Benevento

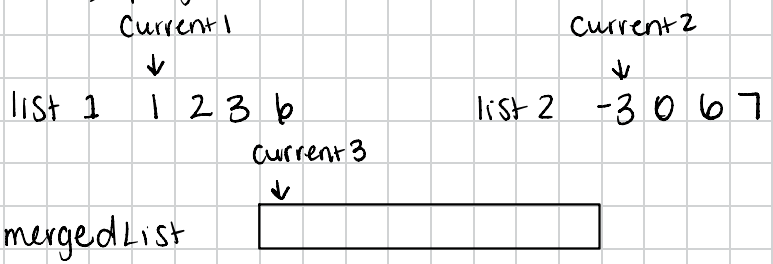
Assignment 2 - Sorting Algorithms

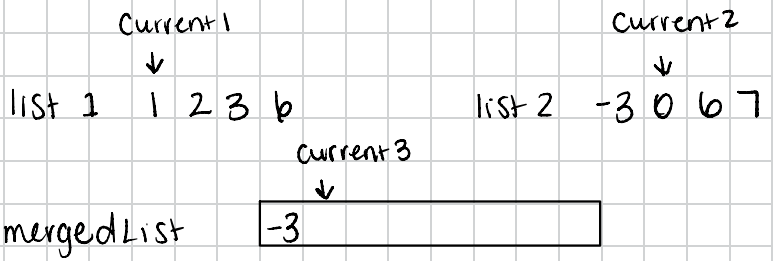
Problem 1 (merge sort)

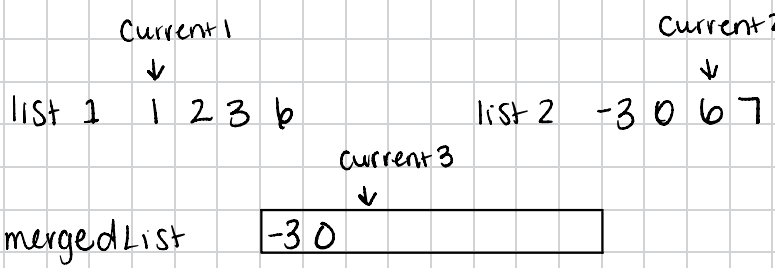
compare current 1 to current 2

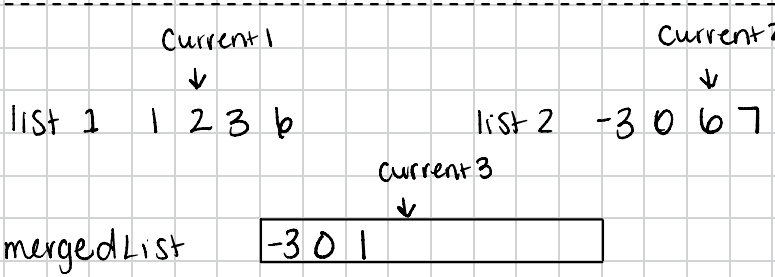
Store smaller value in current 3, then progress the pointer of the list with the smaller value to the next element.

