CSCE 221 Assignment 2

Alex Benavides, Omar Rodriguez, Cory Thompson

September 2014

1 Description

For this assignment we ran five different sorting algorithms in order to sort four different sets of inputs $10^2, 10^3, 10^4, 10^5$.The five sorting algorithms were: selection sort, insertion sort, bubble sort, shell sort and radix sort all in C++. A sorting algorithm is an algorithm that puts elements from a list in a certain order. After getting the results we compared the time it took for each algorithm to compile then compared them using Big-O asymptotic notation. After these analysis were done we were able to distinguish which algorithm works most effectively in each given scenario.

Instructions on running the program:

1.make clean (just to remove all unnecessary files)

2.make

3.copy input files into main directory(from Assignment2 - InputGenerators - set1, or set2, or set3)

4. /automate.sh (int the main folder) (this command should automate all the iterations needed its the script I made)

5. when it finishes executing (be patient it may take a while its running all the sorting algorithms)

6. put output files in a output files folder (so they don't get deleted when running clean or get overwritten when running the automate script)

7. you will have to copy and paste all the cpp files in each of the B1-S4 folders either to a different folder or to your local machine each one of these are hard coded specifically to run inside the same folder as they are in Ex. cpp in B1 must be placed in B1 after the next few steps are done

8. in the ouput files folder run ./mfiles.sh for the first set of runs makes sure you move the B1-S4 folders either to a different folder because the script will create the folder and overwrite whatever is in it

9. Repeat that procedure(excluding the moving the B1-S4 folders after the first time through) for set1-3 and ./mfiles1-3.sh

2 Explanations

With the implementation of the sort.h file we were able to define all of the algorithms in order to make the program much more efficient. Objects which are usually classes, are used to interact with one another during the process of compiling the program. These objects are always defined in class hierarchies.

3 Algorithms

Selection sort: It has a run time of $O(n^2)$ and is not very efficient on large list. This algorithm is popular for its simplicity and in certain situations it has several advantages over more complex algorithms.

Insertion sort: Best case performance for this sort is $O(n)$ similar to the selection sort the insertion sort is not very efficient on large list but extremely efficient on small data sets.

Bubble sort: Simple algorithm that works by going through the given data and comparing each pair of adjacent items then swapping them if they are not in the correct order. Like the other two algorithms it is not the best to use for large pieces of data.

Shell sort: The running time of this algorithm heavily relies on the gap sequence it uses. The way that the elements are sorted is by dividing different parts of the array into equal amounts then using the insertion sort to get the finalized arranged elements.

Radix sort: Sort data with integer keys by grouping keys which share a significant position and value. Radix sort can also be used to sort integers or strings. It always starts with the least significant digit.

4 Theoretical analysis

Complexity	Best	Average	Worst	inc	ran	dec
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(n^2)	O(n^2)	O(n^2)
Insertion Sort	O(n^2)	O(n^2)	O(n^2)	O(n)	O(n^2)	O(n^2)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(n)	O(n^2)	O(n^2)
Shell Sort	O(nLogn)	O(n^2)	O(n^2)	O(nLogn)	O(n^2)	O(n^2)
Radix Sort	O(Nk)	O(Nk)	O(Nk)	O(n)	O(n)	O(n)

inc = Increasing Order
dec = Decreasing Order
ran = Random Order

5 Experiments

RT	Selection Sort			Insertion Sort			Bubble Sort			Shell Sort			Radix Sort		
n	inc	ran	dec	inc	ran	dec	inc	ran	dec	inc	ran	dec	inc	ran	dec
100	.005	.023	.035	.004	.022	.037	.003	.061	.062	.008	.015	.010	.014	.014	.012
10^3	.108	1.72	3.349	.137	1.7	1.7	.062	6.19	5.602	1.11	.217	.114	1.91	.143	.130
10^4	.108	168	33.63	.137	166	166	.062	596	551.583	.217	3.23	1.512	1.91	1.95	1.707
10^5	1.05	16830	33462	1.15	16801	16801	.589	60276	55861.4	15.2	1298073	18.72	24.7	24.6	21.1833

