Khyati Mahajan

Comparison Based Sorting

Introduction

The methods being studied for this project consist of the following algorithms for sorting:

1. Insertion sort
2. Merge sort
3. Quick sort (in-place)
4. Modified quick sort (using median of three as pivot)

Runtime analysis

This section details the theoretical and experimental runtime of the algorithms, and comparative analysis.

Theoretical Runtime

The theoretical runtime analysis of the given algorithms:

Runtime	Insertion sort	Merge sort	In-place Quick sort	Modified Quick sort
Best-case	O(n)	O(nlogn)	O(nlogn)	O(nlogn)
Worst-case	$O(n^2)$	O(nlogn)	$O(n^2)$	$O(n^2)$

Experimental Runtime (random inputs, sample run)

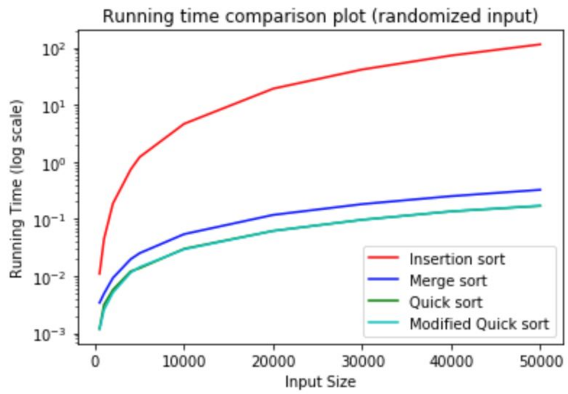
Runtime results:

Input Size	500	1000	2000	4000	5000	10000	20000	30000	40000	50000
Insertion Sort	0.03143	0.11621	0.48393	1.89506	2.97783	11.6940	47.3874	109.870	193.413	298.020
Merge Sort	0.00824	0.00984	0.02132	0.04586	0.05868	0.12567	0.26834	0.41693	0.56883	0.72556
Quicksort	0.00425	0.00631	0.01296	0.02912	0.03713	0.08403	0.17247	0.27279	0.37913	0.48569
Modified quicksort	0.00425	0.00649	0.01294	0.02900	0.03690	0.08396	0.17263	0.27259	0.38061	0.48505

Amortized results:

Input Size	500	1000	2000	4000	5000	10000	20000	30000	40000	50000
Insertion Sort	6.28e-5	1.16e-4	2.41e-4	4.73e-4	5.95e-4	1.16e-3	2.36e-3	3.66e-3	4.83e-3	5.96e-3
Merge Sort	1.64e-6	9.84e-6	1.06e-5	1.14e-5	1.17e-5	1.25e-5	1.34e-5	1.38e-6	1.42e-5	1.45e-5
Quicksort	8.50e-6	6.31e-6	6.48e-6	7.28e-6	7.42e-6	8.40e-6	8.62e-6	9.09e-6	9.47e-6	9.71e-6
Modified quicksort	8.50e-6	6.49e-6	6.47e-6	7.25e-6	7.38e-6	8.39e-6	8.63e-6	9.08e-6	9.51e-6	9.70e-6

Plots:



Special Case Performance

The recursion depth was getting exceeded, so the input sizes have been limited to 500 for studying special case performance.

Sorted Input

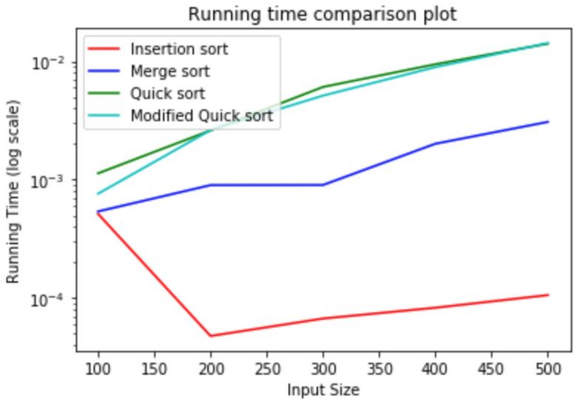
Runtime results:

Input Size	100	200	300	400	500
Insertion Sort	0.000132	0.000108	0.000204	0.000307	0.000406
Merge Sort	0.001228	0.002965	0.004786	0.005957	0.004210
Quicksort	0.002886	0.010667	0.023244	0.027021	0.037875
Modified quicksort	0.002860	0.011158	0.016796	0.024836	0.037711

Amortized results:

Input Size	100	200	300	400	500
Insertion Sort	1.080e-6	1.024e-6	1.023e-6	1.017e-6	1.274e-6
Merge Sort	1.228e-5	1.228e-5	1.595e-5	1.489e-5	8.421e-6
Quicksort	2.886e-5	5.333e-5	7.748e-5	6.755e-5	7.575e-5
Modified quicksort	2.860e-5	5.579e-5	5.598e-5	6.209e-5	7.542e-5

Plots:



Reverse Sorted Input

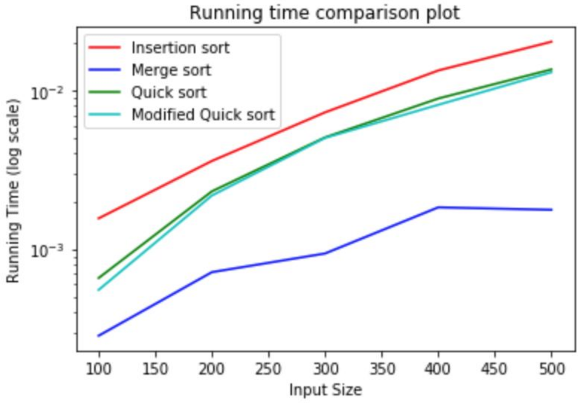
Runtime results:

Input Size	100	200	300	400	500
Insertion Sort	0.005059	0.013921	0.024141	0.044173	0.059211
Merge Sort	0.001205	0.002897	0.005258	0.006546	0.005431
Quicksort	0.003201	0.010826	0.014683	0.031641	0.040731
Modified quicksort	0.003212	0.011523	0.019056	0.024992	0.039253

Amortized results:

Input Size	100	200	300	400	500
Insertion Sort	5.059e-5	6.960e-5	8.047e-5	1.104e-4	1.184e-4
Merge Sort	1.205e-5	1.448e-5	1.752e-5	1.636e-5	1.086e-5
Quicksort	3.201e-5	5.413e-5	4.894e-5	7.910e-5	8.146e-5
Modified quicksort	3.006e-5	5.439e-5	6.486e-5	6.218e-5	7.515e-5

Plots:



Observations

1. The experimental runtime analysis generally agrees with the theoretical expectations.
2. Quick sort works better as an average when the input size is larger.
3. Insertion sort performs worse with increasing input size.
4. For the sorted input:
    a. Insertion sort works best, as expected.
    b. Quick sort works worst, as there is a possibility that the pivot selection is causing a worst case scenario.
5. For the reverse sorted input:
    a. Insertion sort works worst, as this is the worst case scenario.
    b. Merge sort works best.
6. Modified quick sort generally performs better than quicksort, as it has the advantage of choosing a somewhat better pivot.
7. Merge sort runs in almost the same amortized running time, no matter the input type, agreeing with theoretical analysis.