Assignment 2 - Sorting Algorithms

Due: Thurs, Feb 6, 2025 @ 9:59 pm

Jayden Cruz

1. txt [2pts] Merge Sort [1,16,25,31] & [-3,0,16,27] - already sorted

Sorted Array: [-3,0,1,16,16,25,27,31]

Steps:

1. compare 1 < -3 = [-3]
2. 0 < 1 = [-3,0]
3. 1 < 16 = [-3,0,1]
4. 16 = 16 = [-3,0,1,16]
5. 25 > 16 = [-3,0,1,16,16]
6. 25 < 27 = [-3,0,1,16,16,25]
7. 27 < 31 = [-3,0,1,16,16,25,27]
8. 31 nothing to compare with

Sorted array = [-3,0,1,1,16,16,25,27,31]

2. txt [2pts] Insertion Sort: [-1,-5,67,-10,21,8,4,1]

insert

-10 < -5 insert there

-1 < 21 < 67

[-10,-5,-1,67,21,8,4,1] → [-10,-5,-1,21,67,8,4,1] → [-10,-5,-1,8,21,67,4,1] → [-10,-5,-1,4,8,21,67,1] → [-10,-5,-1,1,4,8,21,67]

-1 < 8 < 21 < 67

-1 < 4 < 8 < 21 < 67

-1 < 1 < 4 < 8 < 21 < 67

SORTED!

[-1,-5,67,-10,21,8,4,1]

↑ is -5 < -1? yes so we swap them

[-5,-1,67,-10,21,8,4,1]

↑ is 67 < -1 or < -5 no so we stay

[-5,-1,67,-10,21,8,4,1]

↑ is -10 < 67 yes so we swap

-5 -1 -10 67,21,8,4,1

↑ is -10 < -1? yes so we swap

-5,-10,-1,67,21,8,4,1

↑ is -10 < -5 yes so we swap

[-10,-5,-1,67,21,8,4,1]

↑ 21 < 67 so we swap not < -1 tho.

[-10,-5,-1,21,67,8,4,1]

↑

3. txt [2pts] QuickSort: [-5,42,6,19,11,25,26,-3]

let the pivot = 11 (p)

Random Selection

A = [-5,42,6,19,11,25,26,-3]

- everything < 11 goes to the left
- everything > 11 goes to the right. ? go through array & compare to pivot to do so.

[S1 = -5,6,-3] P = 11 [S2 = 42,19,25,26]

- < 25 go left
- > 25 go right

p = 25

[S2 = 42,26]

P = 26 S2 = 42

[26,42]

Quicksort

S1 = -5,-3 P = 6

- < 6 go left
- > 6 go right

P = -3 : < L : > R

S1 = -5

[-5,-3,6]

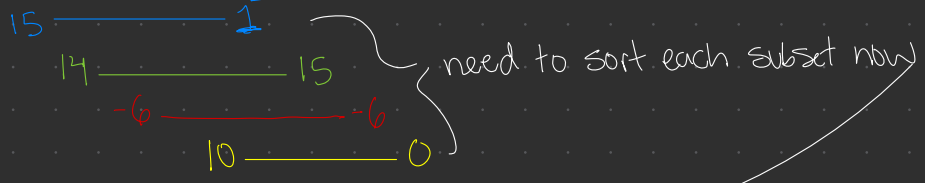
[S1 = 19]

P = 11

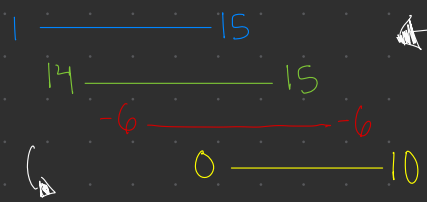
19,25,26,42

[-5,-3,6,11,19,25,26,42]

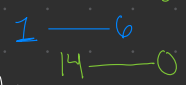
4. txt [2 points] Shell Sort: [15, 14, -6, 10, 1, 15, -6, 0] h=4 bc elements in array/2 = 8/2 = 4

