**Simulation of sorting techniques**

Evaluation on the performance of varying sorting technique

**Contents**

* Summary
* Introduction
  + Data Sets used
  + Run through of sorting methods
  + Running experiments
* Tacoma Deaths results (first Real data set)
* Washington Deaths results (second Real data set)
* Sorting land Deaths results (first synthetic)
* Reverse Sorted Land Deaths results (second synthetic)
* Analysis of sorting techniques

**Summary**

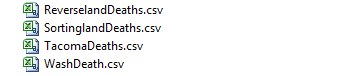
This report conveys the results of a simulation experiment to measure the performance of several sorting algorithms. The ability to effectively measure various performance measures is an essential skill for researchers and engineers, and this report helps us develop this skill. In this assignment, we made a simulation experiment for various sorting techniques, evaluate their performance, and provided graphs that illustrate the results and interpret the behavior of each sorting technique. In conclusion we provide our results and when to use each sorting technique.

**Introduction**

For this experiment we use the following sorting methods: Insertion sort, Selection Sort, Bubble sort, Merge sort, and Quick sort. These methods were practiced on various real data sets from Data.gov, but for this report we will only show 2 of these data sets. The data reflects the deaths per week in Tacoma and the other includes the deaths per week in the state of Washington. We also created 2 synthetic data sets which reflect the real data with the exception that they both have desired degree of data sortedness.

**Data Sets used**

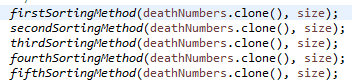
The following comma separated value files represent the data used in this experiment.



The top 2 files being synthetic and the proceeding files being real data.

**Run through of sorting methods**

In our java file we label each of our methods as follows



To translate,

Insertion sort – firstSortingMethod

Selection sort – secondSortingMethod

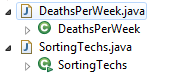
Bubble sort – thirdSortingMethod

Merge sort – fourthSortingMethod, divideConquer, conquer

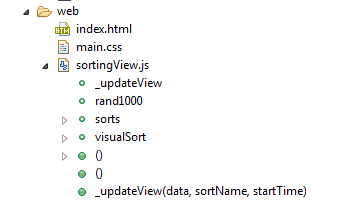
Quick sort – fifthSortingMethod, quickSort, partition and will represent the changing values in our analysis.

**Running Experiments**

For this experiment all files were running using the below java files, with SortingTechs being the main.



In the visualization part of this project coming up, the web folder is used with the below files