COMP	Selection Sort			Insertion Sort		
n	inc	ran	dec	inc	ran	dec
100	198	2927	5041	198	2927	5041
10^3	1998	8258706	500380	1998	258706	50038
10^4	19998	25200236	50003703	19998	25200236	500037
10^5	199998	2502510900	715827881	199998	2506679184	5000036

COMP	Bubble Sort			Shell Sort		
n	inc	ran	dec	inc	ran	dec
100	99	4972	5048	826	1186	1054
10^3	999	499742	500497	14180	20136	17363
10^4	9999	49992757	50005000	201688	294454	244771
10^5	99999	4999871596	715827881	2616692	3885858	3153491

inc = Increasing Order
dec = Decreasing Order
ran = Random Order

6 Discussion

The results of our programming experiments were very similar to the theoretical analysis given to these algorithms. I was surprised by the results of the shell-sort. Being a comparison based algorithm, I was not expecting shell-sort to perform as well as it did even when it was tasked with sorting 100,000 random integers. Radix-sort performed better than the other algorithms when tasked to sort the decreasing or random inputs. However, when the comparison algorithms were given their best case, these comparison algorithms completed much faster than radix-sort.

7 Conclusions

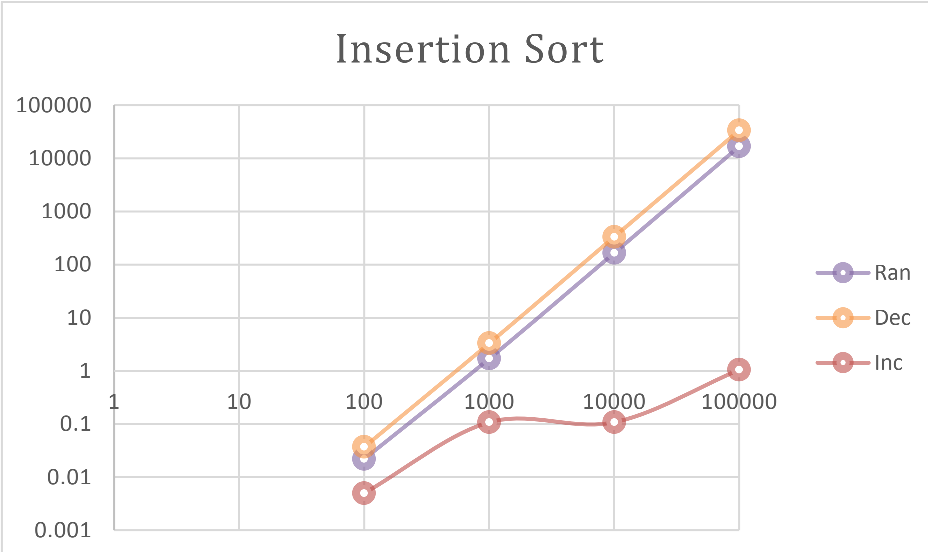
Shell-sort performed better than any other comparison based algorithm unless it was given an array already sorted. The speed of the shell-sort from the random and decreasing inputs was very impressive. All of the comparison algorithms did much better than radix-sort for the increasing inputs. That was not surprising because radix-sort took about the same amount of time no matter if the input is increasing, decreasing, or random. The experimental results agree with the theoretical analysis of these algorithms, but many factors can alter these experimental results. These factors include computer speed or disruptions from any other process the computer might be performing.

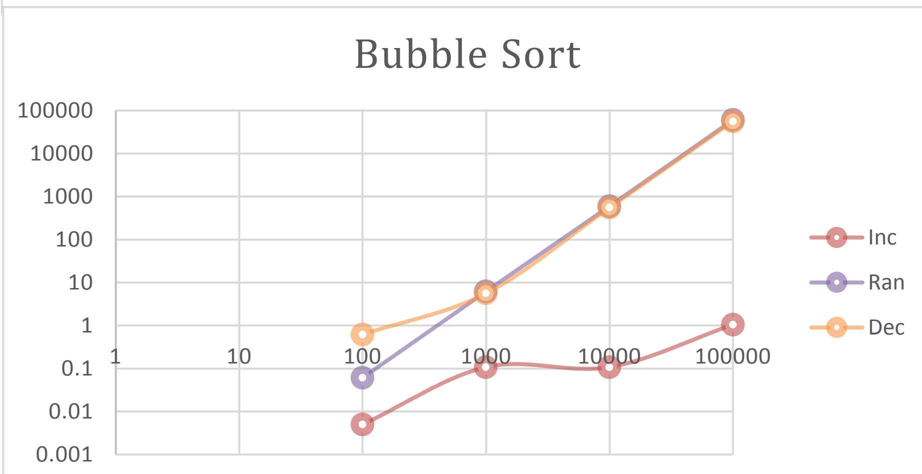
B)

