John Gangemi
U6871-4612
COT4400 Analysis of Algorithms
Project One
10/18/2015

Algorithm Analysis

Algorithm: SelectionSort

Increasing:	$S(n) \approx 109.9423n^2 - 1.6075nlgn + 4.7588n - 8.9439$	⇨ $\Theta(n^2)$
Decreasing:	$S(n) \approx 96.4957n^2 + 40.5709nlgn - 45.7475n + 52.1670$	⇨ $\Theta(n^2)$
Random:	$S(n) \approx 117.8664n^2 - 56.6752nlgn + 63.7592n - 70.4751$	⇨ $\Theta(n^2)$
Constant:	$S(n) \approx 111.6686n^2 - 22.1324nlgn + 34.6618n - 39.1476$	⇨ $\Theta(n^2)$

Algorithm: InsertionSort

Increasing:	$S(n) \approx 0n^2 + 0nlgn + 0n + 0$	⇨ $O(n)$
Decreasing:	$S(n) \approx 105.4342n^2 - 40.2297nlgn + 62.0483n - 71.1063$	⇨ $O(n^2)$
Random:	$S(n) \approx 19.8826n^2 + 284.4600nlgn - 351.5847n + 402.0988$	⇨ $O(n^2)$
Constant:	$S(n) \approx 0n^2 + 0nlgn + 0n + 0$	⇨ $O(n)$

Algorithm: MergeSort

Increasing:	$S(n) \approx 1.0411n^2 - 8.3231nlgn + 18.3328n - 12.8473$	⇨ $O(n^2)$
Decreasing:	$S(n) \approx 0.6772n^2 - 4.6588nlgn + 13.0744n - 6.9220$	⇨ $O(n^2)$
Random:	$S(n) \approx 0.5955n^2 - 5.0337nlgn + 14.9850n - 7.4596$	⇨ $O(n^2)$
Constant:	$S(n) \approx -0.4488n^2 + 4.6123nlgn + 0.6920n + 9.4888$	⇨ $\Theta(nlgn)$

Algorithm: QuickSort

Increasing:	$S(n) \approx -0.1667n^2 + 1.5444nlgn - 1.1080n + 3.3477$	⇨ $\Theta(nlgn)$
Decreasing:	$S(n) \approx 18.3547n^2 + 13.2235nlgn - 16.2088n + 19.9018$	⇨ $O(n^2)$
Random:	$S(n) \approx -0.0319n^2 + 0.3020nlgn + 2.1642n - 0.2594$	⇨ $\Theta(nlgn)$
Constant:	$S(n) \approx 79.5137n^2 - 0.2737nlgn + 1.6010n + 0.7101$	⇨ $O(n^2)$

Results from the regression equations for SelectionSort coincide with the theoretical analysis in all permutations of a given input array.

Regression equations for InsertionSort show unusual behavior with increasing and constant permutations of an input array. Therefore, it can be noted that InsertionSort performs exceptionally well with “pre-sorted” data, at most n iterations must be performed as proven by examining the algorithm. I believe the zero coefficients for the regression equations is purely a characteristic of MATLAB.

Mergesort showed the most uniform and efficient results for each of the array permutations. However, MATLAB results tell a different story for all but constant input data. I cannot begin to rationalize why MergeSort performs well in real-world simulation but fails to meet theoretical analysis in MATLAB simulations.

Quicksort has worst-case complexity when the input data is decreasing or constant. The is a fundamental characteristic of the algorithm where data completely out of order (decreasing or constant) can cause abnormally high use of the stack thus increasing execution run-time.