**Problem1: Goofy Sort algorithm**

1. It may work, in two cases. But in general, that is not going to work.

Case 1: If the input array is already sorted

Case 2: Although it is unlikely, the randomly arranging algorithm may end up arranging them. For example, if the elements in the array are 2, it is highly likely that the random procedure will sort the elements.

Other than, the above two unlikely cases, the algorithm is going to perform terribly bad.

1. The best case of Goofy Sort is when the input array is already sorted.
2. Infinity unless the input array is ordered in the beginning which is unlikely.

1. Goofy Sort is “Inversion bound” because, there are at least as many comparisons as the number of inversions required on the input array and there will be several swaps in the permutation process to rearrange the elements randomly in the second step.

**Problem2: (a)**

**private** **void** bubbleSort(){

**int** len = arr.length;

**for**(**int** i = 0; i < len; ++i) {

**boolean** sorted = **false**;

**for**(**int** j = 0; j < len-1; ++j) {

**if**(arr[j]> arr[j+1]){

swap(j,j+1);

sorted = **true**;

}

}

**if**(!sorted){

**break**;

}

}

}

**Problem2: (b)**

**private** **void** bubbleSort(){

**int** len = arr.length;

**for**(**int** i = 0; i < len; ++i) {

**for**(**int** j = 0; j < arr[arr.length - i - 1]; ++j) {

**if**(arr[j]> arr[j+1]){

swap(j,j+1);

}

}

}

}

**Problem2: (c)**

The running time of sorting are as follows

56 ms -> InsertionSort

156 ms -> SelectionSort

464 ms -> BubbleSort1

466 ms -> BubbleSort2

526 ms -> BubbleSort

The results are as I expected except that theBubbleSort1 has not improved the performance that much from the original BubbleSort. This is Due to the randomization introduced during the generation of test data, which is likely to produce sorted array hence, both methods give approximately similar results. And the BubbleSort2 which minimizes the operations by half compared to the BubbleSort is much lower relatively although both runs at 𝑂(𝑛!). Furthermore, BubbleSort performs worse than both insertion sort and selections sort. This is since the unoptimized implementation of bubble sort runs 𝑂(𝑛2) times even if the array is sorted.

Problem3: Counting O’s and 1’s

**public** **static** **int**[] countZerosAndOnes(**int**[] arr) {

**if** (arr[arr.length - 1] == 0)

**return** **new** **int**[] { arr.length, 0 };

**if** (arr[0] == 1)

**return** **new** **int**[] { 0, arr.length };

**int** lower = 0;

**int** higher = arr.length - 1;

**int** countO = 0;

**while** (lower < higher) {

**int** mid = (lower + higher) / 2;

**if** (arr[mid] == 0) {

countO += (mid - lower) + 1;

**if** (arr[mid + 1] == 1)

**break**;

lower = mid + 1;

} **else** {

**if** (arr[mid - 1] == 0) {

countO = mid;

**break**;

}

higher = mid - 1;

}

}

**return** **new** **int**[] { countO, arr.length - countO };

}