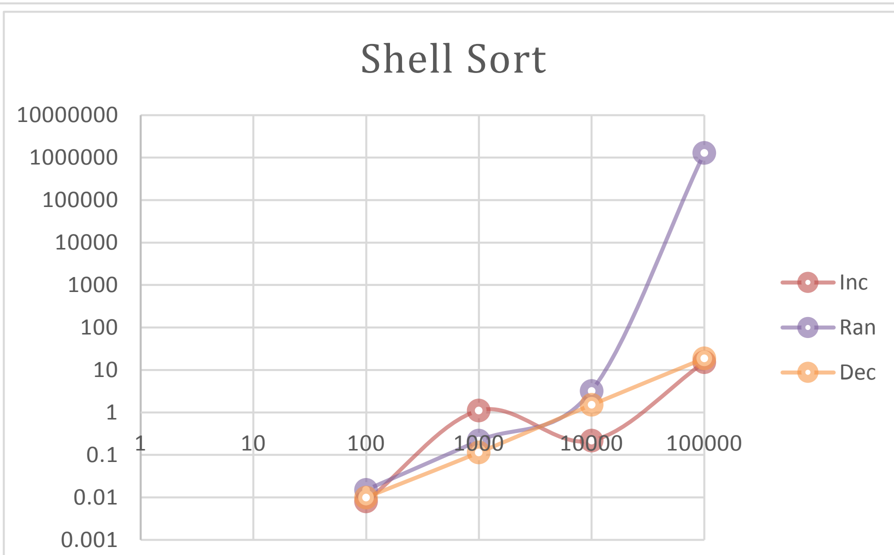
RT	Selection Sort			Insertion Sort		
n	Inc	Ran	Dec	Inc	Ran	Dec
100	0.005	0.023	0.035	0.005	0.022	0.037
1000	0.108	1.72	3.349	0.108	1.7	3.304
10000	0.108	168	33.63	0.108	166	333.857
100000	1.05	16830	33462	1.05	16801	33710.2

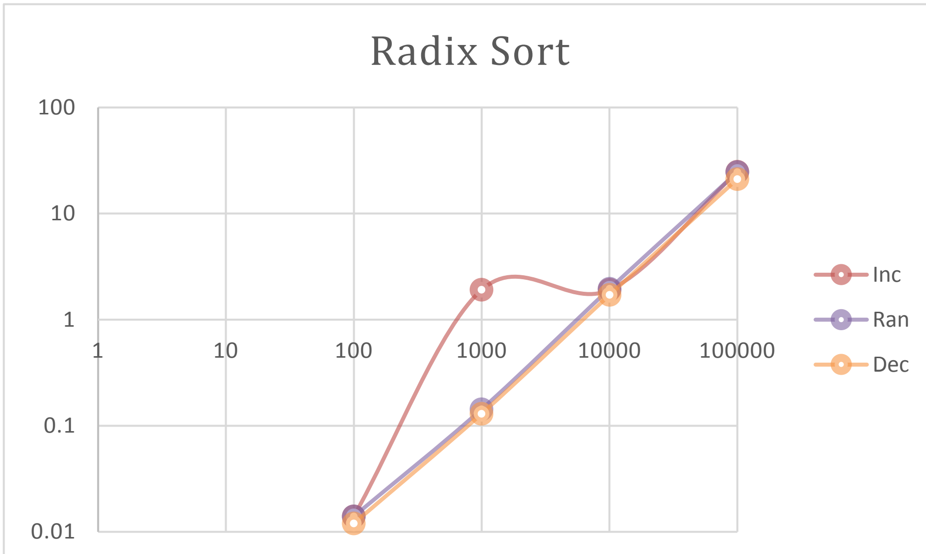
Bubble Sort			Shell Sort		
Inc	Ran	Dec	Inc	Ran	Dec
0.005	0.061	0.62	0.008	0.015	0.01
0.108	6.19	5.602	1.11	0.217	0.114
0.108	596	551.583	0.217	3.23	1.512
1.05	60276	55861.4	15.2	1298073	18.72

