Igli Duro

COT 4400

Analysis of Algorithms

Project 1

Results

	n_{min}	t_{min}	n_{max}	t_{max}
SC	5900	30	750000	463907
SS	5900	30	750000	464371
SR	5800	30	750000	464729
IC	12000000	30	230000000	559
IS	12000000	30	230000000	556
IR	8000	30	5000000	433649
MC	650000	30	230000000	14123
MS	650000	30	230000000	13640
MR	290000	30	230000000	27530
QC	7000	30	750000	326877
QS	465000	30	100000000	373116
QR	245000	30	155000000	309749

Analysis

	t_{max}/t_{min}	n ratio	nln(n) ratio	n^2 ratio	Behavior
SC	15464	127	198	16159	n^2
SS	15479	127	198	16159	n^2
SR	15491	129	202	16721	n^2
IC	19	19	23	367	n
IS	19	19	23	367	n
IR	14455	625	1073	390625	n^2
MC	471	354	509	125207	nlg(n)
MS	455	354	509	125207	nlg(n)
MR	918	793	1214	629013	n
QC	10896	107	164	11480	n^2
QS	12437	215	304	46248	n^2
QR	10325	633	961	400250	nlg(n)

	Best-case complexity	Average-case complexity	Worst-case complexity
SelectionSort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
InsertionSort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
MergeSort	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$
QuickSort	$\Omega(n \lg n)$	$\Theta(n \lg n)$	$O(n^2)$

SelectionSort

Constant Array

The behavior observed on an array of constants was n^2, this matches the expected behavior as SelectionSort is always O(n^2) time complexity.

Sorted Array

The behavior observed on a sorted array was n^2, this matches the expected behavior as SelectionSort is always O(n^2) time complexity.

Random Array

The behavior observed on an array of random values was n^2, this matches the expected behavior as SelectionSort is always O(n^2) time complexity.

InsertionSort

Constant Array

The behavior observed on an array of constants was n, this matches the best-case complexity and is expected as a constant array is considered sorted.

Sorted Array

The behavior observed on a sorted array was n, this matches the expected complexity as InsertionSort performs best, O(n), on a sorted array.

Random Array

The behavior observed on an array of random values was between that of nln(n) and n^2 but closer in behavior to n^2. This would make sense as the average complexity of InsertionSort is O(n^2) meaning most elements were out of order. In this experiment it looks like a good majority were out of order, maybe somewhere around 3/4ths of the elements.

MergeSort

Constant Array

The behavior observed on an array of constants was nln(n), this matches the expected complexity as MergeSort has an best, average, and worst-case complexity of O(nln(n)).

Sorted Array

The behavior observed on a sorted array was nln(n), this matches the expected complexity as MergeSort has an best, average, and worst-case complexity of O(nln(n)).

Random Array

The behavior observed on an array of random values was between that of n and nln(n) but was closer to that of n. This is surprising as the MergeSort algorithm recursively splits an array in half until there's one element, then sorts it. So, the expected time complexity would be O(lgn) for the recursion * O(n) for the merging for a total of O(nlg(n)). Multiple trials were ran with little variance in time. Upon further research there exists an optimization called Natural MergeSort that will merge subsequent arrays if they're already sorted, to save time. However, the MergeSort algorithm used here was not of that variant. If it were, would see O(n) behavior in the constant and sorted array experiments as-well. Perhaps with a larger array size (closer to 1 billion) we would result in O(nlgn) behavior.

QuickSort

Constant Array

The behavior observed on an array of constants was n^2, this matches the expected worst case time complexity of O(n^2) as QuickSort relies on picking a good pivot. So, it performs its worst in a constant array where a good pivot is not possible.

Sorted Array

The behavior observed on a sorted array was between that of nln(n) and n^2, but closer in behavior to n^2. This makes sense as QuickSort tends to perform worse on sorted or reverse sorted arrays. So, its harder to pick a good pivot and the time complexity suffers. The QuickSort for this experiment uses a random element for the pivot. An average-case complexity of O(nlgn) would be possible here, but it would've needed to pick a pivot closer to the middle of the array.

Random Array

The behavior observed on an array of random values was between that of nln(n) and n^2, but closer in behavior to nln(n). This is expected as an average-case complexity of O(nlgn) would makes sense for a random array. Since you would be more likely to pick a good pivot due to probability and a large array size.