Algorithms and Data Structure

Sheet 6

Valdrin Smakaj

Problem 6.1 *Bubble Sort & Stable and Adaptive Sorting* (9 points)

(a) (3 points) *Bubble Sort* is a sorting algorithm that works by repeatedly iterating through the list to be sorted, comparing each pair of adjacent items, and swapping them if they are in the wrong order. This is repeated until no swaps are needed, which indicates that the list is sorted. Write down the *Bubble Sort* algorithm in pseudocode including comments to explain the steps and/or actions.

(b) (2 points) Determine and prove the asymptotic worst-case, average-case, and best-case time complexity of *Bubble Sort*.

(c) (2 points) Stable sorting algorithms maintain the relative order of records with equal keys (i.e., values). Thus, a sorting algorithm is **stable** if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list. Which of the sorting algorithms *Insertion Sort*, *Merge Sort*, *Heap Sort*, and *Bubble Sort* are stable? Explain your answers.

(d) (2 points) A sorting algorithm is **adaptive**, if it takes advantage of existing order in its input. Thus, it benefits from the pre-sortedness in the input sequence and sorts faster. Which of the sorting algorithms *Insertion Sort*, *Merge Sort*, *Heap Sort*, and *Bubble Sort* are adaptive? Explain your answers.

Answer to sub-question (a) has been already solved and attached in the folder submitted

Answers to Question 2 has been already solved and attached in the folder submitted

---

(b) Now in order to determine and prove the asymptotic worst-case, average-case and best-case time complexity of this algorithm, we proceed:

1) Worst-case time complexity: Occurs when we have a decreasing array which means that in the inner loop which first starts at (n-1) continues on (n-2) and so on since every element will be swapped. So the overall time complexity will be:

$$\sum_{i=0}^{n-1} n-i-1 = n^2 - n - \sum_{i=0}^{n-1} i = n^2 - n - \frac{n(n-1)}{2} = \frac{2n^2-2n-n^2-n}{2} = \frac{n^2}{2} - \frac{n}{2}$$

$$\begin{aligned} T(n) &= O(n-1) + n \\ &= O(n^2) \end{aligned}$$

2) Average-case time complexity: Basically can be down to half of the swaps averagely but talking in terms of calculation to which O notation It belongs to, then we can also conclude that it has the same type O notation even though can be faster than the worse one since it depends on how many inner loop iterations it goes depending on how the elements are in the array. But still it goes over the minimum iterations to belong to the same type, because if it was perfect then it would belong to another case explained down below. So the complexity here is yet the same as in the worst case. $T(n) = O(n^2)$

3) Best-case time complexity: Occurs when the elements in the array are already sorted in perfect ascending order. Therefore there will only be one inner loop iteration for the checks which the swaps will be 0 since the elements are already in increasing order, so the loops breaks and in total we only have one loop iteration. Therefore, the time complexity for this case is: $T(n) = O(n)$

(c) The conclusions on the stability of the algorithms are

1) Insert Sort is stable because, during the iteration of the elements, the swap is done only if the element of the cursor is pointing to, is greater than the temporary one set as the element to be compared with. Therefore it holds its stability.

2) Merge Sort is stable because, during the splitting of the array constantly until it reaches the size where no further splits cannot be set then the merging is done with automatically ascending order, so, therefore even if we hold two same elements the one which was in order before the other in the unsorted array will still be in the first position in the sorted one because when the merging is done, it is automatically put it in the pre-position rather than the other one since it goes by compares of strictly lower or larger which in this case they're equal so it will put them in order to which was before the other in the unsorted one. To sum up, it is stable.

3) Heap Sort is unstable because, when we do the heap sort's build max heap, before the new sorting is done, the former positions of the elements in the to-be-sorted array are lost, therefore they'll just be put nearby each other but not in the order of (which was in a pre-position than the other) way. Conclusion: Not Stable.

4) Bubble Sort is stable because, when we do the loop iterations, we are only checking (in the swap condition) if the element is strictly greater than the following one, so if there are same elements, they'll be put in the order they were on their pre-position-un-sorted array. Therefore, it is Stable.

(d) The conclusions on the adaptivity of the algorithms are

(1) Insertion sort is adaptive because, when the array is already sorted, during the algorithm work, the only thing that will happen is just the iteration through the elements from the first to the end of the array. Moreover, we can also see it in the difference of the time complexity comparament:

Best Case Complexity: $T(n) = O(n)$
Worst Case Complexity: $T(n) = O(n^2)$

(2) Merge Sort is not adaptive because, although the elements are in the perfect order and sorted already, the array will be still split in sub-arrays with smaller size and merge the sizes in the end. Moreover, by the time complexity we get the same idea.

Best Case Complexity: $T(n) = O(n \log n)$
Worst Case Complexity: $T(n) = O(n \log n)$

(3) Heap Sort is not adaptive because, it does not take advantage of the pre-sortness of the array. It will still as its usual process of popping of the larges one and still make the process in the same time complexity even though there are less swaps and changes in the order since the array is sorted already, but that still does not change the total time complexity which is pretty much caused by the structure and work-method this algorithm has. Also seen below.

Best Case Complexity: $T(n) = O(n \log n)$
Worst Case Complexity: $T(n) = O(n \log n)$

(4) Bubble Sort is adaptive. The logic behind the explanation is pretty similar with the Insert Sort one since pretty much, since the array is already sorted and nothing needs to be swapped, it will only enter the inner loop once which, when it will go to the swap condition check, it won't make any swaps, leading to break the loop since the condition is not being satisfied (changing the bool variable state to True). Therefore the time complexity will be different and the performance is generally better (as also seen below). So we can come up with the conclusion that the Bubble Sort algorithm is Adaptive.

Best Case Complexity: $T(n) = O(n)$
Worst Case Complexity: $T(n) = O(n^2)$