Radix Sort		
Inc	Ran	Dec
0.014	0.014	0.012
1.91	0.143	0.13
1.91	1.95	1.707
24.7	24.6	21.1833


Insertion Sort
100000
10000
1000
100
10
1
0.1
0.01
0.001
1
10
100
1000
10000
100000
Ran
Dec
Inc

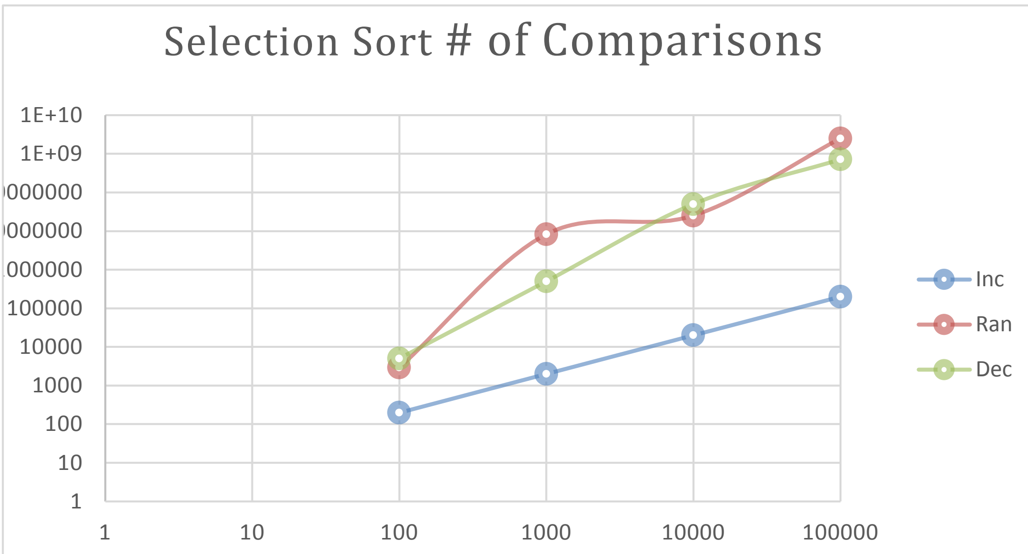

Bubble Sort
100000
10000
1000
100
10
1
0.1
0.01
0.001
1
10
100
1000
10000
100000
Inc
Ran
Dec

