Project 1 Analysis

Kevin Orr

1/1/1111

1 Results

The provided program was run with all combinations of the sorting algorithms selection sort, insertion sort, merge sort, and quicksort; and input arrays that are strictly increasing, strictly decreasing, constant, or randomized. From these results we derive four variables: $n_{min}, t_{min}, n_{max}, t_{max}$. They are defined as follows:

n_{min} The smallest array that took at least 20 ms to sort

t_{min} The time it took to sort the array of size n_{min}

n_{max} The largest array that took no longer than 10 minutes to sort

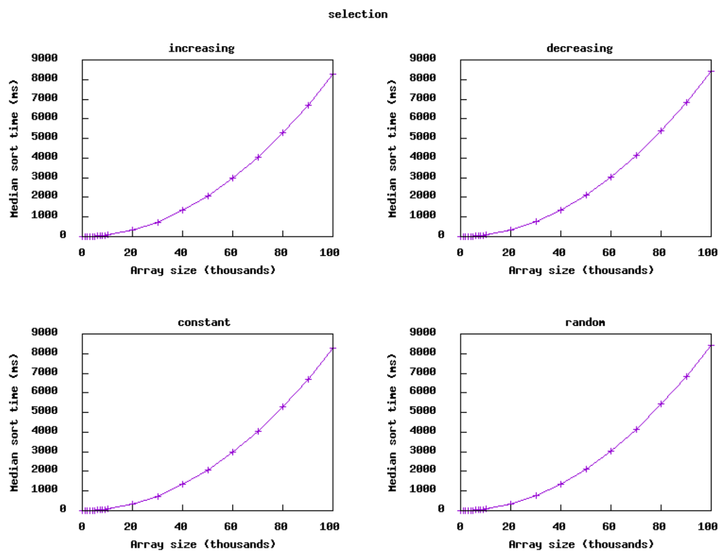
t_{min} The time it took to sort the array of size n_{max}

Algorithm	Input Type	n_{min}	t_{min}	n_{max}	t_{max}
selection	increasing	10000	82	100000	8285
selection	decreasing	10000	84	100000	8421
selection	constant	10000	82	100000	8283
selection	random	10000	84	100000	8420
insertion	increasing	10000000	24	1000000000	2472
insertion	decreasing	10000	46	1000000	463830
insertion	constant	10000000	24	1000000000	2477
insertion	random	10000	46	1000000	464006
mergesort	increasing	1000000	42	1000000000	62201
mergesort	decreasing	1000000	115	1000000000	172232
mergesort	constant	1000000	43	1000000000	62500
mergesort	random	1000000	115	1000000000	172469
quicksort	increasing	1000000	33	1000000000	46599
quicksort	decreasing	1000000	116	1000000000	170442
quicksort	constant	10000	35	100000	3512
quicksort	random	1000000	116	1000000000	170431

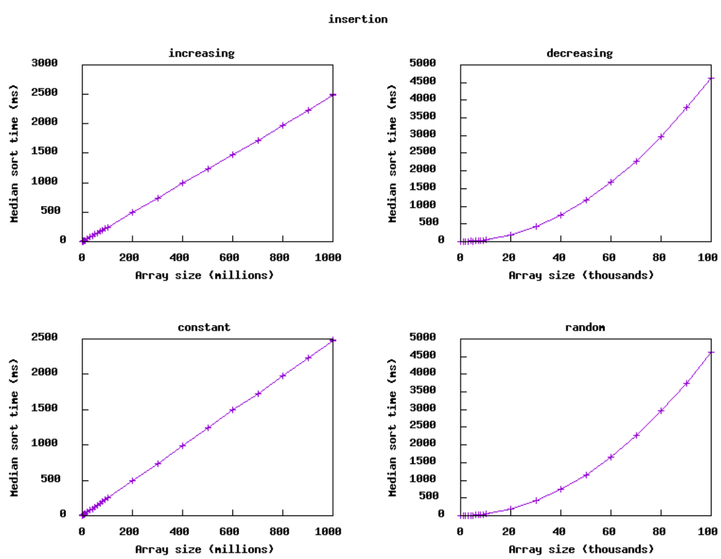
2 Plots

The following plots show the behavior for our four sorting algorithms. Each algorithm was tested with array sizes of $n10^m \leq 10^9$ for $n \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}, m \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

