Project Cover Page

This project is a group project. For each group member, please print first and last name and e-mail address.

1. Kevin Chou
2. Brady Skuza
3. Hasnain Bilgrami

Please write how each member of the group participated in the project.

1. Everyone did equal amount of work. We split the implementation of the algorithms into 3 parts and we collaborated on the experimentation and the report.
2.
3.

Please list all sources: web pages, people, books or any printed material, which you used to prepare a report and implementation of algorithms for the project.

Type of sources:	
People	
Web Material (give URL)	
Printed Material	Textbook
Other Sources	Lecture slides

I certify that I have listed all the sources that I used to develop solutions to the submitted project report and code.
Your signature Kevin Chou Typed Name Kevin Chou Date 9/27/15

I certify that I have listed all the sources that I used to develop a solution to the submitted project and code.
Your signature Brady Skuza Typed Name Brady Skuza Date 9/27/15

I certify that I have listed all the sources that I used to develop solution to the submitted project and code.
Your signature Hasnain Bilgrami Typed Name Hasnain Bilgrami Date 9/27/15

1 Assignment objective

The purpose of this assignment was to implement the following five sorting algorithms: selection sort, bubble sort, insertion sort, shell sort, and radix sort. The program was tested through various inputs and the runtime and number of comparsons it took was compared to the runtime in Big-O notation and theoretical runtimes of these algorithms. To build the program you first have to call the compiler, which is in the the make file, so you would use the command "make". Then you run the progerami with ./sort along with the options: the alogorithm you want to use to sort(command: -a then the algorith), the input(command: -f and file name), the output(command: -o output file name), the help option(command: -h), show input(command: -d), show time(command: -t), show number of comparisons(command: -c), and the last one, show output(command: -p). So a sample command would be: ./sort -a S -f input.txt -o output.txt -d -t -c -p. Also you can use ./sort -h for the help menu. The input file should contain in the first line with a number that tells how many numbers to sort to follow. So for example, you would input 2 in the first line and then a number to sort in the following 2 lines. In the output file, the output would be shows the input data if the option is selected, the sorted data if the option is selected, the number of comparisons if the option is selected, and the average time it took to sort the data if the option is selected.

2 Program structure

Objecte oriented programming features used include classes, polymorphism, and inheritance. An abstract sort class is first defined and each specifc kind of sort are subclasses inherited from the base sort class. Having this structure prevents writing different code for each of the sorts.

3 Algorithms

Selection sort finds the nexts largest element in the array and move it to the final position in the array. It loops through the array n times.

Bubble sort compares each element with the next and swapping their position in the array if the first element is bigger.

Insertion sort runs throught the array only once. It takes an element and compares it to each element in the array and inserts it in the correct position.

Shell sort breaks an array into smaller subarrays and sorts the elements in each subarray using an insertion sort. The subarrays are then sorted after that.

Radix sort sorts the numbers through each digit. It first sorts the tens, then the hundreds, thousands, etc. It is the only one of the five in the program that is not a comparison based sorting algorithm. It is linear time based sorting algorithm.

4 Theoretical Analysis

Complexity	Best	Average	Worst	**Complexity**	Inc	Ran	Dec
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Selecion sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion sort	O(n)	$O(n^2)$	$O(n^2)$	Insertion sort	O(n)	$O(n^2)$	$O(n^2)$
Bubble sort	O(n)	$O(n^2)$	$O(n^2)$	Bubble sort	O(n)	$O(n^2)$	$O(n^2)$
Shell sort	O(nlog(n))	$O(n^2)$	$O(n^2)$	Shell sort	O(nlog(n))	$O(n^2)$	$O(n^2)$
Radix sort	O(n)	O(n)	O(n)	Radix sort	O(n)	O(n)	O(n)

Selection sort is the slowest out of the five algorithms as it has $O(n^2)$ runtime for its best case compared to O(n) and O(nlog(n)) for the other algorithms.

Insertion sort and bubble sort have pretty much the same runtimes and comparisons. They both have O(n) for their best case and $O(n^2)$ for their average and worst case.

Shell sort has O(nlog(n)) for its best case making it slower than insertion sort, bubble sort, and radix sort. Its average and worst case is $O(n^2)$ making it the same as selection sort, insertion sort and bubble sort.

Radix sort is the fastest algorithm of the five with O(n) for its best case, average case, and worst case. It can be O(n) for all of its cases because radix sort is a linear based sort not a comparison based sort.

