Algorithm Analysis and Timing

Computing professionals have two methods for comparing the algorithms' performance, one static (i.e., analysis by people) and the other dynamic (i.e., run on a computer). The process of static analysis of algorithms is mathematical proof and validated through peer review in published papers. The dynamic analysis uses statistical analysis across running algorithms and, while also often validated in published papers, reflects the intricacies that the real world might introduce. It is helpful to understand both variants, so this project will have you working within a team to produce both types of analysis.

Analyzing Sorting Algorithms

Static Analysis

Your team will document the 'order N' performance for each of the sorting algorithms using the notation and analysis techniques noted in the course materials. Rank the algorithms by their expected speed under three conditions:

- 'Nominal' input data (randomly sorted)
- Already sorted data
- Inversely sorted data

This analysis does not need to be 'correct' but merely capture your team's discussion and expectations. Do ***not*** go back and 'fix' your analysis after completing the next step. Your document should create a table similar to the following.

Algorithm	'Order N'	Rank for Random	Rank for Sorted	Rank for Inverse
Bubble		1	7	5
Selection		2	6	2
Insertion		3	5	3
Shell		4	4	1
Merge		5	3	6
Heap		6	2	7
Quick		7	1	4

Dynamic Analysis

The test cases for the sorting algorithms exercise include a framework for capturing the time it takes each algorithm to sort data across three data profiles to be sorted – randomly distributed data, data already sorted, and data 'inversely' sorted. The framework makes sure the same random distribution is given to each algorithm, but many factors might influence the raw timing numbers on a single test. For one thing, your computer might run a background process (e.g., checking email, scanning for viruses) that interrupts one algorithm longer than another. The timing of a single run is not always indicative of the exact differences between the two algorithms. Using statistical analysis across many trials and data profiles offers a better picture of the algorithms' performance. Additionally, some algorithms are so much faster than others, and analyzing all algorithms at the same 'load' (i.e., number of items to sort) adds little value. It

will probably become quickly clear when to 'stop testing' one algorithm, which is clearly 'slower' than the other, but you might need an additional load to understand the remaining algorithms better.

In addition to the variations that can happen on one computer, algorithms might perform differently on different hardware, not to mention possible variations of how each programmer constructs their version of the algorithm. Having a team allows further analysis since you can test each team member's algorithm on each team member's computer to gather even more data about possible differences in your algorithms' performance. Each team member should gather data about the same dynamic performance tests on their device for every team member's algorithms and report the data as described in the next section.

Reporting Performance Data

In addition to the static analysis table above, you should report a summary of the statistical analysis across each variation (e.g., input design, team member) to obtain a full view of your algorithm's performance. For example, you might choose to test the quicksort's performance on random data with and without shuffling (it likely will fail if you do not shuffle the sorted data). Summarize the testing you have completed by creating a new table similar to the one above in the Static Analysis section, only for interesting variations in your data. Include a summary of your testing and findings that answer some of the following questions.

- Was there a clear 'winner' or clear 'losers' in the data?
- Under which circumstances should designers choose one algorithm over the other?
- Were there any notable differences between any of the sub-categories (e.g., variances across devices, algorithm implementations, testing strategies)
- Are there any recommendations for performing such testing in the future?

Designing your own Tests

Sorting is hardly the only algorithms where speed matters! In this course, we also created an array list and a linked list, both iteratively and recursively. Use your newfound knowledge about algorithmic analysis and the example test framework from sorting to conduct a similar analysis on these data structures. You should specifically test:

- Adding to each
- Getting a value from each
- Removing a value from each