James Stell
Algirithms 6114
10/4/2015

PROJECT 1 PART 1

INTRODUCTION

In this project four sorting algorithms were implemented and performance was measured. The four sorting algorithms tested were insertion sort, selection sort, bubble sort, and brute force sort. Each sorting algorithm was tested on varying array sizes from 2 to around 2 million. Each iteration was kept to 5 minutes or under, and this happened somewhere around 2 million elements.

MAIN PROGRAM

My main program and sorting algorithms were written in c++. The compiler optimization flag of '-O3' was used to boost performance during runtime. With any given array size, each sorting algorithm runs 30 times to calculate an average run time. The 30 times are used to calculate a mean time and standard deviation. The results are printed to a text file named 'Time-Output.txt' .

My program accepts 3 arguments from the command line for execution. It accepts the name of a file to process data from, how many elements to process, and a sorting algorithm flag. The sorting algorithm flags are '-i' for insertion sort, '-s' for selection sort, '-b' for bubble sort, and '-bf' for brute force.

I wrote a simple c++ program to generate the numbers to be tested in this project. The numbers are randomly generated and are written to a file called nums.txt.

Files for main program:

- main.cpp → The file that accepts input from the user, reads numbers into an array from the file, calls the specified sorting algorithm, manages the timing of the sorting algorithm runs, and outputs results to file 'Time-Outputs.txt'.
- sort.h → The header file for the sort class, contains method signatures.
- sort.cpp → The source file for the sorting class. This file contains the sorting logic of the four sorting algorithms. Each method is static and is wrapped into a class named Sort.

PROGRAM EXECUTION

The program was executed and tested inside of a Python program. The python program executed each sorting algorithm with array sizes starting at 2 and doubling each iteration until a single iteration took longer than 2.5 hours.

The python program is named "proc-spawn.py".

GRAPH GENERATION

Python's matplotlib was used for the parsing and graphing of the main programs results. The program file is named "python-graph.py". The program starts by using regular expressions to parse the results from the Time-Output file. After the array sizes and times were parsed out the resulting graph was drawn. The x-scale is log base 2 and the y scale is also log base 2. When the bases are not logs the graph appears normal, however when making the y-axis log the graph takes on some very interesting properties. Both graphs are in this document for reference.

RESULTS

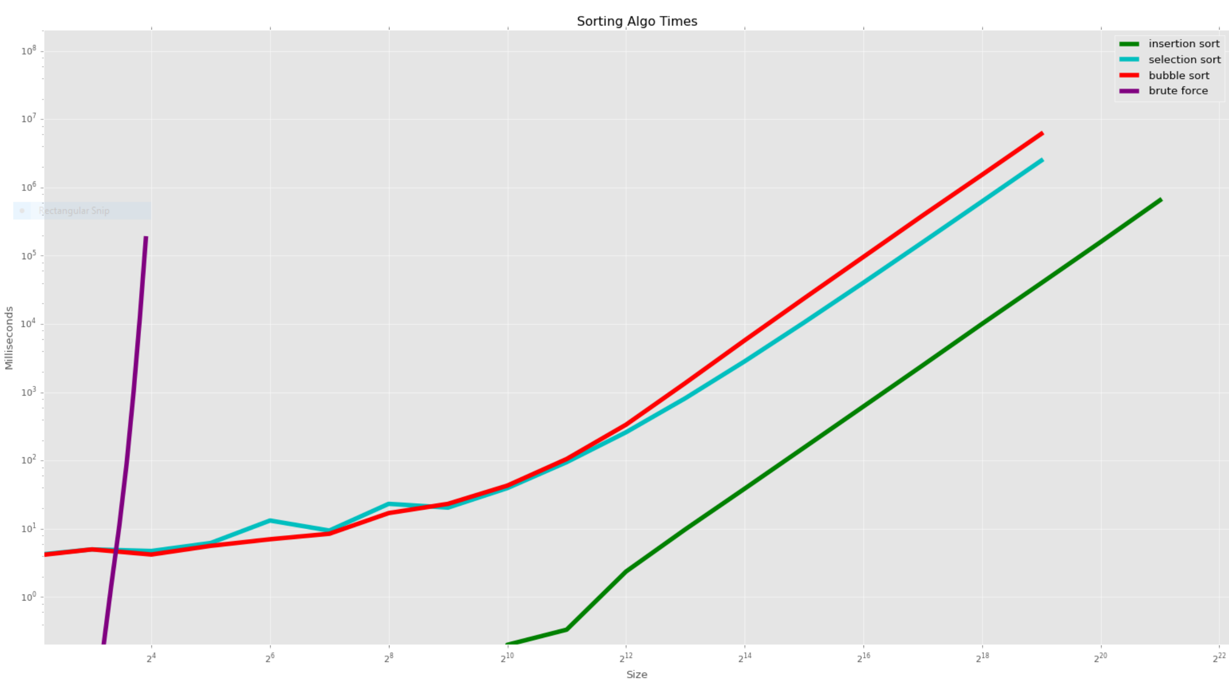
Of the four sorting algorithms one stood out as a top performer and one stood out as a horrible algorithm. To start with the worst, brute force sort could not process any larger than 15 elements without proceeding into hours of calculation. I think it is safe to say that at 20 or so elements brute force will take weeks to crack, and anything over 30 will take years. The best performer was insertion sort. Insertion sort reached its approximate 5 minute max at around 2 million. Bubble sort and selection sort were next on the performance. However they were only able to process around 500,000 elements in total and only around 260,000 in the same time that insertion sort could process 2 million.

