Sort Detective Report

By Chris Joy, Allan Chen & Alexander Nguyen

In this report we aim to deduce the sorting algorithm used by type programs, without knowing their implementation details. In order to do this, we will need to measure the running time of each program and correlate them with a time complexity.

# Sorting Algorithms

The table below contains various sorting algorithms discussed during the lectures with their relevant time complexities.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Best | Average | Worst | Stability |
| Bubble Sort |  |  |  | Stable |
| Bubble Sort EE |  |  |  | Stable |
| Selection Sort |  |  |  | No |
| Insertion Sort |  |  |  | Yes |
| Shell Sort |  | . |  | No |
| Quick Sort |  |  |  | No |
| Heap Sort |  |  |  | No |
| Merge Sort |  |  |  | Yes |

# Experiment Design

Our experiments will essentially consist of two phases.

1. Verify our sorting programs behave correctly (i.e. they’re actually sorting items\_.
2. Measure runtime performance of both algorithms, with various types of inputs.

In order to conduct these experiments, we’ll write and run a shell script, consisting of various tasks needed, in order to complete the requirements above.

We’ll also need to consider various factors when writing and running the experiments, such as:

* Device used to run experiment (lab server / login server)
* System time vs Program time vs User time etc
* Time used to write to file
* Seed used when randomly generating a list of numbers

During testing, the MODE (most common item) will be added to the final results table.

# Correctness Analysis

# Performance Analysis

# Experimental Results

## Correctness Experiments

## Performance Experiments

# Conclusions