ECE345S 2022

Programming Assignment 1 Report

Group Members: Hetav Pandya, Chirag Sethi

a) Analysis of three sorting algorithms

Merge Sort: For merge sort, the runtime complexity we got was of the order of O(nlogn). We used python to generate plots of input size vs time taken for merge sort. We then tried to bound the graph of the runtimes with a function of nlogn. After experimentation, the graph was bounded by (0.00001)(nlogn). The constant 0.00001 affects the performance of the algorithm. The following is the graph. Y axis is time in seconds and X axis is input array size.

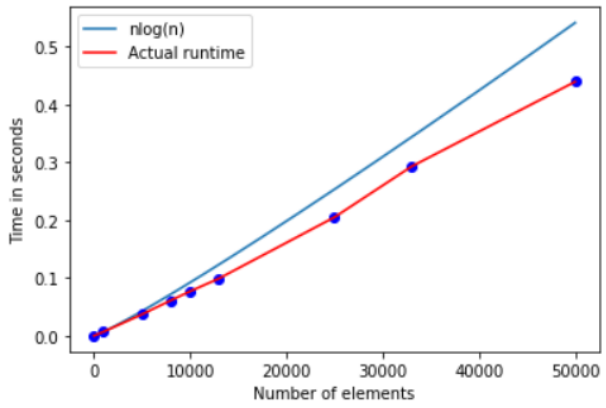

Fig 1.0: Merge sort time analysis

Insertion Sort: For insertion sort, the runtime complexity was obtained to be of the order of O(n^2). We used python to generate plots of input size vs time taken for insertion sort. We then tried to bound the graph of the runtimes with a function of n^2. After experimentation, the graph was bounded by (0.00000015)(n^2). The constant 0.00000015 affects the performance of the algorithm. The following is the graph. Y axis is time in seconds and X axis is input array size.

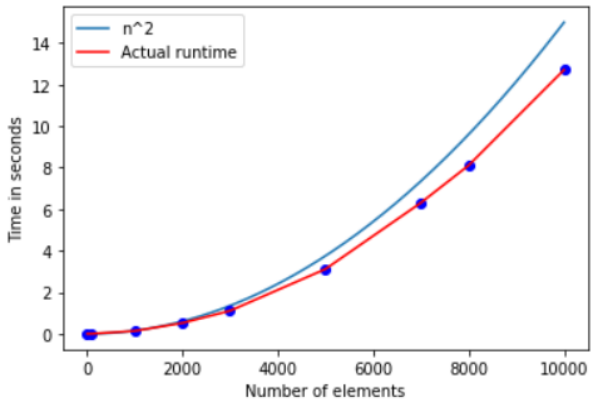

Fig 2.0: Insertion sort time analysis

Bubble Sort: For bubble sort, the runtime complexity we got was of the order of O(n^2). We used Python to generate plots of input size vs time taken for bubble sort. We then tried to bound the graph of the runtimes with a function of n^2. After experimentation, the graph was bounded by (0.0000006)(n^2). The constant 0.0000006 affects the performance of the algorithm. The following is the graph. Y axis is time in seconds and X axis is input array size.

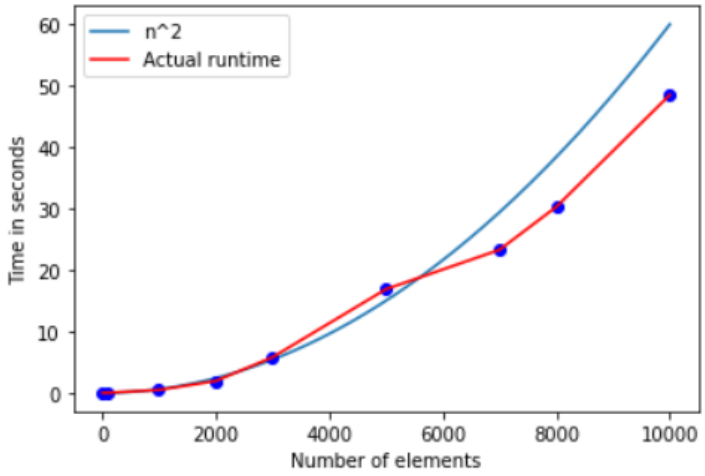

Fig 3.0: Bubble sort time analysis

b) Finding implementation constant

The following constants were derived using trial and error method such that the several graphs were plotted with different values and the constant that gave the tightest bound was selected.