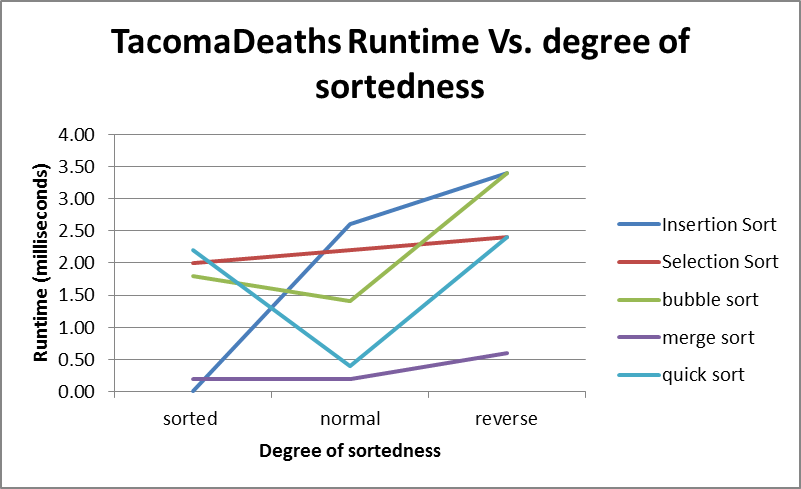


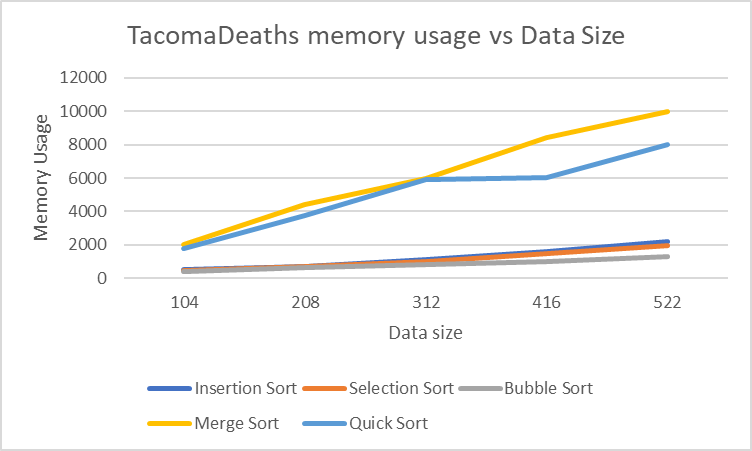
This is incomplete but if you would like to see it in action just copy the index.html path into your browser.

**Tacoma Deaths results**

The illustrations demonstrate the following, First graph: runtime against data size, Second graph: runtime against degree of sortedness, Third graph: memory usage against data size, Fourth graph: memory usage against runtime.

IMPORTANT: our degree of sortedness will be measure of sorted (presorted data), reverse sorted (sorted but in opposite intended direction), and normal (random distribution)





**Washington Deaths results**

The illustrations demonstrate the following, First graph: runtime against data size, Second graph: runtime against degree of sortedness, Third graph: memory usage against data size, Fourth graph: memory usage against runtime.

IMPORTANT: our degree of sortedness will be measure of sorted (presorted data), reverse sorted (sorted but in opposite intended direction), and normal (random distribution)\

**Sorting land Deaths results**

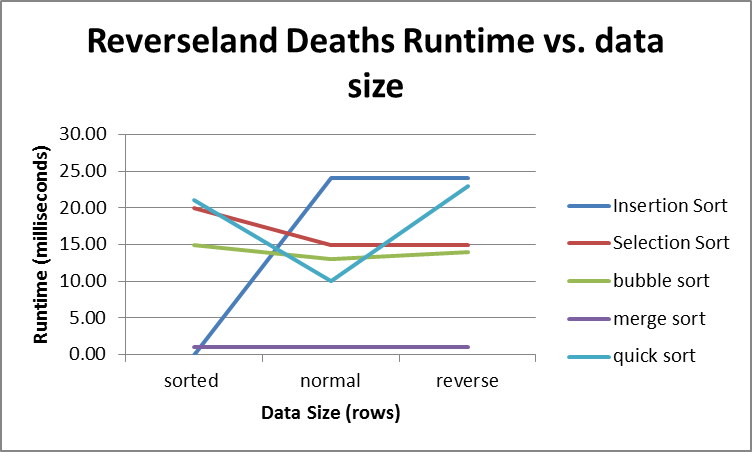
The illustrations demonstrate the following, First graph: runtime against data size, Second graph: runtime against degree of sortedness, Third graph: memory usage against data size, Fourth graph: memory usage against runtime.

IMPORTANT: our degree of sortedness will be measure of sorted (presorted data), reverse sorted (sorted but in opposite intended direction), and normal (random distribution)

**Reverse land Deaths results**

The illustrations demonstrate the following, First graph: runtime against data size, Second graph: runtime against degree of sortedness, Third graph: memory usage against data size, Fourth graph: memory usage against runtime.

IMPORTANT: our degree of sortedness will be measure of sorted (presorted data), reverse sorted (sorted but in opposite intended direction), and normal (random distribution)



**Analysis of sorting techniques**

Insertion sort: its best time was constant in an already presorted or mostly sorted list, also used little memory. Besides this rather slow in all other conditions so use sparingly.

Selection sort: this used hardly any memory so useful in the case where you need to conserve, in all other cases I would avoid if possible

Bubble sort: the same can be said for bubble sort as selection sort, though it was surprisingly faster on average.

Merge sort: was by far the fastest on average, but also used the most memory. So, use when speed is necessary.

Quick sort: second fastest on average, but did use less data than merge so in terms of when to use, quick sort has a wide range.