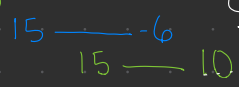


need to sort each subset now

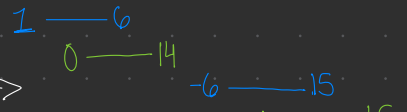


[1, 14, -6, 0, 15, 15, -6, 10] now h=2





need to sort now →



Perform insertion sort on this

[1, 0, 6, 14, -6, 10, 15, 15]

↑ is 0<1 yes so swap

[0, 1, 6, 14, -6, 10, 15, 15]

↑ 6<1? No so stay. or <0 no

[0, 1, 6, 14, -6, 10, 15, 15]

↑ 14<6? No so stay nor is it <1 or 0

[0, 1, 6, 14, -6, 10, 15, 15]

↑ -6<14 yes so swap

0 1 6 -6 14 10, 15, 15

↑ -6<6 yes so swap

0, 1, -6, 6, 14, 10, 15, 15

↑ -6<1 yes so swap

0 -6 1 6 14 10 15 15

↑ -6 < 0? yes so swap

[-6 0 1 6 14 10 15 15]

↑ 10<14? yes so swap

-6 0 1 6 10 14 15 15

↑ 10< 6, 1, 0, -6 no so stay

[-6, 0, 1, 6, 10, 14, 15, 15]

↑ compare to all elements to the left 15<14, 10, 6, 0, -6? No so it stays.

[-6, 0, 1, 6, 10, 14, 15, 15]

↑ compare to all elements to the left 15<15, 14, 10, 6, 0, -6? No so it stays.

[-6, 0, 1, 6, 10, 14, 15, 15] SORTED!

5.(text)[4 pts] Rank the 6 sorting algos

My Ranking:

1. Mergesort: O(n log n)
2. Quicksort: O(n log n)
3. Shellsort: O(n log² n)
4. Insertion Sort: O(n²)
5. Selection Sort: O(n²)
6. Bubble Sort: O(n²)

- mergesort & quicksort have the most efficient time complexity w O(n log n)
- shellsort similar to but performance depends on gap worst = O(n²) but usually better
- insertion, selection, bubble all have quadratic time complexities, could say tie as their best, avg, worst case are the same

Time Complexities of Sorting Algorithms

Algorithm	Best Case	Average Case	Worst Case	Space Complexity
Bubble Sort	O(n)	O(n²)	O(n²)	O(1)
Insertion Sort	O(n)	O(n²)	O(n²)	O(1)
Selection Sort	O(n²)	O(n²)	O(n²)	O(1)
Shell Sort	O(n log n)	O(n log² n)	O(n²)	O(1)
Quick Sort	O(n log n)	O(n log n)	O(n²)	O(log n)
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)

from asn_2 github readME

9.txt[7 points]

Made the graph in Java, a solution to the huge numbers for (ms) could be to log it. Did this in stat modeling & Bayesian graph shown in 17.

10.txt[7 points]

1. Bubble sort performed the worst, with the time increasing alot as the input grew. Which makes sense as it time Complexity is O(n²) took about 15,915.45 for 150k elements

2. Insertion perform better than Bubble Sort, but still was slow w/ larger input. It was faster for the smaller datasets. Slowed down @ 70k

3. Selection, similar to Insertion, but was sometime higher ms time as input grew. Similar on smaller input but slower on larger than Insertion

4. Shell Sort was always faster than the prev. 3 algorithms. Did well on larger input 150k in 13.35ms (!!) So way better than Insertion, Selection, Bubble

