Iterative Sorting Algorithms Efficiency & Extension

- Efficiency/time complexities for the topic are considered. Note that these discussions are not exhaustive or representative of the types of questions that may be found on the exams.
- Practice problems are given to test your understanding of the topic. These problems are not for a grade, but promote a deeper level of understanding of the topic. Unless otherwise specified, assume you are using *n* data/objects/elements in the data structure or algorithm.
- We look at the worst/average/best case of Big O; however, do not expect all three cases to be explored extensively.
- Feel free to discuss these problems in recitation or during office hours at the TA Help Desk.

Efficiency of Iterative Sorting Algorithms

Let's look at the iterative sorting algorithms. First, we explore the time complexities of these algorithms best, worst and average cases. We look at whether or not the algorithm takes advantage of the ordering of the data and is adaptive. We will also discuss stability (which focuses on the data), and In-place vs. Out-of-place (which focuses on memory storage) concepts.

Adaptive - An algorithm is adaptive if it runs faster or terminates earlier if the data it presorted.

Stability - An algorithm is considered stable if the duplicate data remains in the same relative order at the end of the algorithm, as it was at the start of the algorithm. For example: A_{Alok} appears before A_{Tim} prior to the algorithm running, and A_{Alok} appears before A_{Tim} after the algorithm concludes. Unstable algorithms will not preserve the relative ordering of the data. Note: if an algorithm swaps data that is not adjacent to each other in the container, then it will most likely be unstable.

In-place - The algorithm does not copy over all the data in a container into another data structure during an operation. Swapping or using temp variables is considered in-place.
Out-of-place - The algorithm moves the data from the container to a new external data structure during an operation.

Table is provided so you can record the information

Algorithm	Best Case	Average Case	Worst Case	Adaptive	Stable	In Place vs. Out of Place
Bubble Sort	O (n)	O (n^2)	O (n^2)	Yes	Yes	In
Cocktail Shaker Sort	O (n)	O (n^2)	O (n^2)	Yes	Yes	In
Insertion Sort	O (n)	O (n^2)	O (n^2)	Yes	Yes	In
Selection Sort	O (n^2)	O (n^2)	O (n^2)	No	No	In

Bubble Sort is the simplest algorithm to understand and explain. It compares two adjacent elements in the structure, and if the elements are out of order, then it swaps the elements. Effectively moving the largest element to the back (or smallest to the front). It does this using 2 nested loops. The inner loop iterates through each element comparing and swapping. The outer loop keeps track of where to end the inner loop. Thus, the algorithm is sorting *n* elements *n* times. The time complexity is O(n^2).

1) What is the average case time complexity of bubble sort, given some randomized array? O (n^2)____________________

2) What is the worst case time complexity of bubble sort, given an array where the data is in reversed order? O (n^2)____________________

If you implement an optimization in the bubble sort algorithm, where the sort terminates if the array is already sorted, then you will improve the time complexity

3) What is the best case time complexity of bubble sort, given an array where the data is already sorted? O (n)____________________

4) Is bubble sort adaptive?__________

5) Does bubble sort ever copy the elements into an external data structure?__________

6) Is bubble sort in or out of place?__________

7) Does bubble sort ever swap elements that are not adjacent to each other?__________

8) Is bubble sort stable?__________

Cocktail Shaker Sort is just as simple to understand and explain because it is just bubble sort forward and backwards. It compares two adjacent elements in the structure, and moving the largest element to the back of the array. Followed by comparing two adjacent elements at the back of the array moving the smallest to the front. The algorithm does have optimizations built in to terminate early if the data is sorted. The algorithm is still sorting *n* elements *n* times. The time complexity is O(n^2).