Also, progress current 3





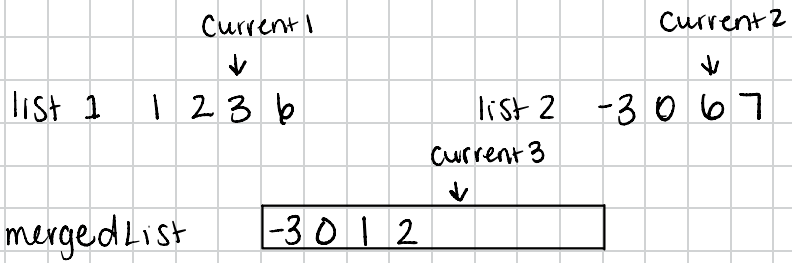


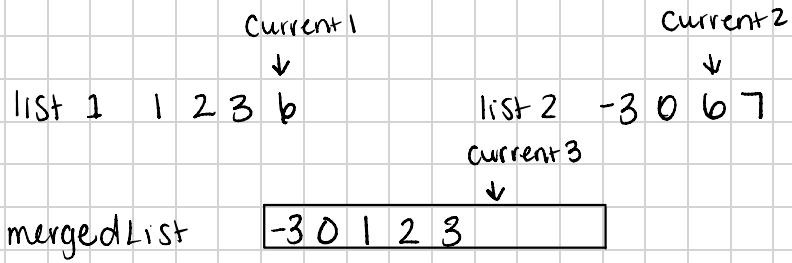


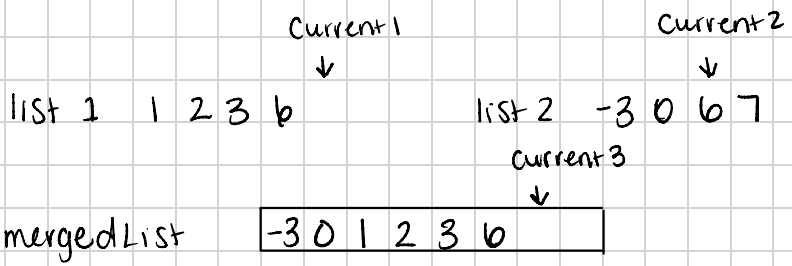
Benevento

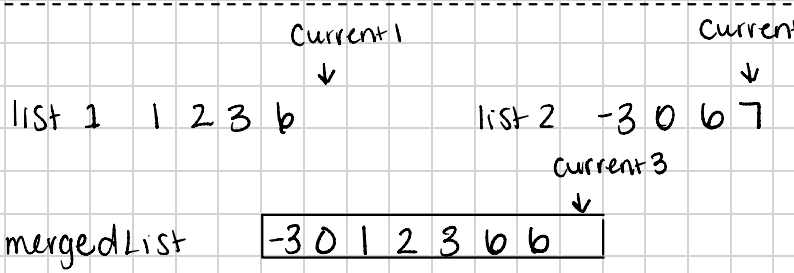
Assignment 2 - Sorting Algorithms

Problem 1 (merge sort)





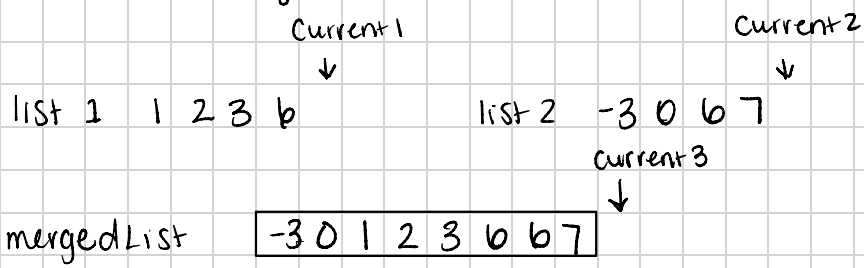




Benevento

Assignment 2 - Sorting Algorithms

Problem 1 (merge Sort)



Benevento
Assignment 2

Problem 2 (Insertion Sort)

Compare value at index 1 to index 0, swap if it is index 1 < index 0.
move to index 2, compare to index 1.
if index 2 < index 1, swap. Then compare the newly swapped index1 to index 0, if index 1 < 0, swap.
move to index 3 ...
Continue the process until the end of the list.

index	0	1	2	3	4	5	6	7
	-21	5	7	-10	61	8	3	10
P=1	-21	5	7	-10	61	8	3	10
P=2	-21	5	7	-10	61	8	3	10
P=3	-21	-10	5	7	61	8	3	10
P=4	-21	-10	5	7	61	8	3	10
P=5	-21	-10	5	7	8	61	3	10
P=6	-21	-10	3	5	7	8	61	10
P=7	-21	-10	3	5	7	8	10	61

Benevento
Assignment 2

Problem 3 (quick sort)

first, choose a pivot. Take the first, middle, and last element, and choose the median. In this case

-5 619 -3

-5 will be our pivot.

Next, partition the array using the pivot. Left of the pivot are values less than or equal to the pivot. Right of the pivot are values greater than the pivot.