5. QuickSort did better than all the previous. Its time was always way lower than the other. Did 150k in 6.35ms.

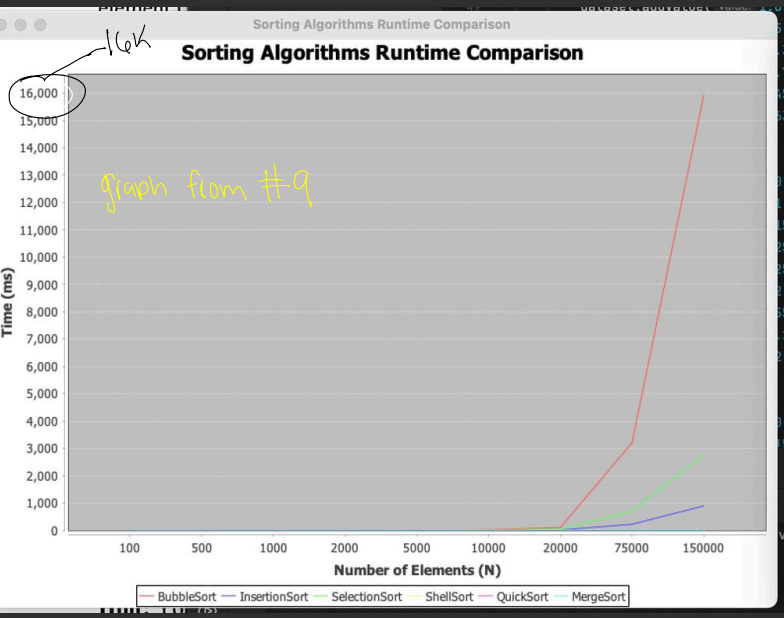
6. MergeSort did well but a little slower than Quick. Time increased slightly as the input did.

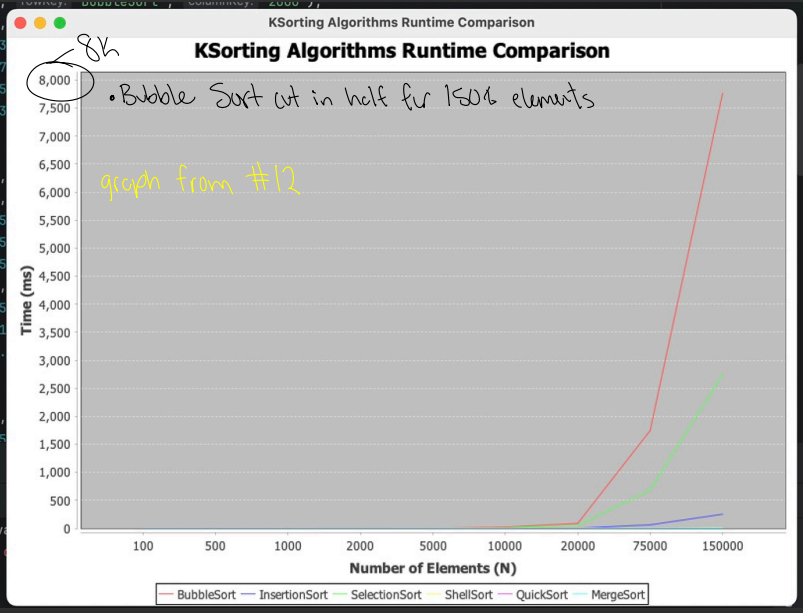
The experiment does align with the algorithms asymptotic analysis—time complexities

- Bubble, Insertion, & Selection have Quadratic time complexities O(n²), matching how slow they preformed for larger inputs.
- Shell sort did better than the 3 quadratic algorithms, its time complexity is closed to O(n log n) best case, but its time complexity can vary
- Quick & Merge Sort are O(nlogn) & this shows in how they performed. QuickSort was fastest in most cases

In Conclusion, these performance matches/confirms my predicted rankings based on their time complexity.

12.K sorted graph made in Java





BubbleSort: 10 sorted: did better, took 0.0 ms for 100 elements slowed down for larger but was faster than random cut time in half for 150K elements compared to random.

Insertion: 10 sorted: was faster, times close to 0ms for smaller inputs slower on random but always better than bubble sort.

Selection: slight improvements, smaller runtimes on small inputs compared to random. On random

Shellsort: was close to 0ms up to 10K elements but 17.15ms on 150K elements.

Quicksort: similar on 10 sorted almost instant results

Merge: was almost instant, only 9.55ms for 150K

did similar on both types of data.

•Bubble sort had the biggest difference its much faster on 10-sorted, this is bc it does fewer swaps when input is already sorted or nearly sorted
•Insertion: also improved but not as much as Bubble.

The quadratic algorithms show the most improvements on 10 sorted. Quadratic ones benefit the most

Extra Credit: done in Java Class [Graph.java] & [KGraph.java]