CS 101A Fall 2025 - Quiz 3
November, 11, 2025 - 20 Minutes **Name:** **Student ID:**

---

1. (15 points) True or False

Determine whether the following statements are true or false.

(a) (3') Merge sort is not an in place sorting since it requires $\Theta(\log n)$ space. ◯ True ◯ False

(b) (3') When an array contains distinct elements and is arranged in descending order, the time complexity of quick sort (always choose the last element as pivot) is $\Theta(n^2)$. ◯ True ◯ False

(c) (3') There exists an comparison-base sort algorithm that needs $O(1)$ extra space and takes $o(n \log n)$ time. ◯ True ◯ False

(d) (3') For an array $\{a_n\}$ with distinct elements, for fixed i, j, if $\forall a_k \neq a_i$, $a_k \neq a_j$, $(a_k - a_i)(a_k - a_j) > 0$, then a_i and a_j will be compared in any case when using randomized quick-sort to make $\{a_n\}$ sorted. ◯ True ◯ False

(e) (3') When performing flagged bubble sort on a array of n elements, the minimum number of comparisons is $n - 1$. ◯ True ◯ False

2. (15 points) Fill in the blanks

(a) (3') If we use randomized quick-sort (i.e., randomly choosing pivots) to sort the array $[3, 4, 6, 2, 1, 5, 8, 0]$, the probability that 2 and 5 are compared is _____.

(b) (3') The worst case runtime of insertion sort is Θ (_____), and the best case runtime is Θ (_____). (The array has n entries.)

(c) (3') In average case, the space complexity of quick sort on n elements is Θ(_____) .

(d) (3') The following is the process of sorting an array $\{7, 8, 5, 2, 4, 6, 3\}$ by a sorting algorithm $\{7, 8, 5, 2, 4, 6, 3\} \to \{7, 5, 2, 4, 6, 3, 8\} \to \{5, 2, 4, 6, 3, 7, 8\} \to \{2, 4, 5, 3, 6, 7, 8\} \to \{2, 4, 3, 5, 6, 7, 8\} \to \{2, 3, 4, 5, 6, 7, 8\} \to \{2, 3, 4, 5, 6, 7, 8\}$

Which sorting algorithm is it? _____.

(e) (3') The array {8,9,10,4,5,6,20,1,2} is the result of _____ sort after two passer of the outer loop. (fill in the blank with "bubble" or "insertion")

3. (20 points) Alternating bubble sort

```
template <typename Type>
void bubble(Type* const array, int n) {
        int lower = 0;
        int upper = n - 1;
        while (true) {
                int new_upper = lower;
                for (int i = lower; i < upper; ++i) {
                        if (array[i] > array[i + 1]) {
                                Type tmp = array[i];
                                array[i] = array[i + 1];
                                array[i + 1] = tmp;
                                new_upper = i;}
                }
```

```
        upper = new_upper;
        if (lower == upper) { break; }
        int new_lower = upper;
        for (int i = upper; i > lower; --i) {
                if (array[i - 1] > array[i]) {
                        Type tmp = array[i];
                        array[i] = array[i - 1];
                        array[i - 1] = tmp;
                        new_lower = i;}
        }
        lower = new_lower;
        if (lower == upper) { break; }
}
```

Suppose we apply this algorithm to an array with n entries.

(a) (3') Compared with the standard one-way bubble sort, the main optimization of the given code is: _____

A. Using binary search to locate swap positions

B. Using a bidirectional pass to push both the maximum and minimum toward the boundaries, shrinking the unsorted range

C. Using a faster swap trick to reduce assignments

(b) (3') In the worst case (array in reverse order), the time complexity of this algorithm is Θ (_____).

(c) (6') After the first for loop (from lower to upper) in the first iteration of the while loop completes, the statement "upper = new_upper" updates the right boundary of the unsorted range to the last swap position i, skipping the already sorted suffix. Thus, the unsorted range of the array shrinks to [_____, _____] (fill in the name of the variable). If no swap occurs in this pass, then *new_upper* will be equal to _____.

(d) (3') For this code to sort a custom **Type** correctly, Type must define the operator $>$. Additionally, swapping via a temporary requires that the type supports copying or moving. ◯ True ◯ False

(e) (5') Sort the array {3, 7, 4, 2, 6, 5, 1, 0} using the algorithm above. Write down the array after each **for** loop.