9) What is the average case time complexity of cocktail shaker sort, given some randomized array? O (n^2)____________________

10) What is the worst case time complexity of cocktail shaker sort, given an array where the data is in reversed order? O (n^2)____________________

If you implement an optimization in the cocktail shaker sort algorithm, where the sort terminates if the array is already sorted, then you will improve the time complexity

11) What is the best case time complexity of cocktail shaker sort, given an array where the data is already sorted? O (n)____________________

12) Is cocktail shaker sort adaptive? Yes__________

13) Does cocktail shaker sort ever copy the elements into an external data structure? No

14) Is cocktail shaker sort in or out of place? In

15) Does cocktail shaker sort ever swap elements that are not adjacent to each other? No

16) Is cocktail shaker sort stable? Yes

Insertion Sort is based on the premise that the subarray in the front is presorted. The next element is then inserted into the subarray in its correct order. This algorithm also compares two adjacent elements in the structure, and moving the new element to the correct position in the subarray. The optimization is already there because if the new element is larger than the last element in subarray, then no swaps are needed. However, the algorithm is still sorting *n* elements *n* times. The time complexity is O(n^2).

17) What is the average case time complexity of insertion sort, given some randomized array? O (n^2)

18) What is the worst case time complexity of insertion sort, given an array where the data is in reversed order? O(n^2)

19) What is the best case time complexity of insertion sort, given an array where the data is already sorted? O (n)

20) Is insertion sort adaptive? ____________

21) Does insertion sort ever copy the elements into an external data structure? ____________

22) Is insertion sort in or out of place? ____________

23) Does insertion sort ever swap elements that are not adjacent to each other? ____________

24) Is insertion sort stable? ____________

Selection Sort searches for a single largest element (max) in the entire array. When the search completes it moves the max to last position in the array. The algorithm repeats looking for the next largest, and so on. The algorithm has to search the entire array each time to be sure it found the max. There are no comparisons of adjacent elements, only comparisons with the max where ever it is in the array. The algorithm is sorting *n* elements *n* times. The time complexity is O(n^2).

25) What is the average case time complexity of selection sort, given some randomized array? O (n^2)

26) What is the worst case time complexity of selection sort, given an array where the data is in reversed order? O (n^2)

27) What is the best case time complexity of selection sort, given an array where the data is already sorted? O (n^2)

28)Is selection sort adaptive? No________

29)Does selection sort ever copy the elements into an external data structure? No________

30)Is selection sort in or out of place? In________

31)Does selection sort ever swap elements that are not adjacent to each other? Yes________

32)Is selection sort stable? No________

Extension
Conceptual

1. Watch the following sorting algorithm dances found in this Youtube playlist: https://www.youtube.com/playlist?list=PLOmdoKois7_FK-ySGwHBkltzB11snW7KQ . Watch insertion sort, bubble sort, merge-sort, quick-sort, and selection sort.

   A. What do you notice that is different about the way the creators of the video implemented bubble sort compared to the way that we asked you to implement in the homework?

   B. What do you notice that is different about the way the creators of the video implemented quick sort compared to the way that we asked you to implement in the homework?

2. Which of the sorting algorithms would sort the following array of integers most efficiently? Explain your answer.

Index	0	1	2	3	4	5	6	7	8
Element	0	1	2	3	4	5	6	-1	8

Insertion Sort because it is the best way to shift an element from end to front.

Diagramming

3. Perform bubble sort on the following array of numbers, showing the state of the array after each iteration. There may be more rows than necessary.

Index	0	1	2	3	4	5	6	7	8
Element	15	4	22	8	12	-4	10	2	1

4. Perform insertion sort on the following array of numbers, showing the state of the array after each iteration. There may be more rows than necessary.

Index	0	1	2	3	4	5	6	7	8
Element	15	4	22	8	12	-4	10	2	1

5. Perform selection sort on the following array of numbers, showing the state of the array after each iteration. There may be more rows than necessary.

Index	0	1	2	3	4	5	6	7	8
Element	15	4	22	8	12	-4	10	2	1

6. Perform cocktail shaker sort on the following array of numbers, showing the state of the array after each iteration. There may be more rows than necessary.

Index	0	1	2	3	4	5	6	7	8
Element	15	4	22	8	12	-4	10	2	1