$\frac{0+7}{2} = 3.5 = 3$

-5	4	2	619	11	5	620	-3

pivot = -3

i						j	
-5	4	2	619	11	5	620	-3
	i				j		
-5	4	2	619	11	5	620	-3
	i			j			
-5	4	2	619	11	5	620	-3
	i		j				
-5	4	2	619	11	5	620	-3
	i	j					
-5	4	2	619	11	5	620	-3
	i	j					
-5	4	2	619	11	5	620	-3

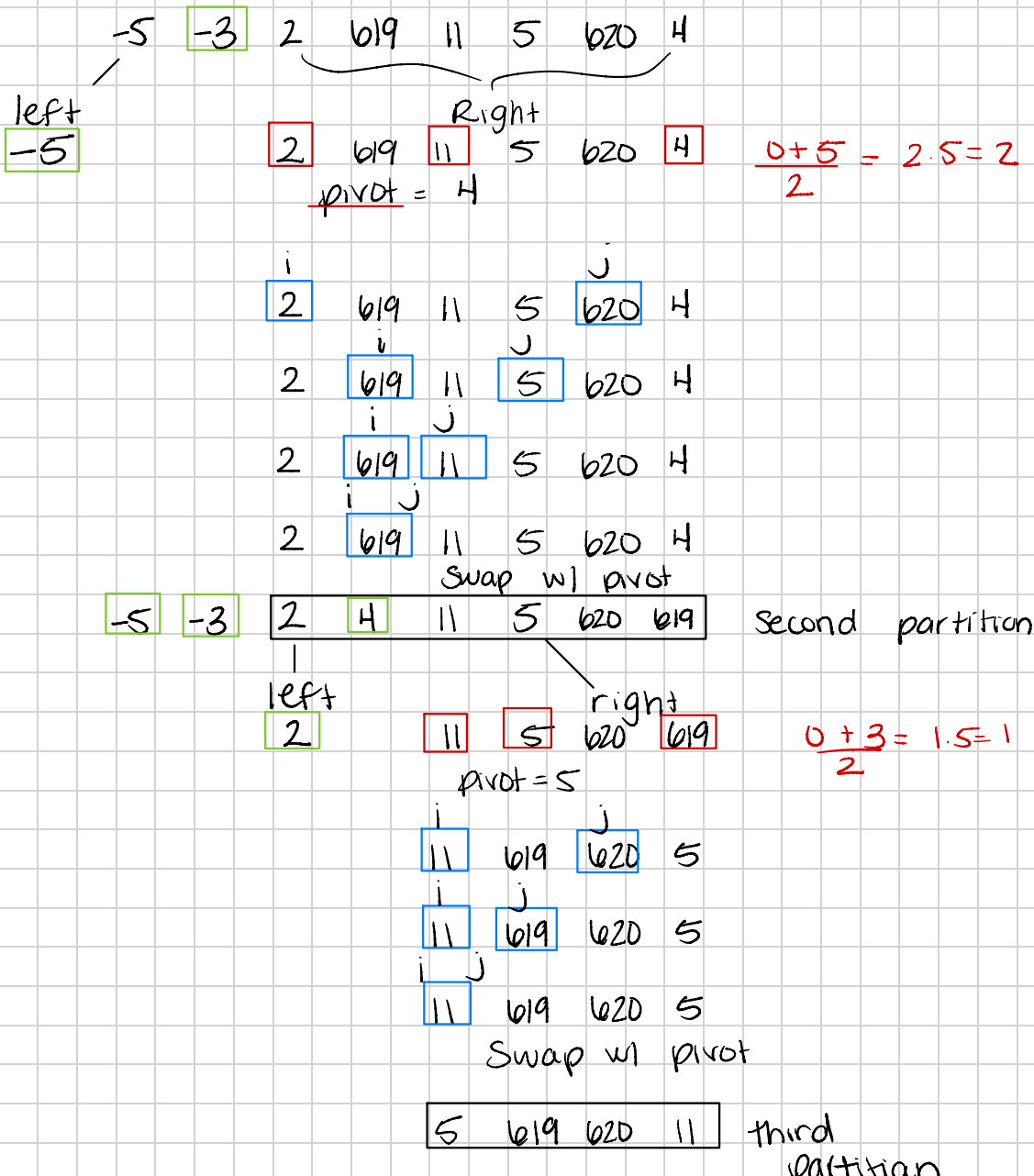
swap w/ pivot

-5	-3	2	619	11	5	620	4

first partition

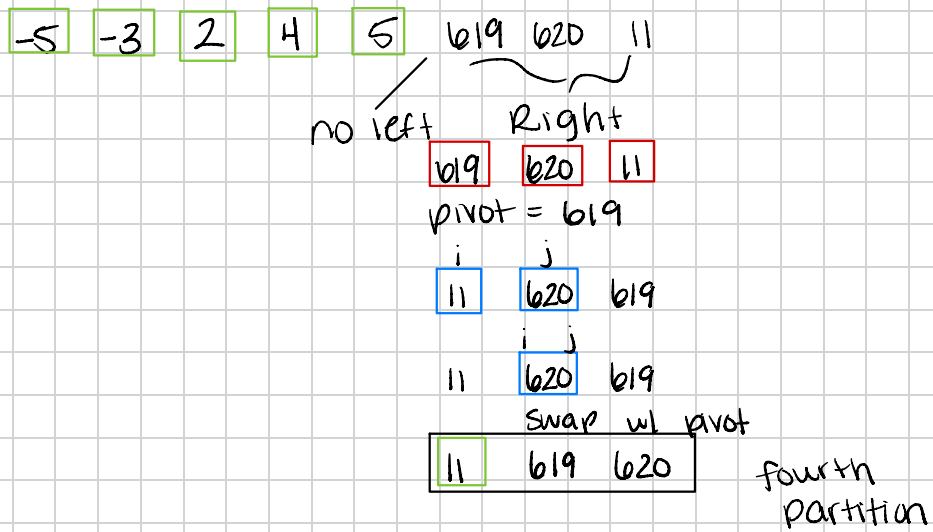
Benevento
Assignment 2

Problem 3: Quicksort



Benevento
Assignment 2

Problem 3: Quick Sort



Quick Sorted list

-5 -3 2 4 5 11 619 620

Benevento
Assignment 2

Problem 4: Shell Sort

n = list length = 8

gap = n/2 = 8/2 = 4

gap = 4

A = | 5 | 10 | 60 | 0 | -1 | 34 | 6 | 10 |

| 5 | -1 |

Sort Sublist 1

Swap

A = | -1 | 10 | 60 | 0 | 5 | 34 | 6 | 10 | After 1st iteration

| 10 | 34 |

Sort Sublist 2

A = | -1 | 10 | 60 | 0 | 5 | 34 | 6 | 10 | After 2nd iteration

| 60 | 6 |

Sort Sublist 3

Swap

A = | -1 | 10 | 6 | 0 | 5 | 34 | 60 | 10 | After 3rd iteration

| 0 | 10 |

Sort Sublist 4

A = -1 10 6 0 5 34 60 10 After 4th iteration

Benevento
Assignment 2

Problem 4 : Shell Sort

gap = 4/2 = 2

gap = 2 A = | -1 | 10 | 6 | 0 | 5 | 34 | 60 | 10 |

