Methew Guda - A00381751
Nrimoni Chowdhury - A00371596

Bubble & Insertion Sort program written in JAVA

```
package com.priom;
import java.util.*;

public class BubbleInsertSort {
   public static int bubbCompCount = 0;
   public static int bubbSwapCount = 0;
   public static int bubbPassCount = 0;
   public static int insrtCompCount = 0;
   public static int insrtSwapCount = 0;

   public static void main(String[] args) {
//    Generate random list
      Random random = new Random();
      int r[] = new int[100];
      int r2[] = new int[100];
      for (int m = 0; m < r.length; m++) {
         r[m] = random.nextInt(500);
         r2[m] = r[m];
      }
//    Print BUBBLE sorted list
      bubbleSort(r);
      System.out.print("BUBBLE Sorted List: ");
      for (int p = 0; p < r.length; p++) {
         System.out.print (r[p] + " ");
      }
      System.out.println();
      System.out.printf("BUBBLE Sort: Comparison: %d, Swaps: %d, Passes: %d.",
            bubbCompCount, bubbSwapCount, bubbPassCount);
//    Print INSERTION sorted list
      insertSort(r2);
      System.out.println();
      System.out.println();
      System.out.print("INSERTION Sorted List: ");
```

```
        for (int q = 0; q < r.length; q++) {
            System.out.print (r[q] + " ");
        }
        System.out.println ();
        System.out.printf ("INSERTION Sort: Comparison: %d, Swaps: %d.",
                insrtCompCount, insrtSwapCount);
        System.out.println ();
    }
//    Iterative function for BUBBLE sort
    public static void bubbleSort(int a[]) {
        for (int i = 1; i < a.length; i++) {
            for (int j = 0; j < a.length-1; j++)  {
                bubbCompCount = (a.length * (a.length-1)) / 2;
                if (a[j] > a[j+1]) {
                    bubbSwapCount++;
                    bubbPassCount = a.length - 1;
                   int temp = a[j];
                   a[j] = a[j+1];
                   a[j+1] = temp;
                }
            }
        }
    } //end func
//    Iterative function for INSERTION sort
    public static void insertSort(int b[]) {
        for(int i = 0; i<b.length; i++) {
            int temp = b[i];
            int j;
            for(j = i-1; j >= 0 && temp < b[j]; j--)
            {
                b[j+1] = b[j];
                insrtSwapCount++;
                insrtCompCount++;
            }
            b[j+1] = temp;
            insrtCompCount++;
        }
    } //end func
} //end class
```

Bubble Sort Analysis:

List of 25:

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Comparison	300	300	300	300	300
Swap	145	157	144	143	169
Pass	24	24	24	24	24

Number of comparisons and passes remains constant all through our experiments.
Comparisons: 300
Passes: 24

Swaps kept changing in our experiment.
Maximum Swaps: 169
Minimum Swaps: 143
Average Swaps: 152

List of 50:

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Comparison	1225	1225	1225	1225	1225
Swap	685	574	700	540	571
Pass	49	49	49	49	49

Number of comparisons and passes remains constant all through our experiments.
Comparisons: 1225
Passes: 49

Swaps kept changing in our experiment.
Maximum Swaps: 700
Minimum Swaps: 540
Average Swaps: 614

List of 100:

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Comparison	4950	4950	4950	4950	4950
Swap	2468	2522	2376	2237	2544
Pass	99	99	99	99	99

Number of comparisons and passes remains constant all through our experiments.
Comparisons: 4950
Passes: 99

Swaps kept changing in our experiment.
Maximum Swaps: 2544
Minimum Swaps: 2237
Average Swaps: 2429

List of 500:

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Comparison	124750	124750	124750	124750	124750
Swap	62757	64949	61289	63684	62119
Pass	499	499	499	499	499

Number of comparisons and passes remains constant all through our experiments.
Comparisons: 124750
Passes: 499
Swaps kept changing in our experiment.
Maximum Swaps: 64949
Minimum Swaps: 61289
Average Swaps: 62960

List of 1000:

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Comparison	499500	499500	499500	499500	499500
Swap	245613	248815	250243	254901	245483
Pass	999	999	999	999	999

Number of comparisons and passes remains constant all through our experiments.
Comparisons: 499500
Passes: 999

Swaps kept changing in our experiment.
Maximum Swaps: 254901
Minimum Swaps: 245483
Average Swaps: 249011

Insertion Sort Analysis:

List of 25:

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Comparison	170	182	169	168	194
Swap	145	157	144	143	169

Comparison kept changing in our experiment.
Maximum Comparison : 194
Minimum Comparison : 168
Average Comparison : 177

Swaps kept changing in our experiment.
Maximum Swaps: 169
Minimum Swaps: 143
Average Swaps: 152

List of 50:

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Comparison	735	624	750	590	621
Swap	685	574	700	540	571

Comparison kept changing in our experiment.
Maximum Comparison : 750
Minimum Comparison : 590
Average Comparison : 664

Swaps kept changing in our experiment.
Maximum Swaps: 700
Minimum Swaps: 540
Average Swaps: 614

List of 100:

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Comparison	2568	2622	2476	2337	2644
Swap	2468	2522	2376	2237	2544

Comparison kept changing in our experiment.
Maximum Comparison : 2644
Minimum Comparison : 2337
Average Comparison : 2529

Swaps kept changing in our experiment.
Maximum Swaps: 2544
Minimum Swaps: 2237
Average Swaps: 2429

List of 500:

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Comparison	63257	65449	61789	64184	62619
Swap	62757	64949	61289	63684	62119

Comparison kept changing in our experiment.
Maximum Comparison : 65449
Minimum Comparison : 61789
Average Comparison : 63460

Swaps kept changing in our experiment.
Maximum Swaps: 64949
Minimum Swaps: 61289
Average Swaps: 62960

List of 1000:

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Comparison	246613	249815	251243	255901	246483
Swap	245613	248815	250243	254901	245483

Comparison kept changing in our experiment.
Maximum Comparison : 246483
Minimum Comparison : 246613
Average Comparison : 250011

Swaps kept changing in our experiment.
Maximum Swaps: 254901
Minimum Swaps: 245483
Average Swaps: 249011

According to our experiment, the swap count in both bubble and insertion sort is same so, is the time complexity of $O(n^2)$.

The bubble sort algorithm can be easily optimized by observing that the n^{th} pass finds the n^{th} largest element and puts it into its final place. So, the inner loop can avoid looking at the last n-1 items when running for the n^{th} time.

More generally, it can happen that more than one element is placed in their final position on a single pass. In particular, after every pass, all elements after the last swap are sorted, and do not need to be checked again. This allows us to skip over a lot of the elements, resulting in about a worst case 50% improvement in comparison count (though no improvement in swap counts), and adds very little complexity because the new code subsumes the "swapped" variable:

The optimized algorithm is written below:

```
function bubbleSort( A : list of sortable items )
    n = length(A)
    repeat
      newN = 1
      for i = 1 to n-1 inclusive do
        if A[i-1] > A[i] then
          swap(A[i-1], A[i])
          newn = i
        end if
      end for
      n = newN
    until n = 1
end function
```