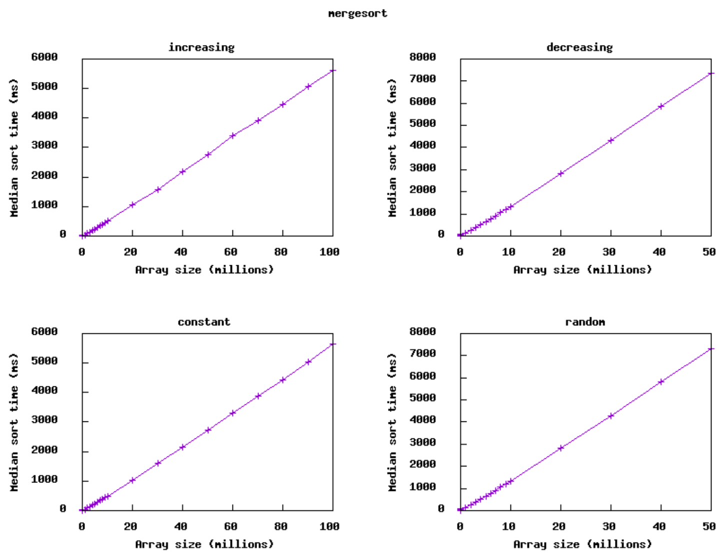
2.1 Selection Sort



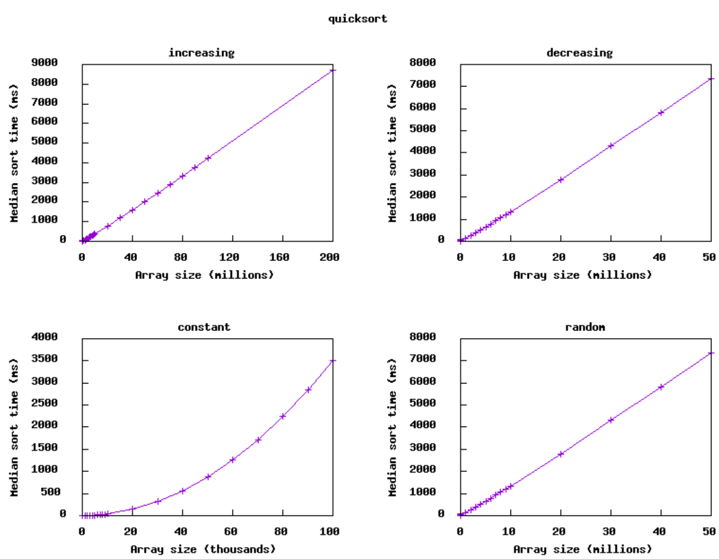
2.2 Insertion Sort



2.3 Merge Sort



2.4 QuickSort



3 Analysis

The results from section 1 were then used in the following calculations to determine the time complexity of the sorting algorithms. For each $f_i \in \left\{f_1(n) = n, f_2(n) = n \lg n, f_3(n) = n^2\right\}$, I compared $f_i(n_{max})/f_i(n_{min})$ to t_{max}/t_{min}. Ideally, the first ratio should be sufficiently close to the second when $f_i(n)$ corresponds to the theoretical time complexity of the algorithm of interest. These ratios are shown in the following table:

Algorithm	Input Type	t_{max}/t_{min}	n_{max}/n_{min}	$\frac{n_{max} \lg n_{max}}{n_{min} \lg n_{min}}$	n_{max}^2/n_{min}^2
selection	increasing	101.03658	10.0	12.5	100.0
selection	decreasing	100.25	10.0	12.5	100.0
selection	constant	101.01219	10.0	12.5	100.0
selection	random	100.2381	10.0	12.5	100.0
insertion	increasing	103.0	100.0	128.57143	10000.0
insertion	decreasing	10083.261	100.0	150.0	10000.0
insertion	constant	103.208336	100.0	128.57143	10000.0
insertion	random	10087.087	100.0	150.0	10000.0
mergesort	increasing	1480.9762	1000.0	1500.0	1000000.0
mergesort	decreasing	1497.6696	1000.0	1500.0	1000000.0
mergesort	constant	1453.4884	1000.0	1500.0	1000000.0
mergesort	random	1499.7305	1000.0	1500.0	1000000.0
quicksort	increasing	1412.091	1000.0	1500.0	1000000.0
quicksort	decreasing	1469.3276	1000.0	1500.0	1000000.0
quicksort	constant	100.34286	10.0	12.5	100.0
quicksort	random	1469.2328	1000.0	1500.0	1000000.0

3.1 Theoretical time complexities

Algorithm	Best-case	Average-case	Worst-case
selection	$\Theta\left(n^2\right)$	$\Theta\left(n^2\right)$	$\Theta\left(n^2\right)$
insertion	$\Omega(n)$	$\Theta\left(n^2\right)$	$\mathrm{O}\left(n^2\right)$
mergesort	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$
quicksort	$\Omega(n \lg n)$	$\Theta(n \lg n)$	$\mathrm{O}\left(n^2\right)$

3.2 Inferred time complexity of each case

After comparing the two previous tables, the following conclusions were made:

Algorithm	Input Type	Complexity	Case
selection	increasing	$\Theta(n^2)$	Any
selection	decreasing	$\Theta(n^2)$	Any
selection	constant	$\Theta(n^2)$	Any
selection	random	$\Theta(n^2)$	Any
insertion	increasing	$\Theta(n)$	Best
insertion	decreasing	$\Theta(n^2)$	Worst
insertion	constant	$\Theta(n)$	Best
insertion	random	$\Theta(n^2)$	Average
mergesort	increasing	$\Theta(n \lg n)$	Any
mergesort	decreasing	$\Theta(n \lg n)$	Any
mergesort	constant	$\Theta(n \lg n)$	Any
mergesort	random	$\Theta(n \lg n)$	Any
quicksort	increasing	$\Theta(n \lg n)$	Best or Average
quicksort	decreasing	$\Theta(n \lg n)$	Best or Average
quicksort	constant	$\Theta(n^2)$	Worst
quicksort	random	$\Theta(n \lg n)$	Best or Average

4 Remarks

Most of the timed results were quite spot-on when compared to the theoretical time complexities. The most inaccurate ratio was quicksort with an increasing input array, but this was only ~5.9% off.