-1	6	5	60
-1	5	6	60

→ insertion sort

A = | -1 | 10 | 5 | 0 | 6 | 34 | 60 | 10 |

10	0	34	10
0	10	34	10
0	10	10	34
0	10	10	34

After 1st iteration

insertion sort

A = | -1 | 0 | 5 | 10 | 6 | 10 | 60 | 34 |

After 2nd iteration

gap = 2/2 = 1

A = | -1 | 0 | 5 | 10 | 6 | 10 | 60 | 34 |

use insertion sort for final list

Shell Sort finished list

A = | -1 | 0 | 5 | 6 | 10 | 10 | 34 | 60 |

Benevento
Assignment 2

Problem 5: Algorithm Rankings
(1 = fastest, 6 = slowest)

1. Quick Sort
2. Shell Sort
3. merge Sort
4. Insertion Sort
5. Bubble Sort
6. Selection Sort

} tie - can change order

Explanation:
Quick Sort is generally fastest because it splits the array into subarrays and then sorts them. merge Sort is similar in how it splits the array but requires extra memory for merging the sublists back together. Shell Sort compares and sorts gap values, moving them closer to their final positions and reducing swaps later on. Insertion, Bubble, and Selection Sort are the slowest and interchangeable with the same $O(n^2)$ time complexity because they typically need to make comparisons of all elements in the array.