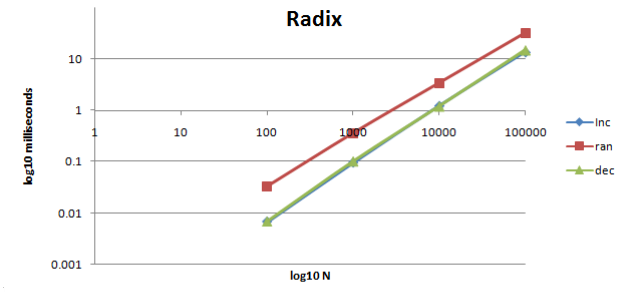
5 Experiments

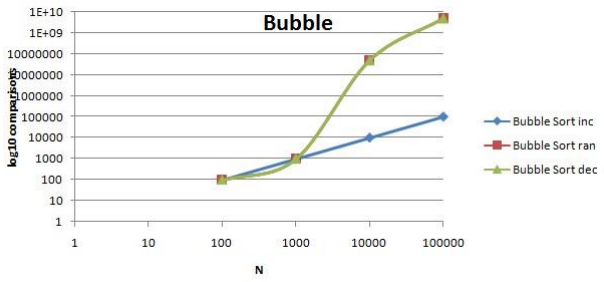
RT	Selection sort	Insertion sort	Bubble Sort	Shell sort
n	inc, ran, dec	inc, ran, dec	inc, ran, dec	inc, ran, dec
100	.0172, .0192, .0179	.00082, .00083, .00072	.00057,.00052,.00062	.0036, .0034, .0035
10^3	1.65, 1.75, 1.7	.006, .007, .006	.0038, .0042, .0042	.0473, .053, .052
10^4	147,147,170	.0057, 103, 208	.00359, 355, 328	.73, .776, .75
10^5	15455,15135,15350	.56, 10300, 20310	.035, 39020, 33712	8.45, 12.3, 13.4

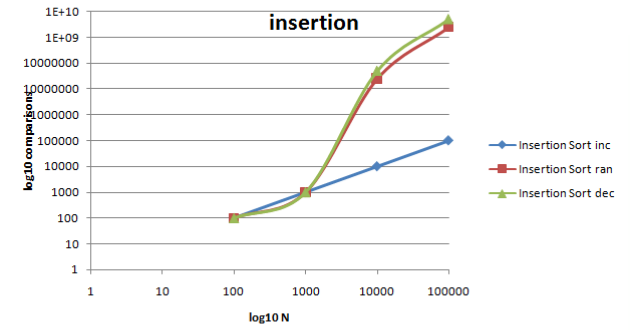
RT	Radix sort
n	inc, ran, dec
100	.005, .003,.005
10^3	.097, .362, .103
10^4	1.26, 3.4, 1.2
10^5	13.8,32.6,16

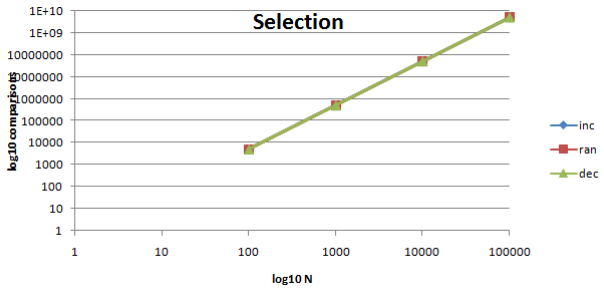
#COMP	Selection Sort	Insertion Sort	Bubble Sort
n	inc, ran, dec	inc, ran, dec	inc, ran, dec
100	4950, 4950, 4950	99, 99, 99	99, 99, 99
10^3	499500, 499500, 499500	999, 999, 999	999, 999, 999
10^4	49995000, 49995000, 49995000	9999, 25044713, 49995000	9999, 49971995, 49995000
10^5	4.9 X 10^9, 4.9 X 10^9, 4.9 X 10^9	99999, 4.5 X 10^9, 4.9 X 10^9	99999, 4.9 X 10^9, 4.9 X 10^9

