ShanghaiTech University

CS101 Algorithms and Data Structures

Fall 2020

Homework 4

Due date: 23:59, October 12, 2020

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL NAME to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
5. When submitting, match your solutions to the according problem numbers correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero grade.

1: (2'+2') Sorting Practice

Given array:

$$1, 5, 3, 8, 7, 2, 4, 6$$

we want to sort this array in ascending order in-place.
Please **show your steps** taken by each type of sorting method on this array:

Question 1. ***Quicksort.*** *After each partition during the algorithm, write the ordering of the list,* ***circle*** *the pivot that was used for that partition, and* ***underline*** *the sub-array being partitioned. Assume that the pivot is always the first item in the sublist being sorted.*

[1] 5 3 8 7 2 4 6
1 [5] 3 8 7 2 4 6
1 [3] 2 4 5 [8] 7 6
1 2 3 4 5 [7] 6 8
1 2 3 4 5 6 7 8

Note that depending on how the partition method is implemented, you may have different orderings of elements on either side of your pivot after a partition. The only property that must hold is that all elements less than the selected pivot go to the left, and all elements greater than the selected pivot go to the right.

Question 2. ***Merge sort****. Show the intermediate merging steps. In each merging step, circle the subarray that is to be merged.*

[1] [5] [3] [8] [7] [2] [4] [6]
[1 5] [3 8] [2 7] [4 6]
[1 3 5 8] [2 4 6 7]
1 2 3 4 5 6 7 8

2: (4×1') Identifying Sorts

Below you will find intermediate steps in performing various sorting algorithms that sort the input in ascending order on the same input list. The steps **do not necessarily represent consecutive steps in the algorithm** (that is, many steps are missing), but they are in the correct order. For each of them, **select the correct algorithm from the following choices**: insertion sort, bubble sort, quicksort (first element of sequence as pivot), and mergesort.

Input list: 1557, 2200, 5500, 1086, 443, 556, 2800, 7777, 4396, 0, 999

Question 3.
1557, 2200, 5500, 443, 1086, 556, 2800, 7777, 4396, 0, 999
1557, 2200, 443, 1086, 5500, 556, 2800, 7777, 0, 999, 4396
443, 1086, 1557, 2200, 5500, 0, 556, 999, 2800, 4396, 7777

Mergesort. One identifying feature of mergesort is that the left and right halves do not interact with each other until the very end.

Question 4.
1086, 443, 556, 0, 999, 1557, 2200, 5500, 2800, 7777, 4396
443, 556, 0, 999, 1086, 1557, 2200, 5500, 2800, 7777, 4396
0, 443, 556, 999, 1086, 1557, 2200, 2800, 7777, 4396, 5500

Quicksort. First item was chosen as pivot, so the first pivot is 1557, meaning the first iteration should break up the array into something like / < 1557 / = 1557 / > 1557 /

Question 5.
1086, 1557, 2200, 5500, 443, 556, 2800, 7777, 4396, 0, 999
443, 1086, 1557, 2200, 5500, 556, 2800, 7777, 4396, 0, 999
443, 556, 1086, 1557, 2200, 5500, 2800, 7777, 4396, 0, 999

Insertion Sort. Insertion sort starts at the front, and for each item, move to the front as far as possible. These are the first few iterations of insertion sort so the right side is left unchanged

Question 6.
1557, 2200, 1086, 5500, 443, 556, 2800, 7777, 4396, 0, 999
1557, 2200, 1086, 443, 556, 2800, 5500, 4396, 0, 999, 7777
1557, 2200, 443, 556, 1086, 2800, 5500, 4396, 0, 999, 7777

Bubble Sort. After the first iteration of bubble sort, the biggest element 7777 has reached the end of array.

3: (3×1') Single Choice

The following questions are single choice questions, each question has **only one** correct answer. Select the correct answer.
Note: You should write those answers in the box below.

Question 7	Question 8	Question 9

Question 7	Question 8	Question 9
A	C	D

Question 7. *Consider the quicksort algorithm which sorts elements in ascending order using the first element as pivot. Then which of the following input sequence will require a maximum number of comparisons when this algorithm is applied on it?*

(A) 22 25 56 67 89

(B) 52 25 76 67 89

(C) 22 52 67 25 76

(D) 52 25 89 67 76

Question 8. *Which of the following statements is **NOT** true?*

(A) The worst case time complexity of quicksort is $O(n^2)$.

(B) The average case time complexity of quicksort is $O(nlogn)$.

(C) On the same array, quicksort always performs faster than mergesort .

(D) Comparing to quicksort, mergesort requires additional space complexity.

Question 9. *Given extra information about the input array, we may design sorting algorithms that perform faster than $O(NlogN)$. Which of the following prior knowledge will lead to worst time complexity **slower** than $O(N)$?*

(A) Knowing the input array has no more than N inversions.

(B) Knowing the input array has exactly $(N^2 - N)/2$ inversions.

(C) Knowing the input array has less than N pairs of numbers that are not inversions.

(D) None of the above.

Question 10. ***(4')Stable sort***

We say a sort is "stable" if it ***always*** *preserves the original order of equal elements in an array. For example, if we have an array:* $4, 2, 3_a, 3_b, 1$*, after sorting it has a chance to become* $1, 2, 3_b, 3_a, 4$ *(* 3_a *and* 3_b *are equal when comparing), then this sorting algorithm is* ***NOT*** *stable.*

Among insertion sort, quicksort and mergesort, which of them are stable? (1') Briefly explain your answer. (3')

Insertion sort and mergesort are stable.

Insertion sort: During the insertion sort process, we will only swap the ordering of any two items if the item on the right is less than the item to its left. Therefore, the ordering of two equivalent items will always be preserved in insertion sort.

mergesort: During the merging process, we preserve the ordering of equivalent objects. If the first element of the left list being merged is equivalent to the first element of the right list, we always insert the element from the left list before the right list. Through this process we always preserve the initial relative ordering of equivalent items in the list, resulting in a stable sorting algorithm.

Quicksort: not stable. As a very simple counterexample, consider an array which contains the same elements. When selecting any item as pivot, that pivot item will finally be placed to where two pointers meet. Since this position is irrelevant to the pivot chosen, as a result, it is not stable.

(As a reminder: There does exist a method to heavily modify quicksort and makes it stable, but it requires additional space complexity. In fact, if we memorize the initial position of all elements, any sorting algorithm can become stable. So we don't consider this special case.)