Time Complexities on next page.

Benevento
Assignment 2

Problem 5: Algorithm Rankings

Time Complexity

Quick Sort has a worst case time complexity of $O(n^2)$ where n is the number of elements in the array if the pivot results in an empty subarray on either side.
The best case is $O(n \log n)$ because the pivot divides the list in equal halves.

merge Sort has a time complexity of $O(n \log n)$ where n is the number of elements in the array because it always divides the array into two halves.

Shell Sort has a worst case time complexity of $O(n^2)$ because it may need to be compared with all elements, even if they're already sorted.
It has a best case of $O(n \log^2 n)$ if the array is almost sorted because it can reduce the number of swaps.

Benevento
Assignment 2

Problem 5: Algorithm Rankings

Time Complexity

Insertion Sort has a time complexity of $O(n^2)$ where n is the number of elements in the array because it may need to compare and swap every element in the array.

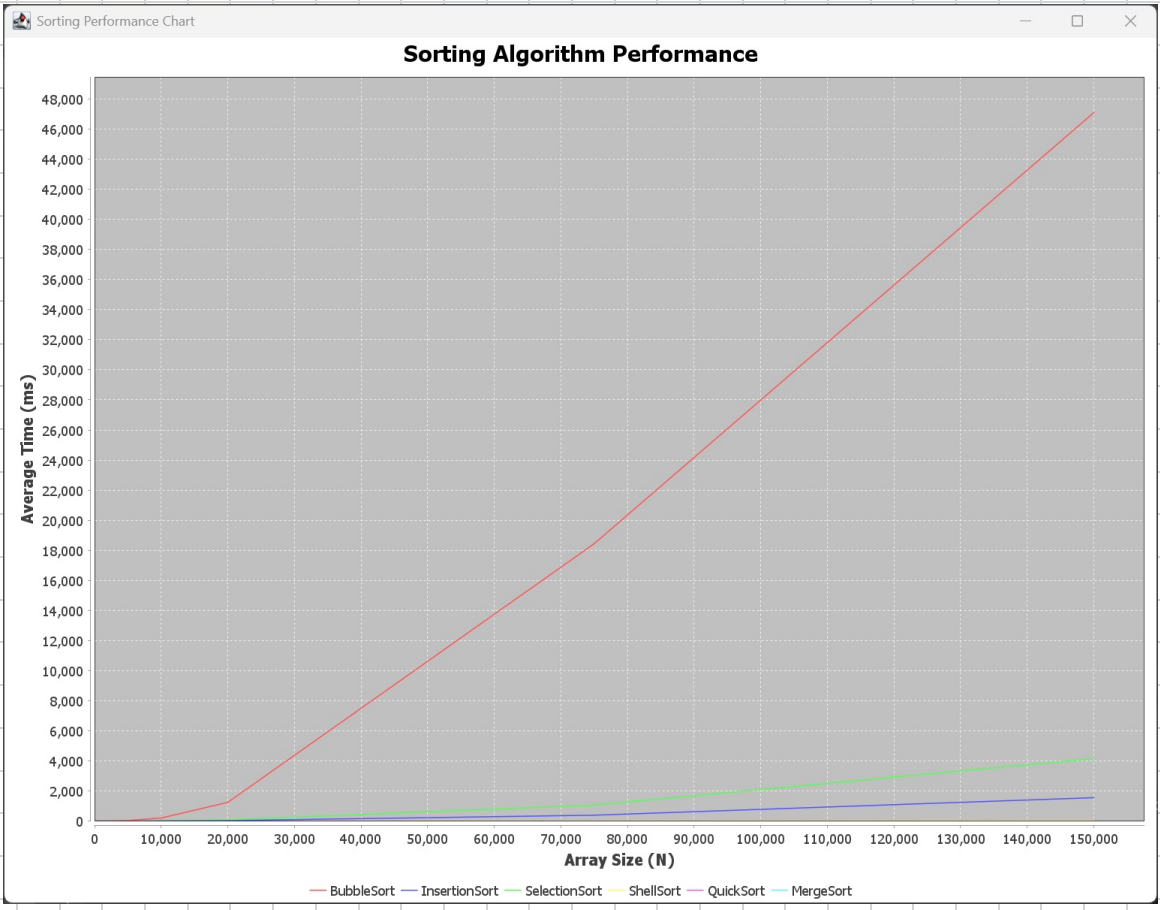
Bubble Sort has a time complexity of $O(n^2)$ where n is the number of elements in the array because it may need to compare and swap every element in the array.