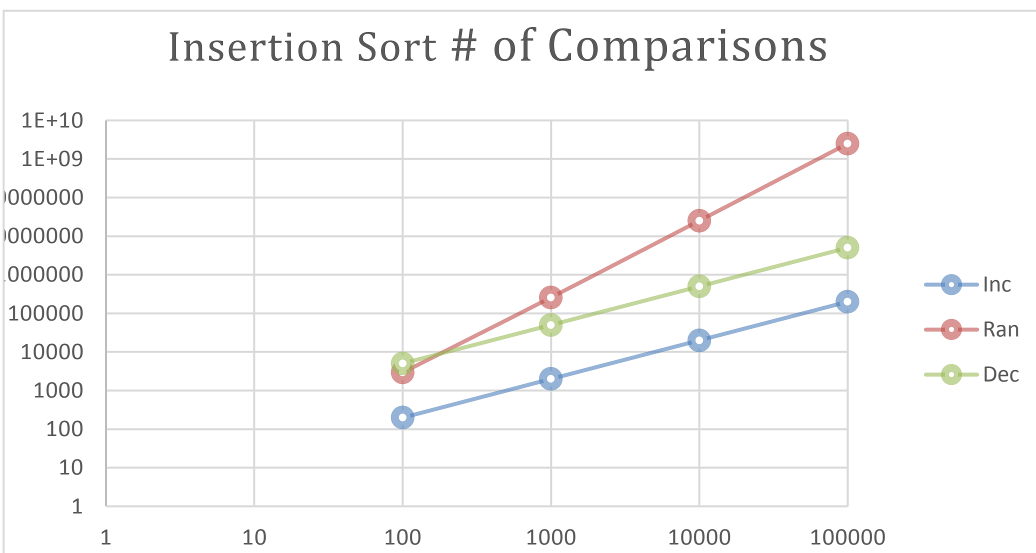

Shell Sort
10000000
1000000
100000
10000
1000
100
10
1
0.1
0.01
0.001
1
10
100
1000
10000
100000
Inc
Ran
Dec

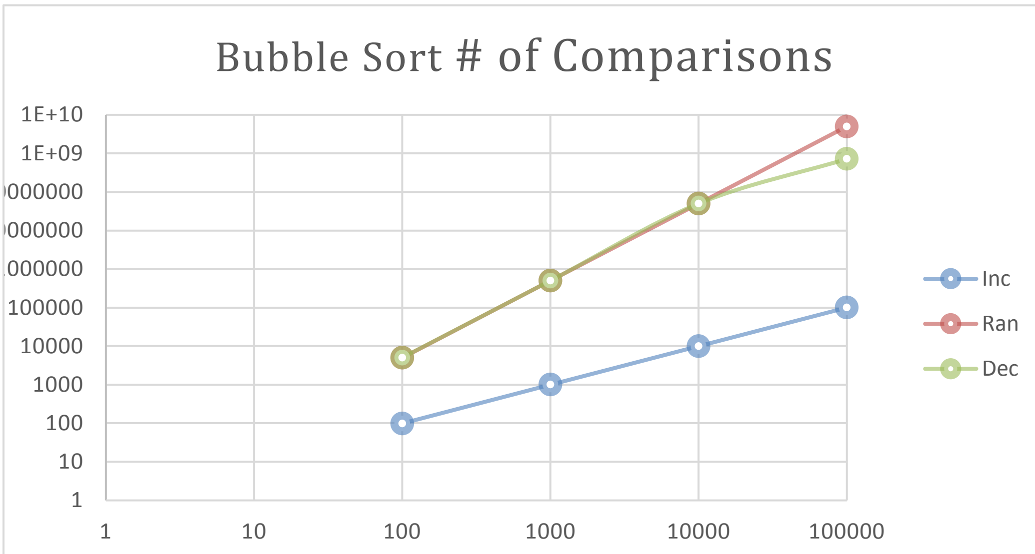


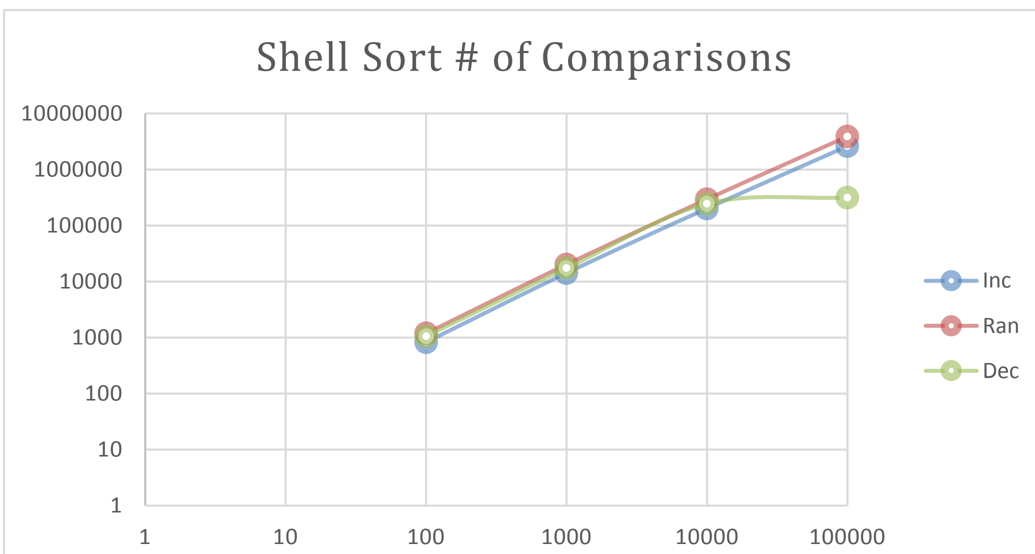
COMP		Selection Sort			Insertion Sort	
n	Inc	Ran	Dec	Inc	Ran	Dec
100	198	2927	5041	198	2927	5041
1000	1998	8258706	500380	1998	258706	50038
10000	19998	25200236	50003703	19998	25200236	500037
100000	199998	2.503E+09	7.16E+08	199998	2506679184	5000036

