Dunya Oguz 40181540
Hugo Joncour 40139230
Sarabraj Singh 29473858
John Purcell 27217439

Assignment 3

Question 1 (10 points)

As described in the notes, insertion sort goes sequentially through the array when making comparisons to find the proper place for the element that is currently being processed.

Suppose that the sequential search is replaced by a binary search.

a) Will the change decrease the overall cost of insertion sort? Explain.

Insertion sort with sequential searches for insertion has $O(n^2)$ complexity. This is because in insertion sort, we iterate through each element of an array and compare each element to its neighbour, swapping them with one another if the latter element is less than the other (assuming that we want to sort in increasing order). In the best case, we have an array that's already sorted, so we don't have to do any swaps when we make pairwise comparisons as we iterate through the array. Insertion sort then takes $O(n)$ time.

In the worst case, however, we have an array that's sorted in the reverse direction (increasing if we want to sort in decreasing order and vice-versa). In this situation, as we iterate through our array, we will have to do an increasing number of swaps at each iteration. We will start with 1 swap in the first element of the array, and end with $n-1$ swaps for the last element. As such, the version of insertion sort which uses sequential pairwise comparisons will take $n \cdot (n-1)$ at maximum, therefore having $O(n^2)$ complexity.

Theoretically we can switch the sequential search part of the insertion sort algorithm with binary search in order to decrease the time complexity of comparisons to $O\big(log(n)\big)$. This would make the overall run time cost of insertion sort $O\big(n \cdot log(n)\big)$. However, because we have to take into consideration not only the cost of comparison but also the cost of swapping elements – the overall time complexity of insertion sort with binary search will remain as $O(n^2)$. This is because, even if we are able to lower the time complexity of comparisons in insertion sort to $O\big(log(n)\big)$ – inserting a given element into the correct position in the array will still take $O(n)$ time since we might have to do $n-1$ swaps.

b) With respect to overall cost, will the modified version of insertion sort be in a faster-θ category than the version in the notes ? Explain.

No – because the lower and upper bounds of how long insertion sort can take will not change. It will remain as $O(n)$ for the lower bound (best case) and $O(n^2)$ for the upper bound (worst case).

Dunya Oguz 40181540
Hugo Joncour 40139230
Sarabraj Singh 29473858
John Purcell 27217439

Question 2 (10 points)

As Consider applying radix sort to a list of 10,000 integers in the range from 1,000,000 to 9,999,999.

a) Not counting the space needed to store the integers themselves, how many bytes of memory would be required? Assume pointers occupy 4 bytes.

Radix sort takes $O(n + N)$ space where:

$$n = number of items$$
$$N = Keys(0\text{to}9)$$

$$=10000 + 10$$
$$=10010 * 4$$
$$=40040 bytes$$

b) How many times would radix sort examine each value in the list?

Since there are 7 keys: $(1_1 0_2 0_3 0_4 0_5 0_6 0_7)$, and each value will be examined once per key, with Radix sort, each value would be examined 7 times.