Selection Sort has a time complexity of $O(n^2)$ where n is the number of elements in the array because it needs to compare each element in the array to find the smallest and move it to the beginning

Benevento
Assignment 2

Problem 9

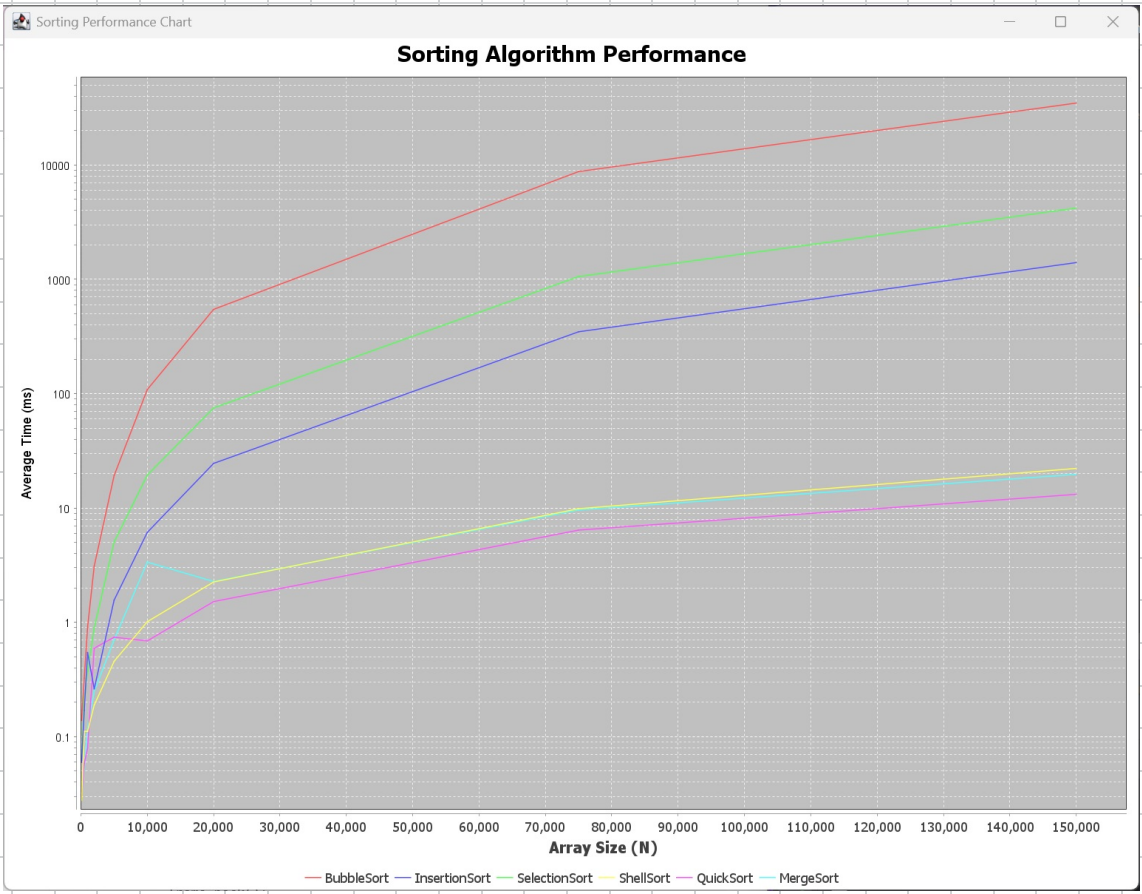
Original graph



Because Bubble, insertion, and selection sort have a quadratic time complexity, $O(n^2)$, the array size increases quicker than the other algorithms.

