1) MergeSort:

[**1**, 2, 3, 6] and [**-3,** 0, 6, 7]

Compare the first index (1 vs -3), and append the lowest index to the sorted array.

Then move to the next index in the same array. -3 is first so move to the next index in the right array).

1 vs 0 append 0

1 vs 6 append 1

2 vs 6 append 6

3 vs 6 append 6

6 vs 6 append 6 (I chose to take the left one)

Empty vs 6 append 6

Empty vs 7 append 7

[-3,0,1,2,3,6,6,7] **sorted array**

2) InsertionSort:

[-21, 5, 7, -10, 61, 8, 3, 10]

Start with comparing –21 vs 5 because 5 is larger it stays

[-21| 5, 7, -10, 61, 8, 3, 10]

Same with 7 it stays

[-21, 5 | 7, -10, 61, 8, 3, 10]

Now –10 must shift down by comparing to 7,5, and -21

[-21, -10, 5, 7,| 61, 8, 3, 10]

61 is larger so it stays

[-21, -10, 5, 7, 61|, 8, 3, 10]

8 shifts down once because 61> 8

[-21, -10, 5, 7, 8, 61,| 3, 10]

3 < 61 shift to the right

3 < 8 shift to the right

3 < 7 shift to the right

3 < 5 shift to the right

3 > 10 stays here

[-21, -10, 3, 5, 7, 8, 61, |10]

10 < 61 shift to the right

10 >8 10 stays

[-21, -10, 3, 5, 7, 8, 10, 61] Sorted array

3) QuickSort:

[-5, 4, 2, 619, 11, 5, 620, -3]

Random pivot 11

s1[-5, 4,2,-3], pivot [11], s2 [619, 620]

Split s1 and s2

S1a [-5, -3, 2] s1pivot [4] | s2 [619, 620] “sorted”

Concatenate all parts

S1a + s1 pivot + pivot + s2

[-5, -3, 2], [4], [11], [619, 620]

[-5, -3, 2, 4, 11, 619, 620] **sorted**

4) Shell Sort:

[5, 10, 60, 0, -1, 34, 6, 10] unsorted (index 8, so use gaps 4,2, 1)

Gap 4:

[5,-1] -> [-1,5]

[10, 34] no change

[60, 6] -> [6, 60]

[0, 10] no change

After gap 4: [-1,10, 6, 0, 5, 34, 60, 10]

Gap 2:

(indexes 0,2,4,6): [-1, 6, 5, 60]

Sorted: [-1, 5, 6, 60]

(indexes 1,3,5,7): [10, 0, 34, 10]

Sorted: [0, 10, 10, 34]

After gap 2: [-1, 0, 5, 10, 6, 10, 60, 34]

Gap 1:

Do insertion sort over all indexes

[-1|, 0, 5, 10, 6, 10, 60, 34]

[-1, 0|, 5, 10, 6, 10, 60, 34]

[-1, 0, 5|, 10, 6, 10, 60, 34]

[-1, 0, 5, 10|, 6, 10, 60, 34]

[-1, 0, 5, 6, 10|, 10, 60, 34] swapped 6,10

[-1, 0, 5, 6, 10, 10|, 60, 34]

[-1, 0, 5, 6, 10, 10, 60|, 34]