		Bubble Sort			Shell Sort	
	Inc	Ran	Dec	Inc	Ran	Dec
	99	4972	5048	826	1186	1054
	999	499742	500497	14180	20136	17363
	9999	49992757	50005000	201688	294454	244771
	99999	5E+09	7.16E+08	2616692	3885858	315491

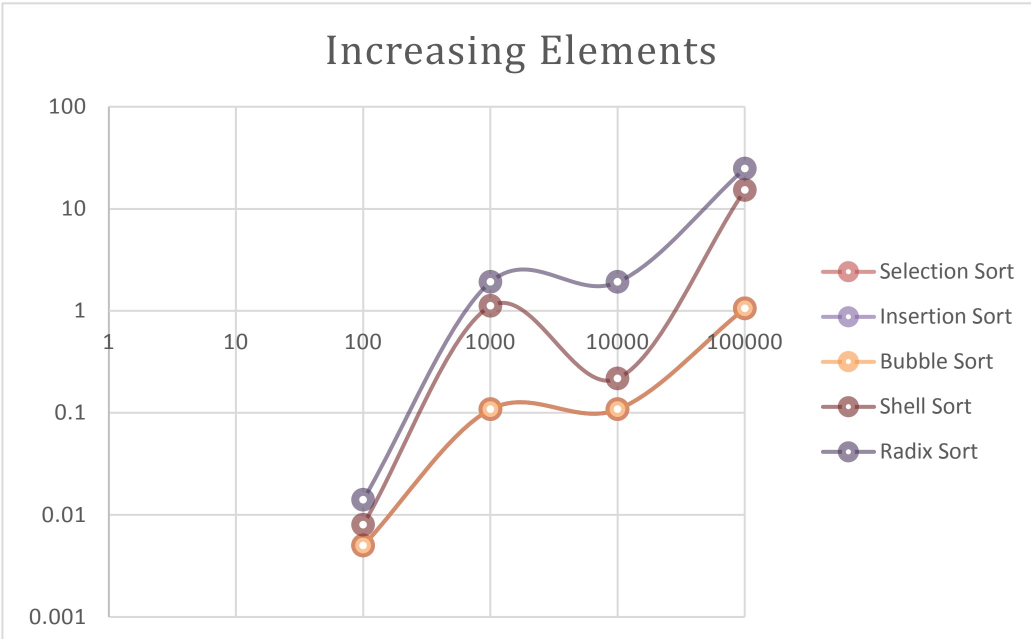

Selection Sort # of Comparisons
1E+10
1E+09
)000000
)000000
.000000
100000
10000
1000
100
10
1
1
10
100
1000
10000
100000
Inc
Ran
Dec


Insertion Sort # of Comparisons
1E+10
1E+09
)000000
)000000
.000000
100000
10000
1000
100
10
1
1
10
100
1000
10000
100000
Inc
Ran
Dec