Benevento
Assignment 2

Problem 9:



Using a logarithmic y-axis compresses the larger numbers and helps all algorithms to be visible.

Benevento
Assignment 2

Problem 10:

Bubble Sort, Insertion Sort, and Selection Sort all proved to be the slowest, which I predicted. However, I predicted they would tie and that wasn't exactly right. Bubble sort took the most time, followed by selection, then insertion.

I was correct that Quick Sort would be the fastest. Shell Sort and merge Sort seem to have tied.

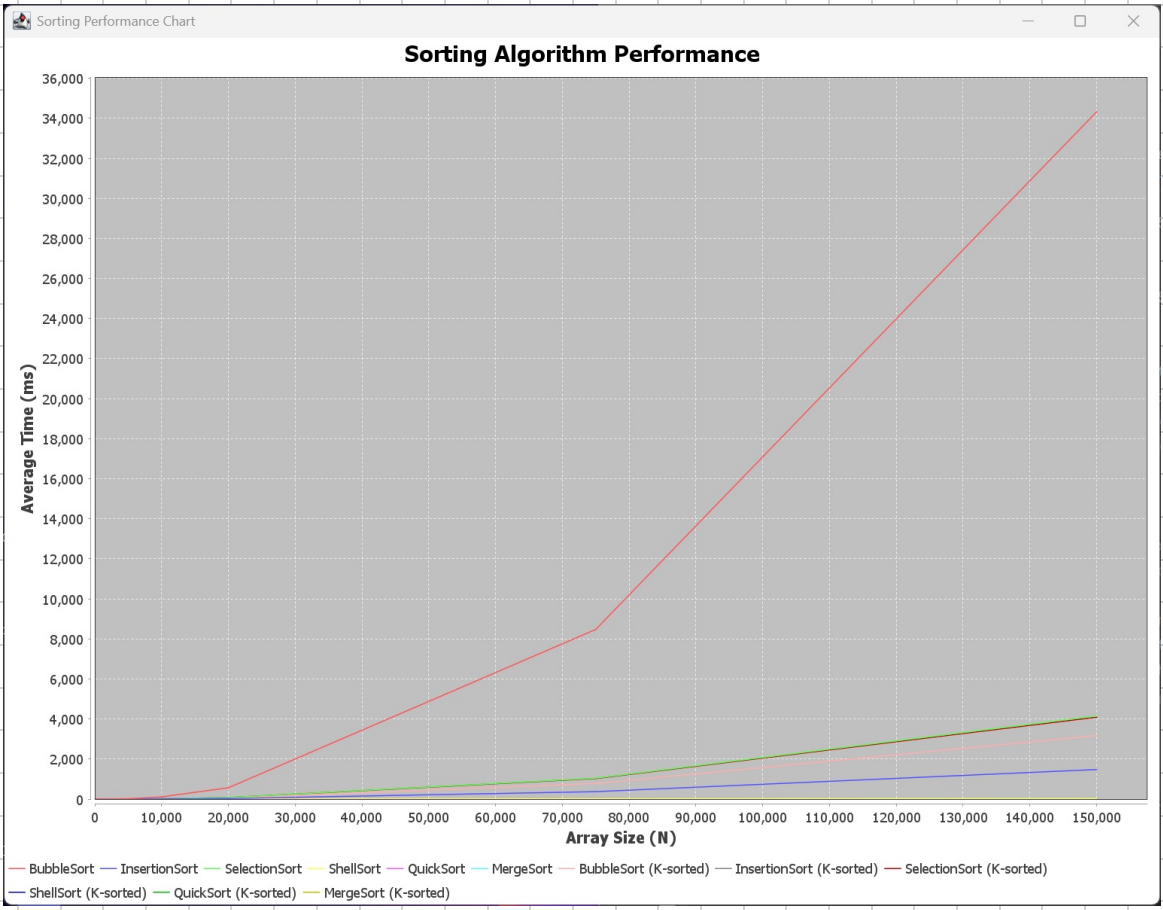
Problem 11: K-Sorted Data
10-Sorted Ranking

1. Insertion Sort
2. Bubble Sort
3. Quick Sort
4. Shell Sort
5. merge Sort
6. Selection Sort

Because Insertion Sort has a best case run time of O(n) if the list is already sorted, I would expect it to be the most optimal in K-sorted data. For the same reason, I would expect Bubble Sort to perform better. Then, I expect Quick Sort, Shell Sort, and merge Sort to perform about the same based on their best case run times O(n log n). At the end would still be Selection Sort because it always checks the entire array.