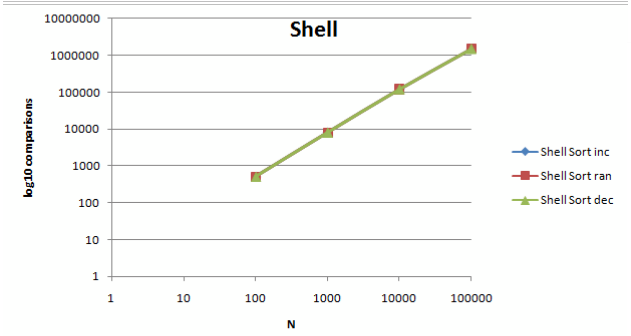
#COMP	Shell Sort
n	inc, ran, dec
100	503, 503, 503
10^3	8006, 8006, 8006
10^4	120005, 120005, 120005
10^5	1500006, 1500006, 1500006

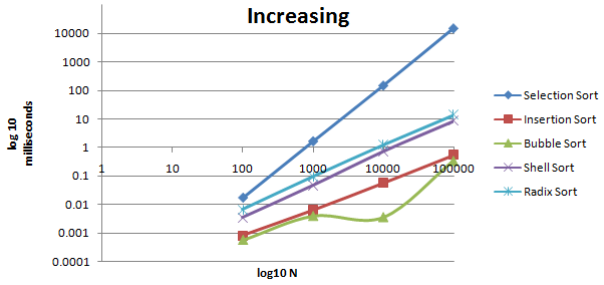

Radix
log10 milliseconds
log10 N
inc
ran
dec

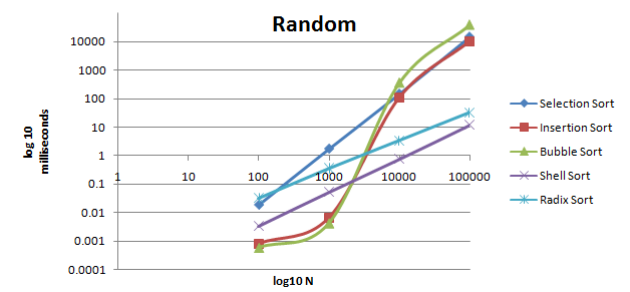

Bubble
log10 comparisons
N
Bubble Sort inc
Bubble Sort ran
Bubble Sort dec

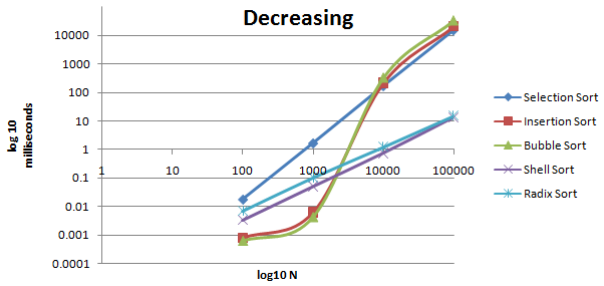

insertion
log10 comparisons
log10 N
Insertion Sort inc
Insertion Sort ran
Insertion Sort dec


Selection
log10 comparisons
log10 N
inc
ran
dec


Shell
log10 comparisons
N
Shell Sort inc
Shell Sort ran
Shell Sort dec


Increasing
log 10 milliseconds
log10 N
Selection Sort
Insertion Sort
Bubble Sort
Shell Sort
Radix Sort


Random
log 10 milliseconds
log10 N
Selection Sort
Insertion Sort
Bubble Sort
Shell Sort
Radix Sort


Decreasing
log 10 milliseconds
log10 N
Selection Sort
Insertion Sort
Bubble Sort
Shell Sort
Radix Sort

6 Discussion

Our experimental results were pretty accurate to the theoretical results with a few exceptions. In the increasing or best case scenario, insertion sort and bubble sort ran the fastest, and was followed by shell, radix, and selection respecitively. In the decreasing or random scenario (average or worst case) the grraphs and data were very similar as almost all of them had $O(n^2)$. The exception we found was that insertion sort and bubble sort were faster than radix sort and shell sort when $n \leq 1000$. This is because Big-O only marks the end behavior of an algorithm. Radix sort seems to be the best choice for a large input but not as good for a small input.

7 Conclusion

From the experiment, we can conclude that insertion sort and bubble sort are good for a relatively small input, and shell sort and radix sort are good for larger inputs. Selectoin sort is the worst of all the algorithms and is not very good for both small and large inputs. The experimental results mostly matched up to theoretical results and factors that should be considered in this experiment include, different implementation of each sorting algorithm or possibly the correctness of the implentation of the algorithms.