1. Determine the asymptotic running time of the following procedure (an exact computation of number of basic operations is not necessary):

```
int[] arrays(int n) {
  int[] arr = new int[n];
  for(int i = 0; i < n; ++i){
   arr[i] = 1;
  }
  for(int i = 0; i < n; ++i) {
    for(int j = i; j < n; ++j){
      arr[i] += arr[j] + i + j;
    }
  }
  return arr;
}
```

Soln: Running time is O(n) + O(n^2). By an exercise, this is O(n^2).

2. Consider the following problem: As input you are given two sorted arrays of integers. Your objective is to design an algorithm that would merge the two arrays together to form a new sorted array that contains all the integers contained in the two arrays. For example, on input
[1, 4, 5, 8, 17], [2, 4, 8, 11, 13, 21, 23, 25]
the algorithm would output the following array:
[1,2,4,4,5,8,8, 11, 13, 17, 21, 23, 25]
For this problem, do the following:

A. Design an algorithm Merge to solve this problem and write your algorithm description using the pseudo-code syntax discussed in class.

Soln: Pseudo-code design description

```
Algorithm Merge(arr1, arr2):
  Input: Two sorted arrays arr1 and arr2
  Output: A merged sorted array containing all elements of arr1 and arr2

  If length of arr1 is equal to 0 then
    return arr2
  End If

  If length of arr2 is equal to 0 then
    return arr1
  End If

  n1 ← length(arr1)
  n2 ← length(arr2)
  result ← array of size (n1 + n2)
  i ← 0   // Pointer for arr1
  j ← 0   // Pointer for arr2
  k ← 0   // Pointer for result
```

```
        // Traverse both arrays until one is exhausted
        while i < n1 and j < n2 do
            if arr1[i] ≤ arr2[j] then
                result[k] ← arr1[i]
                i ← i + 1
            else
                result[k] ← arr2[j]
                j ← j + 1
            end if
            k ← k + 1
        end while

        // Store remaining elements of arr1
        while i < n1 do
            result[k] ← arr1[i]
            i ← i + 1
            k ← k + 1
        end while

        // Store remaining elements of arr2
        while j < n2 do
            result[k] ← arr2[j]
            j ← j + 1
            k ← k + 1
        end while

        return result
    end Algorithm
```

B. Examining your pseudo-code, determine the asymptotic running time of this merge algorithm

Soln: The overall running time of the algorithm is dominated by the main while loop and the step that handles the remaining elements. Therefore, the asymptotic running time of the merge algorithm is O(n1+n2).

C. Implement your pseudo-code as a Java method merge having the following signature:

int[] merge(int[] arr1, int[] arr2)
Be sure to test your method in a main method to be sure it really works!

Soln:

```java
public class MergeSortedArrays {

    private static int[] merge(int[] arr1, int[] arr2){
        if(arr1.length == 0 ){
            return arr2;
        }
        if(arr2.length == 0 ){
            return arr1;
        }

        int n1 = arr1.length;
        int n2 = arr2.length;
        int[] result = new int[n1 + n2];
        int i = 0, j = 0, k = 0; // Initialize the pointers for arr1, arr2 and result respectively.

        // Traversing both arrays
        while (i < n1 && j < n2){
            if(arr1[i] <= arr2[j]){
                result[k++] = arr1[i++];
            }else{
                result[k++] = arr2[j++];
            }
        }
        // Store remaining elements of arr1
        while(i < n1){
            result[k++] = arr1[i++];
        }
        // Store remaining elements of arr2
        while(j < n2){
            result[k++] = arr2[j++];
        }
        return result;
    }

    public static void main(String[] args) {
        int[] arr1 = {1, 4, 5, 8, 17};
        int[] arr2 = {2, 4, 8, 11, 13, 21, 23, 25};
        int[] arr3 = {};
        int[] arr4 = {};

        int[] mergedArray1 = merge(arr1, arr2);
        int[] mergedArray2 = merge(arr1, arr3);
        int[] mergedArray3 = merge(arr3, arr4);

        System.out.println("Merged array1: " + Arrays.toString(mergedArray1));
        System.out.println("Merged array2: " + Arrays.toString(mergedArray2));
        System.out.println("Merged array3: " + Arrays.toString(mergedArray3));

    }
}
```

3. **Big-oh and Little-oh.** Use the definitions of $O(f(n))$ and limit facts about $o(f(n))$ given in class to decide whether each of the following is true or false, and in each case, prove your answer.