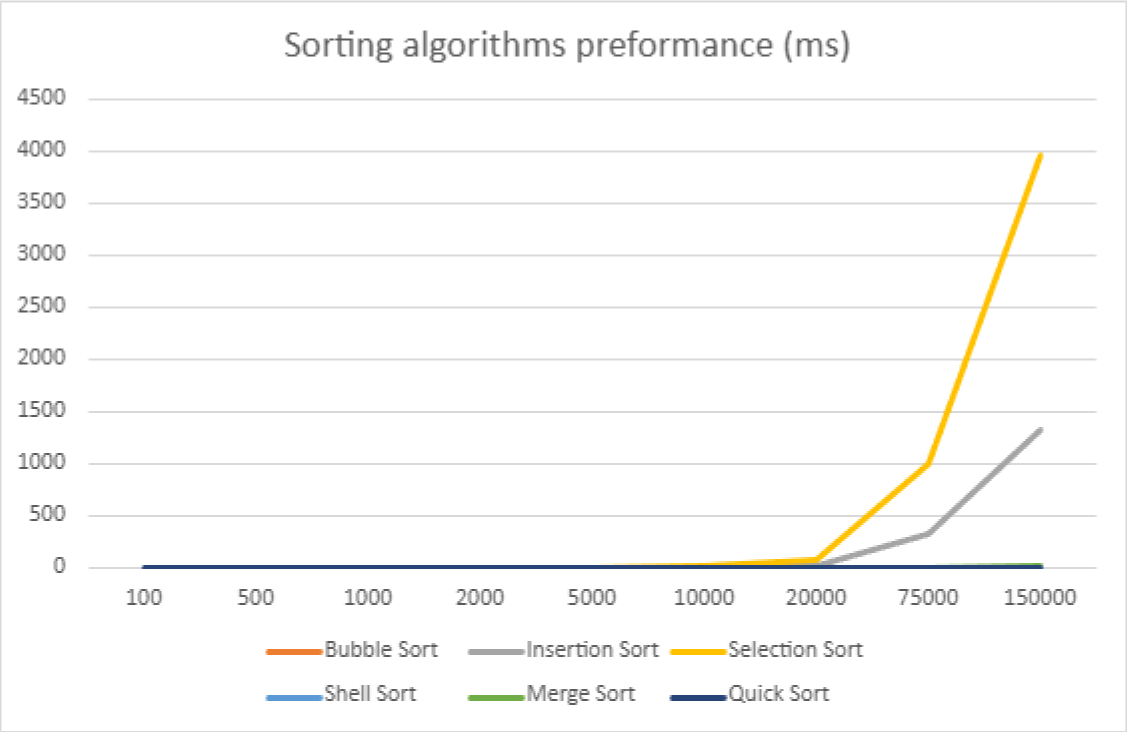
[-1, 0, 5, 6, 10, 10, 34, 60|] swapped and sorted

5) Predictions

1. Mergesort
2. Quicksort
3. Shell sort
4. Insertion sort & Bubble sort
5. Selection sort

Based off of best and worst cases of each sorting algorithm. Merge sort is always n log n, so its consistency leads it to be the fastest. Quicksort is also n log n but in its worst case its n^2 so i put it second. Shell sort also has the same best and worst case but it heavily depends on the gap sequence. Insertion and bubble both have the same best case (O(n)) and worst case (O(n^2)), so they're a tie. Selection sort is always n^2 so i put it last

9)



Input Size	Bubble Sort	Insertion Sort	Selection Sort	Shell Sort	Merge Sort	Quick Sort
100	0.002	0.052	0.065	0.023	0.026	0.023
500	0.01	0.14	0.175	0.087	0.041	0.034
1000	0.02	0.3	0.244	0.074	0.088	0.068
2000	0.037	0.284	0.781	0.157	0.185	0.096
5000	0.054	1.511	4.537	0.324	0.337	0.232
10000	0.012	5.986	19.22	0.718	0.695	0.503
20000	0.022	23.835	74.822	1.566	1.516	1.075
75000	0.061	327.372	1010.811	6.929	6.309	4.486
150000	0.122	1337.124	3975.664	15.243	13.189	9.386

10)

Selection sort took the longest on average which matches my prediction because it is always n^2 in every case. The insertion sort took longer than the bubble sort which is different from my predictions (a tie). Bubble sort went the fastest with an average time of 0.122 MS at 150000 inputs. Quicksort (9.386 MS), mergesort (13.189 MS), and shell sort (15.243 MS) all did relatively well compared to insertion and selection, but mergesort and quicksort swapped compared to my predictions.