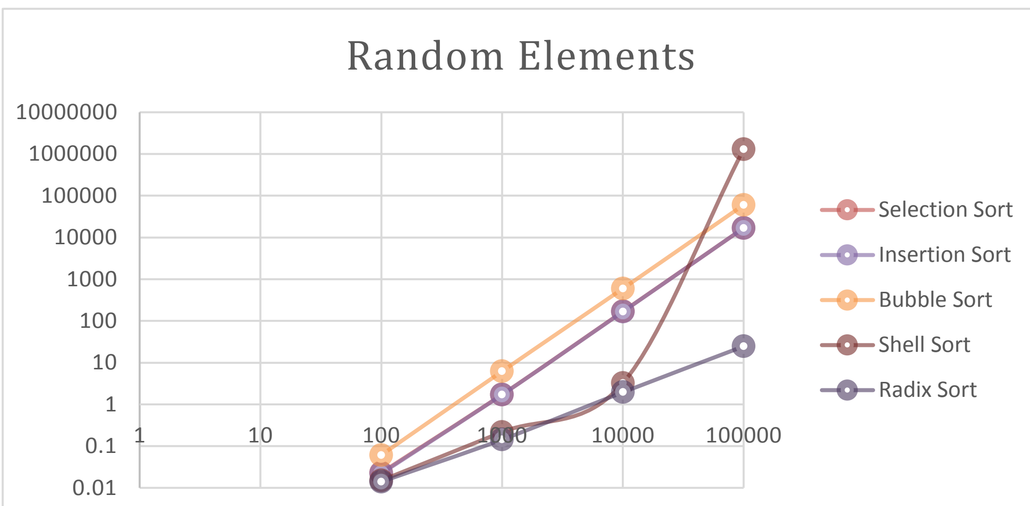

Bubble Sort # of Comparisons
1E+10
1E+09
100000
10000
1000
100
10
1
1
10
100
1000
10000
100000
Inc
Ran
Dec


Shell Sort # of Comparisons
10000000
1000000
100000
10000
1000
100
10
1
1
10
100
1000
10000
100000
Inc
Ran
Dec

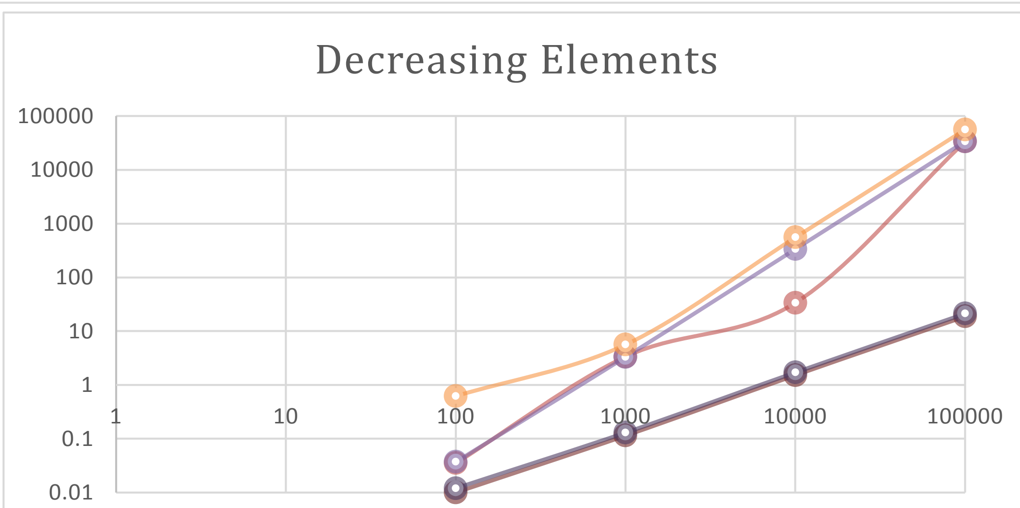
C)



Increasing Elements
100
10
1
0.1
0.01
0.001
1
10
100
1000
10000
100000
Selection Sort
Insertion Sort
Bubble Sort
Shell Sort
Radix Sort



Random Elements
10000000
1000000
100000
10000
1000
100
10
1
0.1
0.01
1
10
100
1000
10000
100000
Selection Sort
Insertion Sort
Bubble Sort
Shell Sort
Radix Sort



Decreasing Elements
100000
10000
1000
100
10
1
0.1
0.01
1
10
100
1000
10000
100000