CS 101 Fall 2023 - Quiz 3

10/30/2023 - 20 Minutes **Name:** **ID number:**

---

1. (1 points) Notes of discussion

I promise that I will complete this QUIZ independently and will not use any electronic products or paper-based materials during the QUIZ, nor will I communicate with other students during this QUIZ. I have read and understood the notes.

√ True ◯ False

2. (5 points) True or False

Determine whether the following statements are true or false.

(a) (1') The time complexity of Merge Sort is better than Flagged Bubble Sort in any case.

◯ True √ False

(b) (1') Store n integers in a singly linked list in ascending order. If we use binary search to find a number in it, then the average time complexity is $O(\log(n))$.

◯ True √ False

(c) (1') Use Bubble Sort on an array with length 7, the number of swaps required in the worst case is 21.

√ True ◯ False

(d) (1') For a random ordering array with n different elements, we would expect there are approximately $\Theta(n^2)$ pairs of inversions.

√ True ◯ False

(e) (1') Suppose we are performing merge-sort on an array with distinct elements. At a certain step, we need to merge two sorted subarrays (a_1, a_2, a_3) and (b_1, b_2, b_3) into one. Suppose the result is $(a_1, b_1, b_2, a_2, a_3, b_3)$. We can infer that the number of inversions in the original array is at least 4.

√ True ◯ False

3. (5 points) Count

Given an array (15, 6, 8, 31, 2, 9)

(a) (2') If we use basic Bubble Sort, the number of comparisons to sort the array into ascending order is _____, and the number of swaps is _____.

Solution: 15, 8

(b) (2') If we use Insertion Sort with swap, the number of comparisons to sort the array into ascending order is _____, and the number of swaps is _____.

Solution: 11, 8

(c) (1') Compare the run time of the two sorts in general, Bubble Sort _____ Insertion Sort (Fill in '>', '<', '=','≥','≤').

Solution: ≥

4. (6 points) Non-recursive merge sort

The Merge Sort we have learned was done in a recursive way. Let's explore a non-recursive method.

Conceptually, the merge sort process can be illustrated with a recursion tree. The recursive merge sort runs from top to bottom, while the non-recursive method works in a bottom-up way, starting with merging at the leaf nodes.

For example, for (5,4,2,3,1,6), we are merging (5), (4), (2), (3), (1), (6) into (4,5), (2,3), (1,6) during the first iteration (len=1). In the second iteration (len=2), and the merged results are (2,3,4,5), (1,6). The final iteration (len=4) the merged results are (1,2,3,4,5,6).

Now, please fill in the blanks.

```
// Assume arr.size() == temp.size() (length of the array)
void mergeSort(vector<int> &arr, vector<int> &temp) {
    for(int len = 1; len < arr.size(); len = len * 2){
        for (int i = 0; ; i = _________) {
            int left = i;
            int mid = i + len - 1;
            //Avoid exceeding the array length
            if(_________________) break;
            int right = min(_____________, array.size() - 1);
             //We need to merge two sub-arrays of size len into one array of size 2 *
                len
            merge(arr, left, mid, right, temp);
        }
    }
}

// This function merges the sub-array (a_left, ..., a_mid) and (a_mid+1, ..., a_right)
   into a single sorted array
void merge(vector<int>& arr, int left, int mid, int right, vector<int>& temp) {
    int length = right - left + 1;
    int p1 = left, p2 = mid + 1, i = 0;
    while (p1 <= mid && p2 <= right) {
        if(arr[p1] <= arr[p2]) temp[i++] = arr[p1++];
        else temp[i++] = arr[p2++];
    }
```

```
    while(p1 <= mid) temp[i++] = arr[p1++];
    while(p2 <= right) temp[i++] = arr[p2++];
    for (int j = 0; j < length; j++)
        arr[left+j] = temp[j];
}
```

Solution:

```
void mergeSort(vector<int> &arr, vector<int> &temp) {
    for(int len = 1; len < arr.size(); len = len * 2){
        for (int i = 0;  ; i = i + len * 2) {
            int left = i;
            int mid = i + len - 1;
            //Avoid exceeding the array length
            if(mid >= arr.size() - 1) break;
            int right = min(i + 2 * len - 1, array.size() - 1);
            //We need to merge two sub-arrays of size len into one array of size 2 *
                len
            merge(arr, left, mid, right, temp);
        }
    }
}

// This function merges the sub-array (a_left, ..., a_mid) and (a_mid+1, ..., a_right
    ) into a single sorted array
void merge(vector<int>& arr, int left, int mid, int right, vector<int>& temp) {
    int length = right - left + 1;
    int p1 = left, p2 = mid + 1, i = 0;
    while (p1 <= mid && p2 <= right) {
        if(arr[p1] <= arr[p2]) temp[i++] = arr[p1++];
        else temp[i++] = arr[p2++];
    }
    while(p1 <= mid) temp[i++] = arr[p1++];
    while(p2 <= right) temp[i++] = arr[p2++];
    for (int j = 0; j < length; j++)
        arr[left+j] = temp[j];
}
```