Assignment 1:

1.
   a. $4n^3 + n$ is $\theta(n^3)$: True
   b. $\log n$ is $o(n)$: True
   c. 2^n is $\omega(n^2)$: True

2. $O(n^2)$

3.
   a.

```
Algorithm Merge(a, b, la, lb)
  Input: An already sorted array a and b and their respective lengths la and lb
  Output: Sorted array after merging both arrays a and b

    cursorA ← 0
    cursorB ← 0
    result ← [ ]
    pos ← 0

    while cursorA < la && cursorB < lb do
      if a[cursorA] < b[cursorB] then
        result[pos] ← a[cursorA]
        cursorA++
      else
        result[pos] ← b[cursorB]
        cursorB++
      pos++

    if cursorA = la then
      while pos < la+lb do
        result[pos] ← b[cursorB]
        cursorB++
        pos++
    if cursorB = lb then
      while pos < la+lb do
        result[pos] ← a[cursorA]
        cursorA++
        pos++
    return result
```

b. O(n)

c.

```
int[] merge(int[] a, int[] b){
    int cursorA = 0;
    int cursorB = 0;
    int[] result = new int[a.length+b.length];
    int pos = 0;

    while(cursorA<a.length && cursorB<b.length){
        if(a[cursorA] < b[cursorB]){
            result[pos] = a[cursorA];
            cursorA++;
        }else{
            result[pos] = b[cursorB];
            cursorB++;
        }
        pos++;
    }
    if(cursorA == a.length){
        while(pos < result.length){
            result[pos] = b[cursorB];
            cursorB++;
            pos++;
        }

    }
    if(cursorB == b.length){
        while(pos < result.length){
            result[pos] = a[cursorA];
            cursorA++;
            pos++;
        }
    }
    return result;
     }
```

4. $O(n)$
5.

```
Algorithm findZeroesLastIndex(arr, s, e, l)
   Input: sorted array containing 1 and 0, the start and the end index and
the length of the array
  Output: last index of the value 0
 if s > e return -1
 m <-- floor((s+e)/2)
 if isValidLastIndexOfZero(arr,m,l) then
     return m
 else if arr[m] = 0 then
    return findZeroesLastIndex(arr, m + 1, e,l)
 else
   return findZeroesLastIndex(arr, s, m - 1,l)

Algorithm isValidLastIndexOfZero(arr,index,l)
   Input: sorted array containing 1 and 0 and the index to check and the
length of the array
   Output: true or false based on whether the provided index is indeed
the last index
   if arr[index] = 0 && (index = l - 1 || arr[index + 1] = 1) then
        return true
   return false

Algorithm countOnesAndZeroes(arr, s, e)
      Input: sorted array containing 1 and 0, the start and the end index
      Output: number of 1's and 0's in the array
      index <- findZeroesLastIndex(arr, s, e + 1);
      return {zeroCounter: index + 1, oneCounter: e - index};
```

The above algorithm runs in Little-o(n) time because its running time is O(logn) , which has much slower growth rate than n. The algorithm is based on binary search.