Real results:

1. BubbleSort

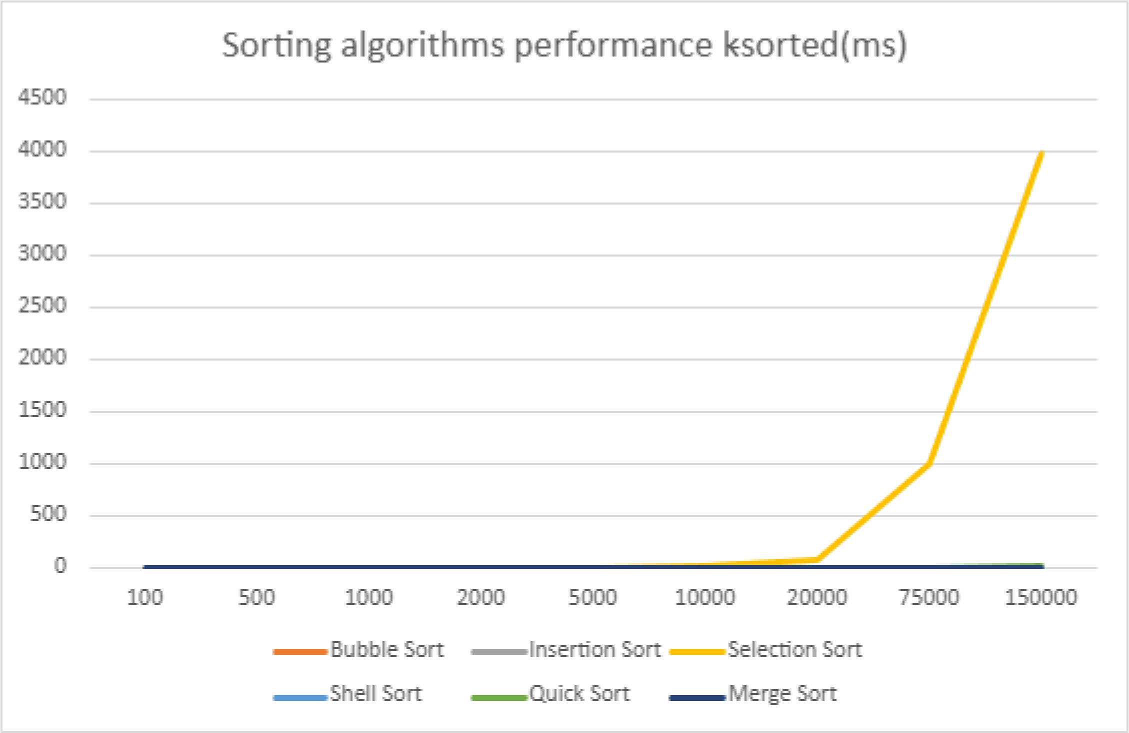
2. QuickSort

3. MergeSort

4. ShellSort

5. insertionSort

6.SelectionSort



Input Size	Bubble Sort	Insertion Sort	Selection Sort	Shell Sort	Merge Sort	Quick Sort
100	0.002	0.012	0.06	0.018	0.023	0.022
500	0.008	0.059	0.162	0.078	0.033	0.03
1000	0.017	0.025	0.212	0.039	0.068	0.064
2000	0.035	0.055	0.726	0.064	0.13	0.062
5000	0.053	0.061	4.353	0.101	0.204	0.167
10000	0.026	0.082	17.201	0.208	0.411	0.404
20000	0.045	0.163	69.014	0.439	0.86	1.011
75000	0.139	0.61	993.566	1.712	3.299	7.767
150000	0.275	1.219	3996.807	3.521	6.777	23.249

12)

The algorithms did like the completely random set however; Insertion sort did considerably better take the second-place ranking. Shell sort improved by almost 5x at 150000 inputs. Surprisingly quicksort slowed down, taking 3xas long as it took for a random set, but merge sort cut its time down by almost half.

Real rankings:

Bubble sort

Insertion sort

Shell short

Mergesort

Quicksort