- Graph for merge sort is shown in Fig. 1.0. The blue line is 0.00001 * nlogn where n is the input size. The blue line bounds the red line which is the performance graph of the algorithm.

- Graph for insertion sort is shown in Fig. 2.0. The blue line is (0.00000015)(n^2) where n is the input size. The blue line bounds the red line which is the performance graph of the algorithm.

- Graph for bubble sort is shown in Fig. 3.0. The blue line is (0.0000006)(n^2) where n is the input size. The blue line bounds the red line which is the performance graph of the algorithm.

c) Table describing growth functions and constants

Algorithm	Growth Function	Constant

Insertion Sort	$O(n^2)$	0.00000015
Bubble Sort	$O(n^2)$	0.0000006
Merge Sort	O(nlogn)	0.00001

d) Optimizing Bubble sort

We optimized the bubble sort algorithm. Bubble sort is an algorithm that runs in $O(n^2)$ time since it has a loop inside a loop. In our initial algorithm, we used to run the inside loop on the entire array in each iteration of the outer loop. From observation, we realized that in bubble sort, after k iterations of the outer loop, the last k elements of the array are in the correct positions. Hence, we can optimize by not doing comparisons on the last k elements for every iteration of the outer loop. This is done by reducing the range of the inner loop, so that last k elements are not accessed in the k+1th iteration of the outer loop.

The optimization reduced the constant from 0.0000006 to 0.0000004 which is about a 33% reduction in runtime. In terms of seconds, the runtime for the case of 10000 array elements reduced from 49 seconds to about 32 seconds.

Algorithm	**Growth Function**	**Constant**
Insertion Sort	$O(n^2)$	0.00000015
Bubble Sort	$O(n^2)$	0.0000006
Merge Sort	O(nlogn)	0.00001
Bubble Sort (optimized)	$O(n^2)$	0.0000004

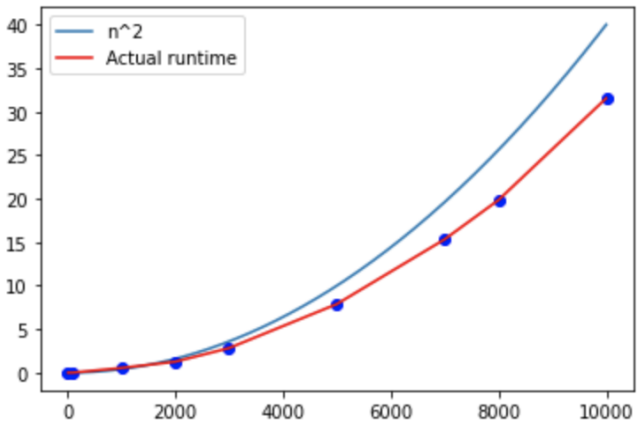

Fig 4.0: Optimized bubble sort time analysis

Performance graph of bubble sort after optimizations. The blue line is $0.0000004n^2$. The blue line bounds the red line which is the performance graph of the algorithm.

e) Comparing performance

The fastest algorithm of the four implemented was the merge sort algorithm. It had a growth function of the order of O(nlogn) as expected. The runtime for a 10000 element array was 0.1 seconds (as seen from the graph)

The remaining algorithms were all of the order of O(n^2). The best of the remaining three was insertion sort. It had the smallest constant term (0.00001). For reference its runtime for a 10000 element array was about 12 seconds.

The algorithm that came in third was the optimized bubble sort. Its constant was 0.0000004. Its runtime for the 10000 element array was about 32 seconds.

The slowest algorithm was the original bubble sort. It had a constant of 0.0000006. Its runtime for the 100000 element array was about 49 seconds.

f) Sources used

[1] Python and Google Collab to code our algorithms.
[2] The graphs were obtained using the matplotlib library in Python.
[3] We also referred to online resources to know more about sorting algorithms different from the ones taught in the lectures (https://www.freecodecamp.org/news/sorting-algorithms-explained/)

Detailed discussion about the optimization is already covered in part d)