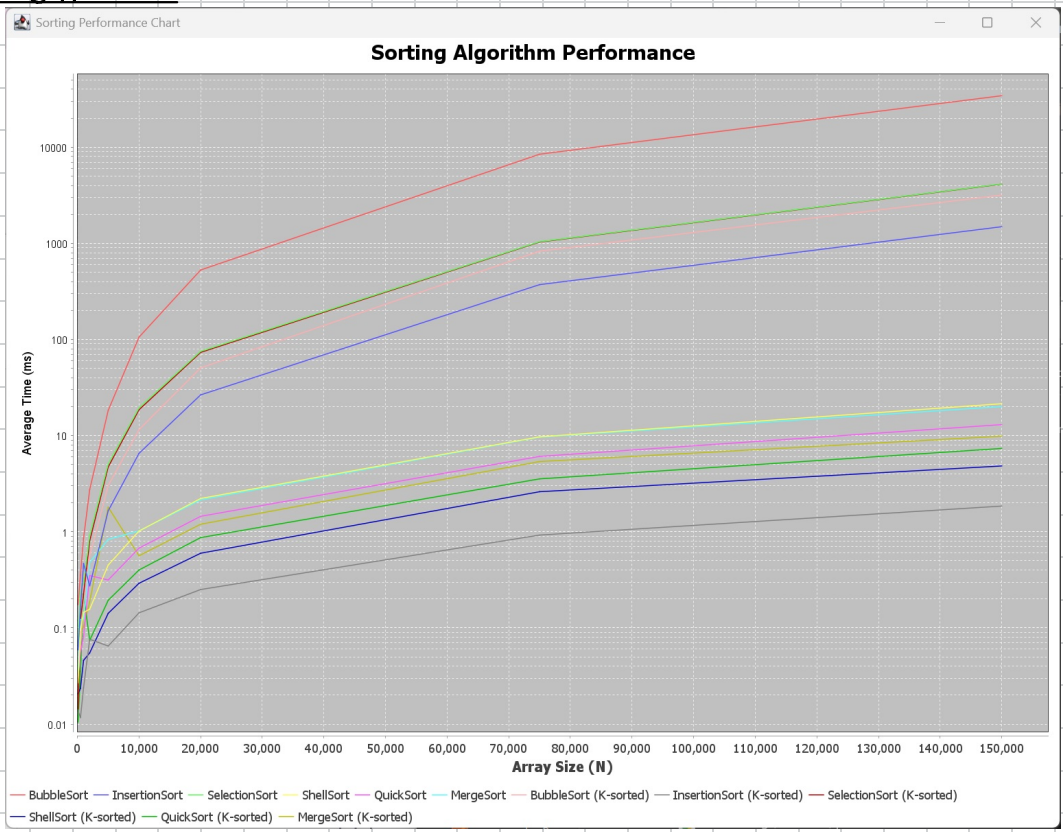
Benevento
Assignment 2

Problem 12



Because Selection Sort always runs with a complexity of $O(n^2)$ where n is the number of elements in the array because it always checks the entire array.

Benevento
Assignment 2
Problem 12



The logarithmic graph again helps the data to be more visible by compressing the y-axis to base 10. The algorithms did not have the same rankings. Insertion sort did prove to be the fastest out of the 10 sorted data which supports the best case time complexity of O(n). It was then followed by shell sort, which was a little surprising since it still has to check all the gaps. Quick sort was third followed by merge sort, which again makes sense, since they split the lists into sublists. I expected bubble sort to perform more like insertion sort since it has a time complexity of O(n) for its best case. Selection sort was the slowest as expected.