Programming Assignment 1 – Intro to Algorithms (COMP 3270) – Jewels Wolter

Results:

Algorithm	Array Type	N(min)	T(min)	N(max)	T(max)
S	C	10.00	6.90E-06	10000.00	2.20E+00
S	S	10.00	6.40E-06	10000.00	2.14E+00
S	R	10.00	8.00E-06	10000.00	2.20E+00
Q	C	10.00	1.64E-05	100.00	7.90E-04
Q	S	10.00	1.40E-05	100.00	8.00E-04
Q	R	10.00	1.55E-05	100.00	8.00E-04
I	C	10.00	2.89E-06	10000000.00	1.10E+00
I	S	10.00	3.00E-06	10000000.00	1.02E+00
I	R	10.00	3.40E-06	100000.00	1.00E-02
M	C	10.00	1.50E-05	100000.00	3.40E+00
M	S	10.00	1.60E-05	100000.00	4.08E+00
M	R	10.00	1.50E-05	100000.00	3.30E-01

Analysis:

Algorithm	Array Type	T(max)/T(min)	n ratio	n ln(n) ratio	n^2 ratio	Behavior/ Closest To
S	C	318840.58	1000.00	4000.00	1000000.00	nlogn
S	S	334375.00	1000.00	4000.00	1000000.00	nlogn
S	R	275000.00	1000.00	4000.00	1000000.00	nlogn
Q	C	48.20	10.00	20.00	100.00	nlogn
Q	S	57.14	10.00	20.00	100.00	nlogn
Q	R	51.61	10.00	20.00	100.00	nlogn
I	C	380622.84	1000000.00	7000000.00	1000000000000.00	n
I	S	340000.00	1000000.00	7000000.00	1000000000000.00	n
I	R	2941.18	10000.00	50000.00	100000000.00	n
M	C	226666.67	10000.00	50000.00	100000000.00	nlogn
M	S	255000.00	10000.00	50000.00	100000000.00	nlogn
M	R	22000.00	10000.00	50000.00	100000000.00	n

Selection Sort:

Theoretically the best, worst, and average cases for selection sort should be O(n^2) no matter the type of the array since it has to iterate through and look at every element in the array to find the minimum then swap it with the first unsorted element.

It makes sense that for each array type, the time complexity that T(max)/T(min) is closest to are the same for each, but it does not necessarily make sense that it is nlogn.

This result is surprising because I would expect the values for T(max)/T(min) would be closest to the n^2 ratio in order to match up with its theoretical time complexity. This result might be able to be explained because the ratio of n^2 is so large that it might just take a larger n(max) to get an accurate depiction of the data.

Quick Sort:

Theoretically the best and average cases are nlogn, which match with my analysis of the quicksort algorithm for any array type.

Theoretically the worst case for quicksort is O(n^2) which happens where there is a poorly chosen pivot. This case may happen when the largest or smallest value is chosen for the pivot. This case would fit into the sorted array case of my analysis. The result of the sorted array and quicksort actually ended up having the highest T(max)/T(min) for any of the quicksort cases, making it the closest to the n^2 ratio out of any of the cases but not quite high enough to be closest to it compared to the nlogn ratio.

I think a more accurate result could have been obtained by a larger N(max) but during each attempt to run with any array larger than 100 elements, I received a maximum recursion depth exceeded error. The results were heading in the direction of the sorted array type behaving in O(n^2), though.

Insertion Sort:

Theoretically the best case for insertion sort would be n, which matches all of the cases that I tested. This best case happens when all of the elements are already sorted, because during insertion sort when each element is compared to another in a sorted array it finds that the elements are already in the correct place in the array, so no shifting is needed. The same goes for in an array where all of the elements are the same. All of this makes it so that for array type C and S, my results make sense since they evaluated closest to ratio n.

Insertion sort at its average or worst case though should theoretically evaluate to n^2. This is because in an average case, the algorithm has to compare each element to many other elements and many shifts will usually have to be done. The average case would have been obtained by looking at a random array. My results for the random array did not reflect it being n^2, though, which is surprising. I could explain these results because as I was evaluating insertion sort for a random array at N(max), I was unable to use the N(max) I used for the constant and sorted arrays because it resulted in a timeout.

I think with a more accurate ratio and result for T(max)/T(min) in the random array case, the result would more accurately reflect the theoretical time complexity of the algorithm.

Merge Sort:

Theoretically, the best, average, and worst case for merge sort is nlogn, which matches for the constant and sorted arrays cases in my results. This makes sense because no matter the type of array the algorithm always divides and merges the array in the same way.

In my results, though, for a random array evaluated the T(max)/T(min) value more closely to the n ratio. This result was surprising, but it could have been caused from a larger N(max), but when trying to run anything larger than 100000, the system timed out. Despite this, the value of T(max)/T(min) that I obtained was almost high enough to be closer to nlogn.

I think in this case that with a more accurate ratio and result for T(max)/T(min) in the random array case, the result would more accurately reflect the theoretical time complexity of the algorithm.