# Lab 1 Solution

## Answer of 1

1. True
2. True
3. True

## Answer of 2

Answer:

## 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

Solutions of 3(B)

Asymptotic notation is:

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:

## 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 . 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 times.

And we know that , the above solution satisfies the requirement.