\

|CS340 Homework 2 25 points Name_____

Fill in answers and submit as a PDF file.

There is an array, A, of numbers (n of them) to be sorted. All numbers are distinct (no number is repeated) and fall in a range p..q. The smallest and largest possible numbers are known and are input as part of the algorithm. For example, all numbers might be between 300 and 400.

Here is the sorting algorithm:

```
public static void mySort(int[] A, int p ,int q){
   int[] scratch = new int[q-p+1];
   for (int i = 0; i < A.length; i++) {
      scratch[A[i]-p] = A[i];
   }
   int counter = 0;
   for (int i = 0; i < scratch.length; i++) {
       if (scratch[i] > 0){
          A[counter] = scratch[i];
          counter++;
      }
   }
```

a. (2 points) The algorithm consists of 2 for loops. How many times does each loop execute if the numbers being sorted are A = {1, 2, 3, 4, 5}.

First loop A.length = 5 Second loop (q - p + 1) = (5 - 1 + 1) = 5

b. (2 points) How many times does each loop execute if the numbers being sorted are A = {0, 400, 800, 900, 999}.

First loop 5 Second loop 999 - 0 + 1 = 1000

c. (5 points) What is the time complexity of this algorithm?

FL: O(n) SL: O(q - p + 1) → O(n + (q - p + 1))

d. (2 points) When is this algorithm faster than MergeSort?

This algorithm can be faster than merge sort when the range of values q-p is not significantly larger than the size of the array n, and when n is fairly small. Because merge sort has a time complexity of O(n log n), for sufficiently small n, an O(n) algorithm can be faster.

e. (2 points) Give 2 advantages to this algorithm relative to either MergeSort or InsertionSort.

- The algorithm is in-place; it doesn't require additional space proportional to n during the sorting process
- It has linear time complexity when the range q-p is close to n, which can make it faster for small datasets with a small range of values

f. (2 points) Give 2 drawbacks to this algorithm relative to either MergeSort or InsertionSort.

- The algorithm depends heavily on the range of input values q-p and can be highly inefficient for large ranges.
- The algorithm cannot handle duplicate values, while both MergeSort and InsertionSort handle duplicates well.

g. (2 points) What is the loop invariant of the first loop?

The loop invariant for the first loop is at the start of each iteration, the array contains all the elements seen so far, placed at an index corresponding to their value minus p

h. (3 points) Prove the correctness of the first loop at initialization, maintenance and termination.

Initialization: Before the first iteration, no element has been seen so the condition is trivially satisfied

Maintenance: During each iteration, we place the current element from array A into the correct position in the scratch array, maintaining the invariant

Termination: At the end of the loop, the scratch array contains all elements from A, each placed at an index equal to its value minus p, proving loop works.

i. (2 points) What is the loop invariant of the second loop?

The loop invariant for the second loop is that, at the start of each iteration, the array A contains all non zero values from scratch seen so far in sorted order.

j. (3 points) Prove the correctness of the second loop at initialization, maintenance and termination.

Initialization: Before the first iteration, no element has been seen, so the condition is trivially satisfied

Maintenance: During each iteration, if a non-zero value is found in the scratch array, it is placed in the next available position in the array A, maintaining the invariant that A contains all non-zero values seen so far in sorted order.

Termination: At the end of the loop, all non-zero values have been placed in A in sorted order, demonstrating that the loop operates correctly.