EXPECTED RESULTS

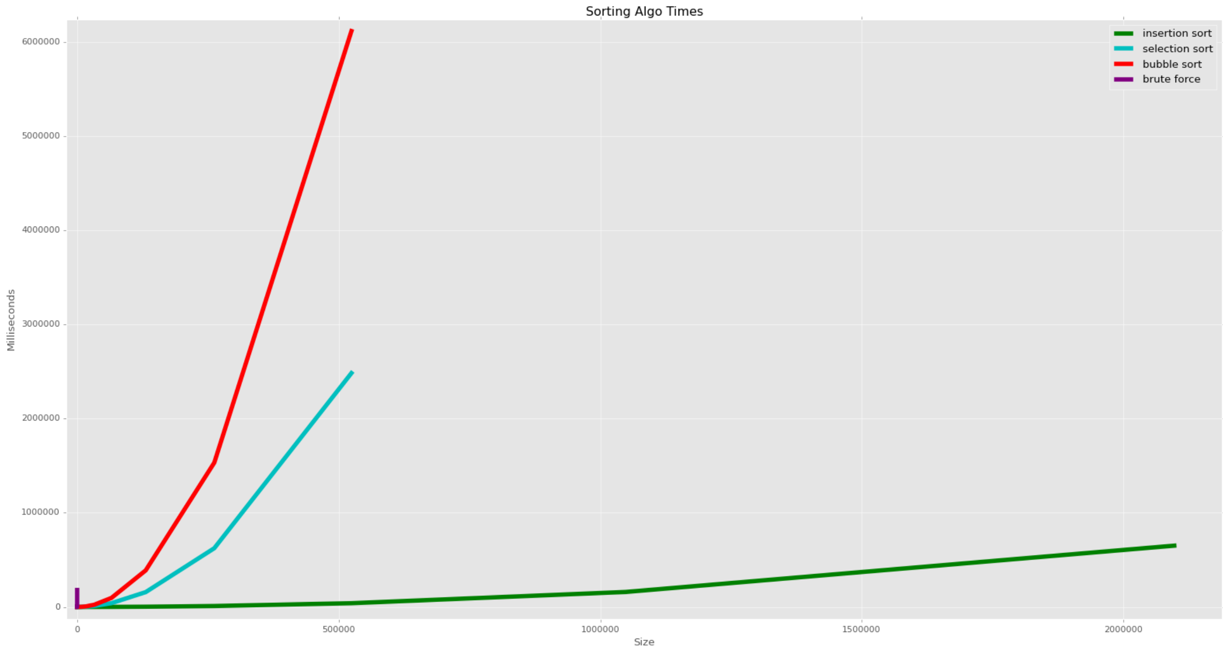
The results were a little surprising between insertion, selection, and bubble sort. There is drastic performance ability in the three considering they are all three in big-oh of n^2. Brute force did not have a surprising outcome, permutations runs in the n! complexity. It was shocking however to note that my gaming rig (gen4 i7 processor, liquid cooled, 16gb of 19000 MHz ram, and a very nice ASUS sabertooth mobo) could only process 15 elements in permutations. For a computer this is a super small number. I would have thought for sure to get into the hundreds or thousands before the results were perceptible to humans.

GRAPHS

log-log scale graph



linear scale graph



As one can see from the graphs, brute force algorithm has barely made it on the chart. The log scale graph shows how the other three sorting algorithms are similar in their performance at an asymptotic scale. However, the linear graph shows how much better insertion sort performs in practice.

STANDARD DEVIATION

Insertion sort

Array Size	Std Dev
2	0
...	...
1024	0.4
2048	0.471405
4096	0.481894
8192	0.453382
16384	0.422953
32768	0.657436
65536	1.78294
131072	5.16839
262144	196.276
524288	597.926
1048576	1221.45
2097152	16927

Selection Sort

Array Size	Std Dev
2	0
4	0.3
8	0
16	0.679869
32	0.6
64	2.04423
128	2.26102
256	7.06486
512	8.71295
1024	19.6532
2048	50.0156
4096	142.373
8192	454.408
16384	1584.21
32768	5866.99
65536	22487.7
131072	88050.2
262144	348503
524288	1387090

Brute Force

Array Size	Std Dev
2	0
4	0
8	0
9	0.718022
10	7.71874
11	59.4163
12	503.692
13	5466.12
14	69009.5
15	966626

Bubble Sort

Array Size	Std Dev
2	0
4	0
8	0
16	0.4
32	0.481894
64	1.11006
128	2.30555
256	7.89515
512	9.68051
1024	21.3026
2048	56.2754
4096	184.566
8192	760.595
16384	3217.92
32768	13245.4
65536	53599.9
131072	216982
262144	855742
524288	3415590