# Lab 1 Solution

## Answer of 1

a. True
b. True
c. True

## Answer of 2

Answer: $O(n^2), \theta(n^2)$

## Answer of 3

Solutions of 3(A)

**Algorithm:** merge(int[] arr1, int[] arr2)

**Input:** arr1 and arr2 are two sorted integer arrays that can be empty or filled but not null.

**Output:** a sorted array merging arr1 and arr2

$if\ arr1.length = 0\ then$
    $return\ arr2$
$else\ if\ arr2.length = 0\ then$
    $return\ arr1$

$Size \leftarrow arr1.length + arr2.length$
$result \leftarrow new\ array\ of\ size\ (Size)$
$index \leftarrow 0$
$left \leftarrow 0$
$right \leftarrow 0$

$while\ left < arr1.length\ and\ right < arr2.length\ do$:
    $if\ arr1[left] \leq arr2[right]\ then$
        $result[index] \leftarrow arr1[left]$
        $left \leftarrow left + 1$
    $else\ do$
        $result[index] \leftarrow arr2[right]$
        $right \leftarrow right + 1$
    $index \leftarrow index + 1$

$while\ left < arr1.length\ do$
$\quad result[index] \leftarrow arr1[left]$
$\quad index \leftarrow index + 1$
$\quad left \leftarrow left + 1$

$while\ right < arr2.length\ do$
$\quad result[index] \leftarrow arr2[right]$
$\quad index \leftarrow index + 1$
$\quad right \leftarrow right + 1$

$return\ result$

Solutions of 3(B)

Asymptotic notation is:
$\theta(m+n),\ O(m+n)$ *where* $m$ *and* $n$ *is the size of the arrays* $arr1$ *and* $arr2$.

Solution of 3(C)

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

    int left = 0, right = 0;

    int[] result = new int[arr1.length + arr2.length];
    int index = 0;

    // append to result from arr1 and arr2 comparing the values
    // until one of them is empty
    while (left < arr1.length && right < arr2.length) {
        if(arr1[left] <= arr2[right]){
            result[index] = arr1[left];
            left++;
        } else {
            result[index] = arr2[right];
            right++;
        }

        index++;
    }

    // append remaining elements of arr1 if arr1 is not empty
    while (left < arr1.length){
        result[index] = arr1[left];
        left++;
```

```
        index++;
    }

    // append remaining elements of arr2 if arr2 is not empty
    while (right < arr2.length){
        result[index] = arr2[right];
        index++;
        right++;
    }

    return result;
}
```

## Answer of 4

n: 0, 1, 2, 3, 4...n
calls: 1, 1, 1, 2, 3....(n-1)

Total calls would be: n+1

Solution: $O(n), \theta(n)$

## Answer of 5

Solution:

```
void countOnesAndZeroes(int[] arr) {
    int left  = 0;
    int right = arr.length - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == 1) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    int numberOfZeroes = left;
    int numberOfOnes   = arr.length - left;

    System.out.println("Array: " + Arrays.toString(arr));
    System.out.println("Number of zeroes: " + numberOfZeroes);
    System.out.println("Number of ones: " + numberOfOnes);

}
```

The above code solves the problem in $O(logn)$ . The algorithm cuts down the search space in half every time reducing the need to search through the entire list. At each iteration, search space is halved, making sure the algorithm runs at most $logn$ times.

And we know that $logn\ is\ o(n)$, the above solution satisfies the requirement.