CSC 172 Lab

Lab 5 Report

Contact

Uzair Tahamid Siam

email: usiam@u.rochester.edu

URID: 31434546 / NETID: usiam

Partner; Abrar Rahman Protyasha

Lab session - M/W 6:15 - 7:30

Lab Synopsis & Methods

For this lab we used the following sorting algorithms – default, bubble, selection, insertion, merge and quick sorts – on 4 different integer arrays ‘a’, ‘b’, ‘c’ and ‘d.’ Array ‘a’ was the original file we were given, ‘b’ was the sorted array, ‘c’ was the reverse sorted so worst case, and ‘d’ was a randomly sorted array we generated from the original using random swaps on the elements in the array. The findings of the lab are found below.

Note: All code for the sorting were from either zybook (readings) or lecture

Runtime Summary

Tables:

All runtimes are in milliseconds

Array size (thousands)	1	2	4	8	16	32	1000
Sort method							
Bubble_Sort a	8	8	45	101	366	1528	1825971
Insertion_Sort a	2	6	18	34	89	297	332285
Java_Default a	0	0	0	1	2	4	71
Mergesort a	1	1	2	3	5	6	188
Quicksort a	0	1	1	2	3	5	99
Selection_Sort a	2	7	25	50	129	258	293188

All sorting algorithm runtimes for array ‘a’

Array size (thousands)	1	2	4	8	16	32	1000
Sort method							
Bubble_Sort b	0	1	7	19	66	242	293336
Insertion_Sort b	0	0	0	0	0	0	8
Java_Default b	0	0	0	0	0	2	4
Mergesort b	1	0	0	2	4	6	89
Quicksort b	0	0	0	1	1	1	19
Selection_Sort b	0	3	6	20	73	249	296809

All sorting algorithm runtimes for array 'b'

Array size (thousands)	1	2	4	8	16	32	1000
Sort method							
Bubble_Sort c	3	2	6	29	86	348	402454
Insertion_Sort c	0	4	14	55	180	768	905395
Java_Default c	0	1	1	0	0	1	17
Mergesort c	0	1	1	1	2	7	71
Quicksort c	0	0	0	0	1	0	18
Selection_Sort c	2	2	8	30	99	348	524128

All sorting algorithm runtimes for array 'c'

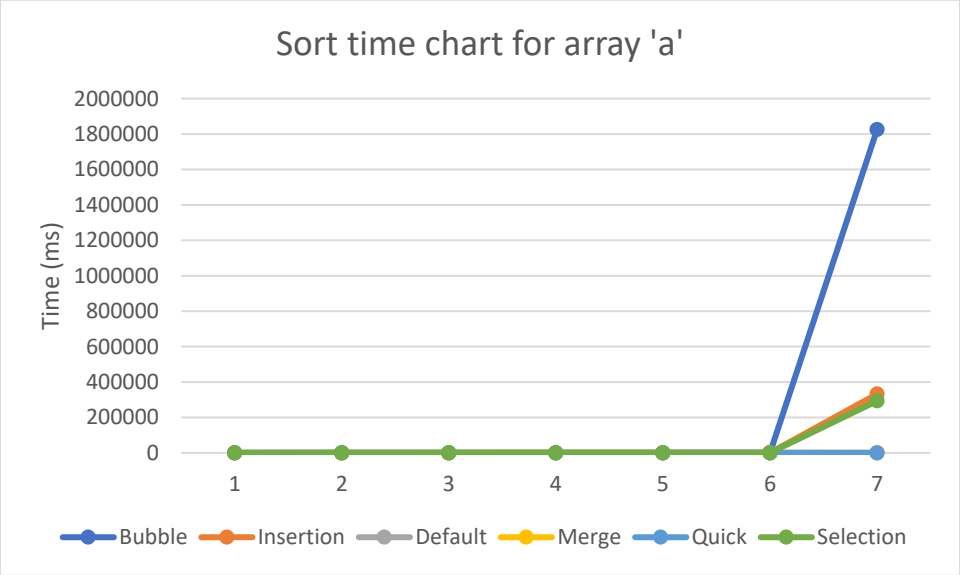
Array size (thousands)	1	2	4	8	16	32	1000
Sort method							
Bubble_Sort d	0	2	5	23	114	553	678246
Insertion_Sort d	0	0	0	1	7	52	39764
Java_Default d	0	0	0	1	2	3	37
Mergesort d	0	0	1	1	4	6	102
Quicksort d	0	0	0	1	1	1	40
Selection_Sort d	1	1	5	28	73	248	321322

