Yuliang Jin 986381

Problem 1

A. Will Goofy’s sorting procedure work at all? Explain This sorting algorithm will not work.

The probability that the array is sorted after the O(n) random arrange is pretty low, 1/(n^2).

B. What is a best case for GoofySort?

The best case is the initial array is already sorted, and the check step is still O(n) running time.

C. What is the running time in the best case?

O(n).

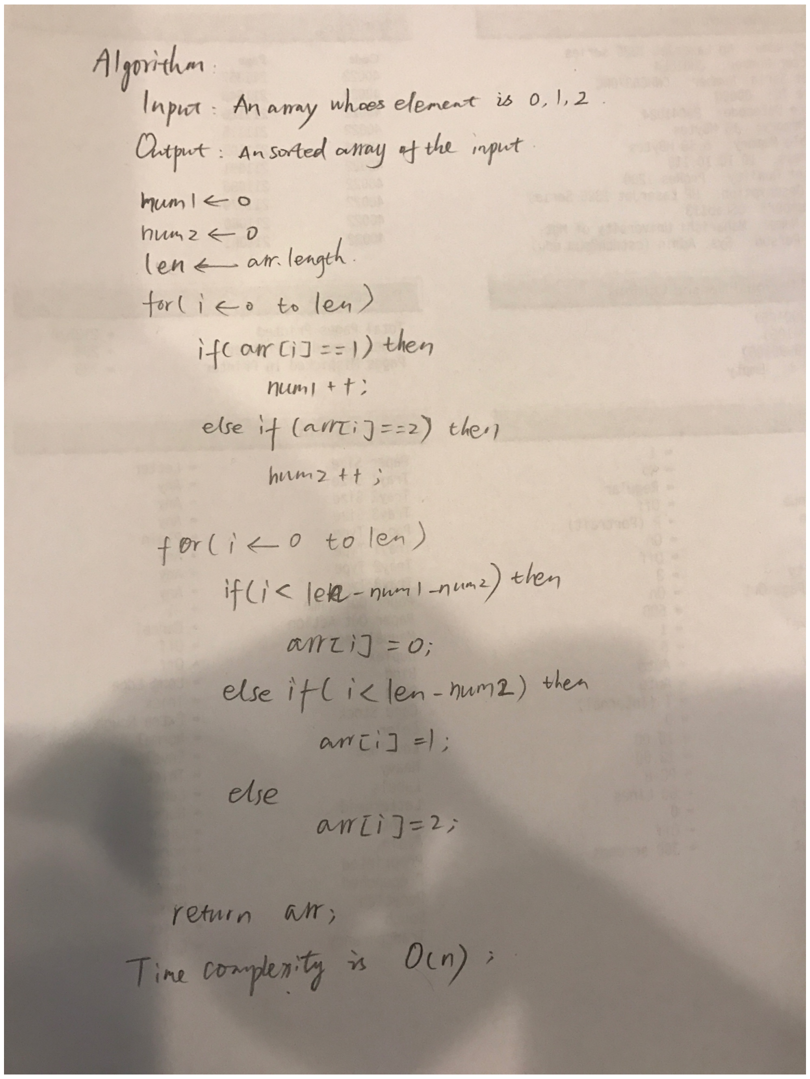
D. What is the worst-case running time?

The worst-case is O(n^2 * (n + n)) = O(n^3).

E. Is the algorithm inversion-bound?

If the random rearrange process includes some inversions, then the number of inversions might be bigger than that of the comparisons. In this case it's not inversion-bound.

Problem 2



```
Algorithm:
  Input: An array whoes element is 0, 1, 2.
  Output: An sorted array of the input.
  num1 ← 0
  num2 ← 0
  len ← arr.length.
  for( i ← 0 to len)
      if( arr[i] == 1) then
          num1 ++;
      else if (arr[i]==2) then
          num2 ++;

  for( i ← 0 to len)
      if(i < len - num1 - num2) then
          arr[i] = 0;
      else if( i < len - num2) then
          arr[i] =1;
      else
          arr[i]=2;

  return arr;
Time complexity is O(n);
```

Problem 3

a

Improve the BubbleSort implementation so that when the input array become sorted after some runs of outer for loop, the algorithm will stop. Call your new Java file BubbleSort1.java.

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

b

Recall that in BubbleSort, at the end of the first pass through the outer loop, the largest element of the array is in its final sorted position. After the next pass, the next largest element is in its final sorted position. After the ith pass (i=0,1,2,...), the largest, second largest,..., i+1st largest elements are in their final sorted position. Use this observation to cut the running time of BubbleSort in half. Implement your solution in code, and prove that you have improved the running time in this way. Call your new Java file, which contains the improvements from this problem and the previous problem, BubbleSort2.java.

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

583 ms -> BubbleSort2 941 ms -> BubbleSort

c

In this lab folder, I have given you an environment for testing sorting routines. Insert into this environment the original BubbleSort file along with your new BubbleSort1 and BubbleSort2 classes, and run the SortTester class. What are the results? Are the

results what you expected? Explain why the running times turned out the way they did.
919 ms -> BubbleSort

736 ms -> BubbleSort1

644 ms -> BubbleSort2