By definition => $O(f(n))$: $g(n) = O(f(n))$ if there exist constants $c > 0$ and n_0 such that for all $n \geq n_0$, $0 \leq g(n) \leq c \cdot f(n)$. and $o(f(n))$: $g(n) = o(f(n))$ if for any constant $c > 0$, there exists an n_0 such that for all $n \geq n_0$, $0 \leq g(n) < c \cdot f(n)$.

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

To prove: $1+4n^2 = O(n^2)$

By the definition of $O(f(n))$, we need to find constants $c > 0$ and n_0 such that $1 + 4n^2 \leq c \cdot n^2$ for all $n \geq n_0$.

For $n \geq 1$, $4n^2 \geq 4$. Thus, $1 + 4n^2 \leq 4n^2 + 1 \leq 4n^2 + n^2 = 5n^2$.

So, $1 + 4n^2 \leq 5n^2$. Let $c = 5$ and $n_0 = 1$.

Hence, $1 + 4n^2 \leq 5n^2$ for all $n \geq 1$, proving that $1 + 4n^2 = O(n^2)$.

Therefore, **true**.

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

To prove: $n^2 - 2n$ is not $O(n)$

Suppose $n^2 - 2n = O(n)$.

By definition, there exist constants $c > 0$ and n_0 such that $n^2 - 2n \leq c \cdot n$ for all $n \geq n_0$.

Dividing both sides by n (for $n > 0$), we get $n - 2 \leq c$.

As n grows larger, the left-hand side $n - 2$ grows without bound, while c is a constant.

This is a contradiction because no constant c can be bound $n - 2$ as $n \to \infty$.

Therefore, **true** because $n^2 - 2n$ is not $O(n)$.

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

To prove: $\log(n) = o(n)$
By definition, $\log(n) = o(n)$ if for any constant $c > 0$, there exists an n_0 such that $\log(n) < c \cdot n$ for all $n \geq n_0$.

Consider the ratio $\log(n) / n$.

$\lim(n\to\infty) \log(n) / n = 0$ because the logarithm function grows much slower than the linear function.

For any $c > 0$, we can find an n_0 such that for all $n \geq n_0$, $\log(n) / n < c$.

Hence, $\log(n) < c \cdot n$ for sufficiently large n, proving $\log(n) = o(n)$.

Therefore, **true**.

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

To prove: $n \neq o(n)$
By definition, $n = o(n)$ would mean that for any constant $c > 0$, there exists an n_0 such that $n < c \cdot n$ for all $n \geq n_0$.

This simplifies to $1 < c$, which is not possible for all constants c.

In particular, if $c = 1$, then $n < 1 \cdot n$, which is a contradiction.

Therefore, n cannot be $o(n)$ because n grows exactly as n.

Hence, **true**. n is not o(n).

Power Set Algorithm. Given a set X, the power set of X, denoted P(X), is the set of all subsets of X. Below, you are given an algorithm for computing the power set of a given set. This algorithm is used in the brute-force solution to the SubsetSum Problem, discussed in the first lecture. Implement this algorithm in a Java method:
List powerSet(List X)
Use the following pseudo-code to guide development of your code

```
Algorithm: PowerSet(X)
Input: A list X of elements
Output: A list P consisting of all subsets of X – elements of P are Sets
P ← new list
S ← new Set //S is the empty set
P.add(S) //P is now the set { S }
T ← new Set
while (!X.isEmpty() ) do
    f ← X.removeFirst()
    for each x in P do
        T ← x U {f} // T is the set containing f & all elements of x
        P.add(T)
return P
```

```java
public class PowerSetAlgo {

    private static List<Set<Integer>> powerSet(List<Integer> X) {
        List<Set<Integer>> P = new ArrayList<>();
        Set<Integer> S = new HashSet<>(); // S is the empty set
        P.add(S); // P is now the set { S }

        while (!X.isEmpty()) {
            Integer f = X.remove(0); // Remove the first element from X
            List<Set<Integer>> newSubsets = new ArrayList<>();

            for (Set<Integer> x : P) {
                Set<Integer> T = new HashSet<>(x); // Create a new set T
                T.add(f); // T is the set containing f and all elements of x
                newSubsets.add(T); // Add T to the new subsets list
            }

            P.addAll(newSubsets); // Add all new subsets to P
        }

        return P;
    }
    public static void main(String[] args) {

        List<Integer> X = new ArrayList<>();
        X.add(100);
        X.add(225);
        X.add(30);
        List<Set<Integer>> powerSet = powerSet(X);
        System.out.println("Power Set: " + powerSet);
    }
}
```