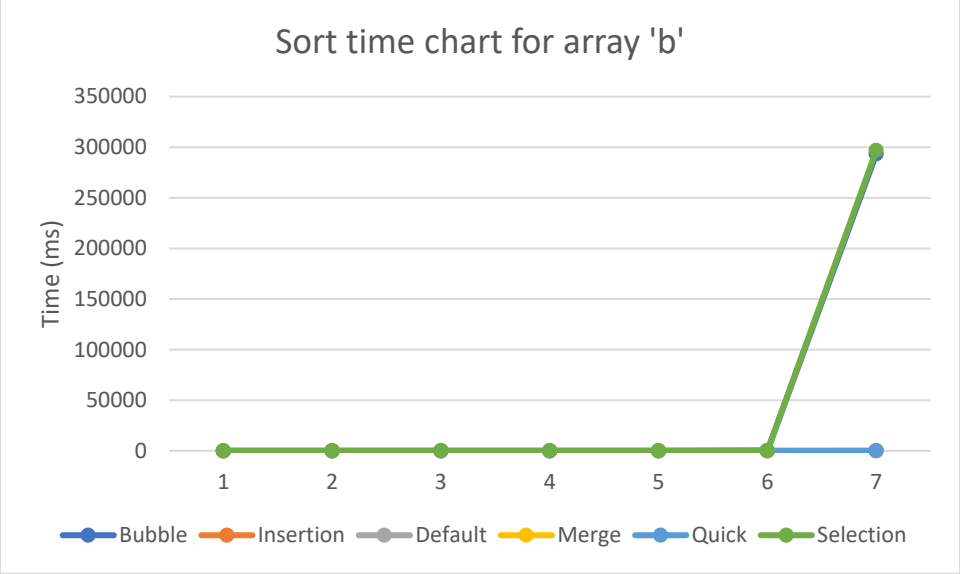
All sorting algorithm runtimes for array 'd'

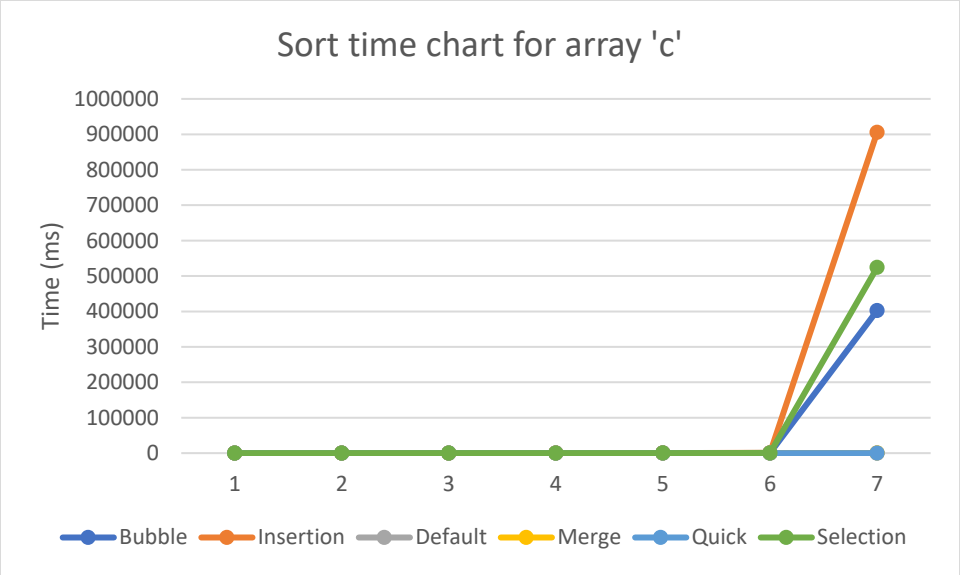
Note: Bubble sort was the slowest every time as expected, and default sort was the fastest.

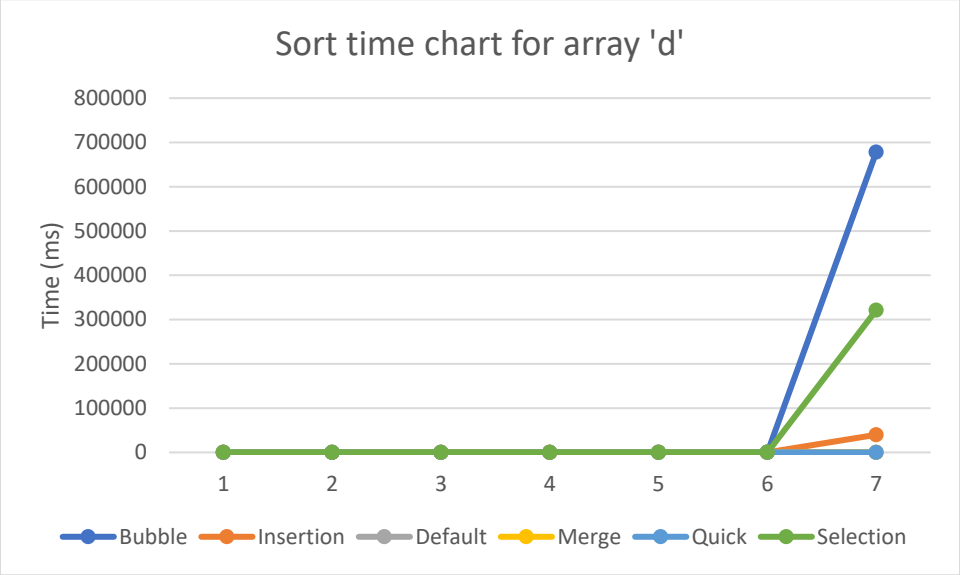
Plots:

The following plots visualize the runtime differences between each sorting algorithm acted on the 4 different arrays.






Sort time chart for array 'c'
Time (ms)
1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
1
2
3
4
5
6
7
Bubble
Insertion
Default
Merge
Quick
Selection


Sort time chart for array 'd'
Time (ms)
800000
700000
600000
500000
400000
300000
200000
100000
0
1
2
3
4
5
6
7
Bubble
Insertion
Default
Merge
Quick
Selection