CS 1203: Data Structures **Semester:** Monsoon 2023

Assignment #3

Instructor: Subhamoy Mitra
Due: October 18, 2023

Student Name: Varsha Baisane
varsha.baisane_ug25#@ashoka.edu.in

2. Time complexities:

For Insertion Sort:

The best case complexity is $O(n)$, this is when the elements are already in sorted order and thus only one comparison takes place and there's no swap at all. Therefore, with n elements, we take only n-1 comparison giving us the best case complexity.

However, in the worst case, the elements are sorted completely descending at the beginning. In each step, all elements of the sorted sub-array therefore have to be shifted to the right so that the element to be sorted can be placed at the beginning as it is smaller than all the elements that are already sorted.
Therefore the worst case complexity is $O(n^2)$

Also, in average case considering that the elements are randomly ordered, each element, on average, is to be compared with half of the other elements in the array. As a result of this, there are approximately $(n^2)/2$ comparisons and swaps, which leads to an average-case time complexity of $O(n^2)$. Therefore, the average time complexity of insertion sort is: $O(n^2)$

For Bubble Sort:

Even for bubble sort, the algorithm must perform n-1 comparisons; therefore: the best-case time complexity of bubble sort is: $O(n)$

Similarly, as the elements would be placed in the descending orders, the algorithm will have to perform the maximum number of comparisons and swaps to get the array sorted. As a result, the worst-case time complexity of bubble sort is: (n^2)

In the average case, again one has about half as many exchange operations as in the worst case since about half of the elements would remain in the correct position compared to the neighboring element. Therefore, the average time complexity of bubble sort case is: $O(n^2)$

Experimental Time Complexity:

Experimental time complexity can be obtained by running these codes using arrays of different sizes and input characteristics, and the execution time could be measured using the time calls or clock functions. It is likely that insertion sort is faster than bubble sort for most of the cases.

3. Time Complexities:

For Merge Sort:

The best case complexity for merge sort is $O(nlogn)$ as in here, the array is already partially or completely sorted. Thus the division and merging requires fewer comparisons and operations compared to unsorted case. Therefore, the best case complexity: $O(nlogn)$

The worst-case time complexity of merge sort is also O(n log n). This is when the input array is in descending order or is completely unsorted. Even in this case, merge sort divides the array into halves and merges them efficiently.
Worst case complexity: $O(nlogn)$

The average-case time complexity of merge sort is again $O(nlogn)$. Here the array could be ordered partially or randomly and merge sort still consistently sorts it in log n time.
Thus, $O(nlogn)$

Quick Sort:

The best case time complexity for quick sort is when the pivot choice is good meaning that the partitions will be well-balanced. Therefore, the best case time complexity is $O(nlogn)$

The worst case time complexity occurs if the pivot element is always the smallest or largest element of the array leading to the array being not divided into two approximately equally sized partitions that results in a time complexity of $O(n^2)$.

The average case time complexity is when the algorithm performs averagely well that is due to random pivot selection ad thus the time complexity in this case is $O(nlogn)$.

Heap Sort:

Heap Sort consistently takes $O(nlogn)$ time to sort, as it relies on the heap data structure to maintain order. And therefore, the best, worst and average time complexity of heap sort is $O(nlogn)$.

Experimental Time Complexity:

To experimentally compare these sorting algorithms, we take in inputs and run these algorithms. The execution time can be measured using the timing or clock functions and it can be seen that merge sort is more efficient and works faster than quick sort when there's larger array size or data sets, while Quick sort is more efficient and works faster than merge sort in case of smaller array size or data sets. Both merge and quick sort are efficient algorithms compared to quick sort.

Source: happycoders.eu