Lab3

# Problem 1

1. Yes. With random arrange all element will have change to be sorted
2. The Best case is inputted array is sorted
3. Running time of the Best case is O(n). This is the time for checking inputted array is sorting
4. Worst-case running time is infinity
5. No. Assume input array is in reverse order so we need to do n \* (n - 1) / 2 time for inversions. And after the first try the array is sorted so we need to do n comparisons for inverse elements and n comparisons for checking array is sorted. The comparisons time is n + n < n \* (n - 1) / 2. So the number of comparisons is less then number of inversions.

# Problem 2

Algorithm sortZeroOneTwo(A)  
 Input: An array A contains list of elements have value 0 or 1 or 2  
 Output: An sorted array  
  
 if A = null || A.length <= 1 then  
 return A  
  
 finalSortedArr <- new int[A.length]  
  
 countZero <- 0  
 countOne <- 0  
 countTwo <- 0  
  
 for i <- 0 to A.length - 1 do  
 if A[i] = 0 then  
 countZero <- countZero + 1  
 if A[i] = 1 then  
 countOne <- countOne + 1  
 if A[i] = 2 then  
 countTwo <- countTwo + 1  
  
 for i <- 0 to countZero do  
 finalSortedArr[i] <- 0  
  
 for i <- countZero to countZero + countOne do  
 finalSortedArr[i] <- 1  
  
 for i <- countZero + countOne to A.length do  
 finalSortedArr[i] <- 2  
  
 return finalSortedArr

We have 4 for loop but it not nested so the running time is O(n)

# Problem 3

A

public void bubbleShort(int[] arr) {  
 for (int i = 0; i < arr.length; i++) {  
 boolean isSwap = false;  
 for (int j = 0; j < arr.length - 1; j++) {  
 if (arr[j] > arr[j + 1]) {  
 swap(j, j + 1);  
 isSwap = true;  
 }  
 }  
 if (!isSwap) {  
 break;  
 }  
 }  
}

I use a flag variable isSwap to check the input Array is sorted or not. With the sorted array isSwap never turn to be true then break the outer for loop.

The run time now is O(n) because: The outer for loop will run n time, the inner for loop run n – 1 time and the because the array is already sorted so isSwap never turn to true then the outer loop is broken. So the run time = n + n – 1 = 2n – 1 (O(n))

B

private void bubbleSort(){  
 int len = arr.length;  
 for(int i = 0; i < len; ++i) {  
 for(int j = 0; j < len - i - 1; ++j) {  
 if(arr[j]> arr[j+1]){  
 swap(j,j+1);  
 }  
 }  
 }  
}

After i time the outer loop run, the element from n – i – 1 to n – 1 will be in final sorted order so the inner loop only need run from 0 to n – i – 1.

The reduce time = 1 + 2 + 3 + … + n – 1 = n2 / 2 => the running time is reduced a haft but still O(n2)

C.

The result is

477 ms -> BubbleSort2

498 ms -> BubbleSort1

532 ms -> BubbleSort

This is not exactly what I expected.

I expect the running time of BubbleSort2 is a haft of BubbleSort.

The running time is show like above (BubbleSort1 & BubbleSort2 < BubbleSort) because the BubbleSort1 & BubbleSort2 are improving version of BubbleSort, we already reduce some redundancy calculating.