++;
Else
$C[k] \leftarrow B[j]$;
j++;
k++;
while($i < n$)
$C[k] \leftarrow A[i]$
i++; k++;
While($j < m$)
$C[k] \leftarrow B[j]$
j++; k++;
return C;

B.

Assuming n is no greater than m, then the first while loop will run n times. Since the procedure inside that while loop has constant runtime, then the asymptotic runtime is big tetha of n. The second while loop will not run because at this point I is equal to n. The last while loop will run (m-n) times. Totally the asymptotic runtime will be big theta of n+(m-n) = m. which is big theta of m, when n<=m.

C. Java code implementation of the above pseudocode

```java
public class MergeSort {

    public static int[] merge(int[] arr1, int[] arr2) {
        int n = arr1.length;
        int m = arr2.length;
        int[] combined = new int[n+m];
        int i=0, j=0, k=0;
        while(i<n && j<m) {
            if(arr1[i]<arr2[j]) {
                combined[k] = arr1[i];
                i++;
            }
            else {
                combined[k] = arr2[j];
                j++;
            }
            k++;
        }
        while(i<n) {
            combined[k] = arr1[i];
            k++; i++;
        }
        while(j<m) {
            combined[k] = arr2[j];
            k++; j++;
        }
        return combined;
    }

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

Problem 3:

We can count the self-call to determine the asymptotic running time of the algorithm. $recursiveFactorial\ (n)$ calls $recursiveFactorial(n-1)$, and $recursiveFactorial(n-1)$ calls $recursiveFactorial(n-2)$, and so on until we reach to the base case $recursiveFactorial(1)$ calls $recursiveFactorial(0)$ which is 1. So, in total the factorial function was called n times. Hence the running time is $\Theta(n)$.

Problem 4:

```java
package problem04;

import java.util.ArrayList;
import java.util.List;

public class PowersetGenerator {

    static List<List<Integer>> powerSet(List<Integer> set){
        List<Integer> setcopy = new ArrayList<>(set);
        List<List<Integer>> powerset = new ArrayList<List<Integer>>();
        List<Integer> emptyset = new ArrayList<>();
        powerset.add(emptyset);
        while(!setcopy.isEmpty()) {
            Integer f = setcopy.remove(0);
            List<List<Integer>> temp = new ArrayList<>(powerset);
            for(List<Integer> subset: temp) {
                List<Integer> subsetcopy = new ArrayList<>(subset);
                subsetcopy.add(f);
                powerset.add(subsetcopy);
            }
        }
        return powerset;
    }

    public static void main(String[] args) {
        List<Integer> set = new ArrayList<>();
        set.add(2); set.add(5); set.add(7);
        List<List<Integer>> powerset = powerSet(set);

        System.out.println("For set: "+set);
        System.out.print("The subsets are: ");
        for(List<Integer> subset: powerset)
            System.out.print(subset+", ");
    }
}
```

Problem 5:

Using the master theorem, for $a = 1$, $b = 2$, $c = 1$, $k = 1$ and since $a < b^k$, the asymptotic runtime of the algorithm is $\Theta(n)$