# Sorting simulation project

TCSS – 558 Master seminar research
Project Report – Fall 2015

Shanthini Sivaraman

# Contents

# 1. Problem Description

Running a simulation experiments for various sorting techniques, evaluate their performance, draw graphs that illustrate the results and interpret the behavior of each sorting technique.

# 2. Sorting technique's algorithm

a) Bubble Sort:

   i. Loop through the array and compare each pair of adjacent elements from the beginning of an array and, if they are in reversed order, swap them.
   ii. The n-th pass finds the n-th largest element and puts it into its final place. So, the inner loop can avoid looking at the last n-1 items when running for the n-th time.
   iii. The algorithm must pass one whole pass without any swap to know if it's sorted.

b) Insertion sort:

   i. Loop through the array, picks one element from input data at a time.
   ii. Compares it with other elements in the array.
   iii. If larger, it leaves the element in place and moves to the next. If smaller, it finds the correct position within the sorted list, shifts all the larger values up to make a space, and inserts into that correct position.
   iv. Algorithm repeats as long as there are no elements in the input array.

c) Selection Sort:

   i. Selection sort follows searching and sorting technique.
   ii. Loop through the array, finds either maximum or minimum in the unsorted list.
   iii. Copies the Index of this max or min element and swaps it with element at the end or beginning of the array accordingly.
   iv. The number of times the sort passes through the array is one less than the number of items in the array.

d) Merge sort:

   i. Merge sort follows divide and conquer technique.
   ii. Find middle of the array(m) and divide the input array into two arrays each containing half the elements from the input array.
   iii. Divide this array recursively until each array list has one element each which is considered sorted.
   iv. Conquer by recursively sorting the two subarrays

A[l...m] and A[m + 1 .. r].
l→ left index
r →right index
m→mid index

v. Combine the elements back in A[l .. r] by merging the two sorted subarrays A[p .. m] and A[m + 1 .. r] into a sorted sequence.

e) Quick sort:

i. Quick sort follows divide and conquer technique.
ii. Choose a pivot value. First element is taken as the pivot value.
iii. Partition : Hoare partition scheme – Rearrange the elements such that all elements which are less than pivot goes to the left of the array and those greater goes to the right if the array. Values equal to the pivot can stay in any part of the array.
iv. Above Quick sort steps are recursively called for the left array with smaller elements and right side array with larger elements until they are sorted.

## 3. Data Sets

**Real Data Set:**

Diabetes_Data is used from the UCI machine learning repository datasets available online.

http://archive.ics.uci.edu/ml/datasets/

Data Set 1: The Patient number of the diabetes database is used to carry out sorting analysis. There are 101,766 records.

Data Set 2: Number of lab procedures attributes is used to carry out sorting analysis. There are 101,766 records.

**Synthetic Data Set**

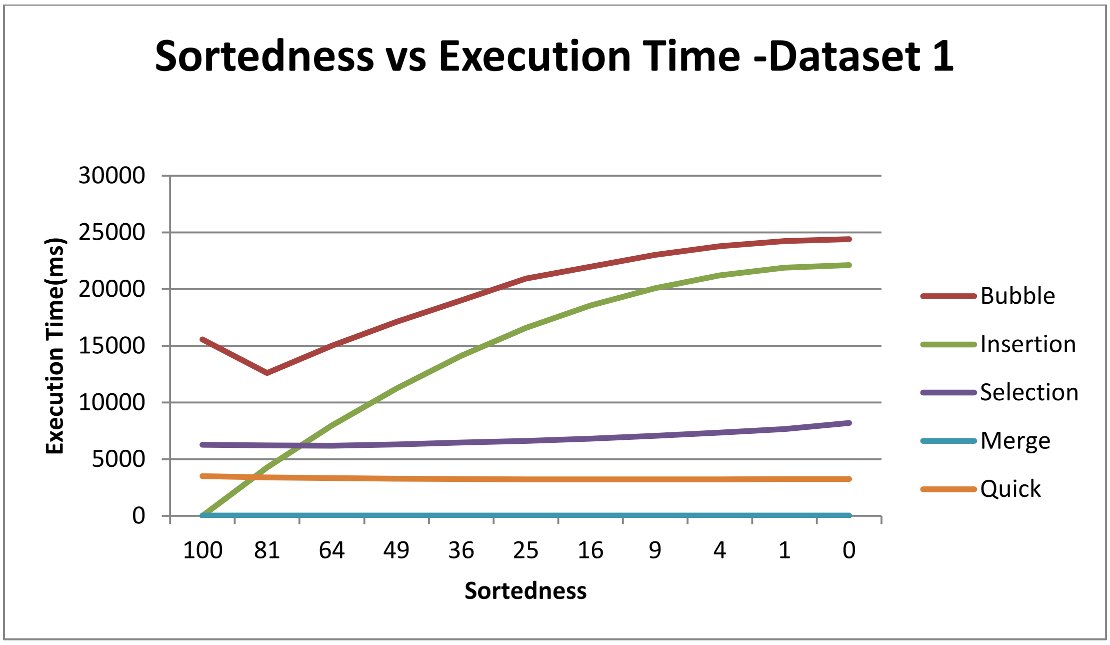
Data Set 1: This dataset is obtained using Normal Distribution with Mean =1000 and Standard deviation of 9999. There are 105,000 records.

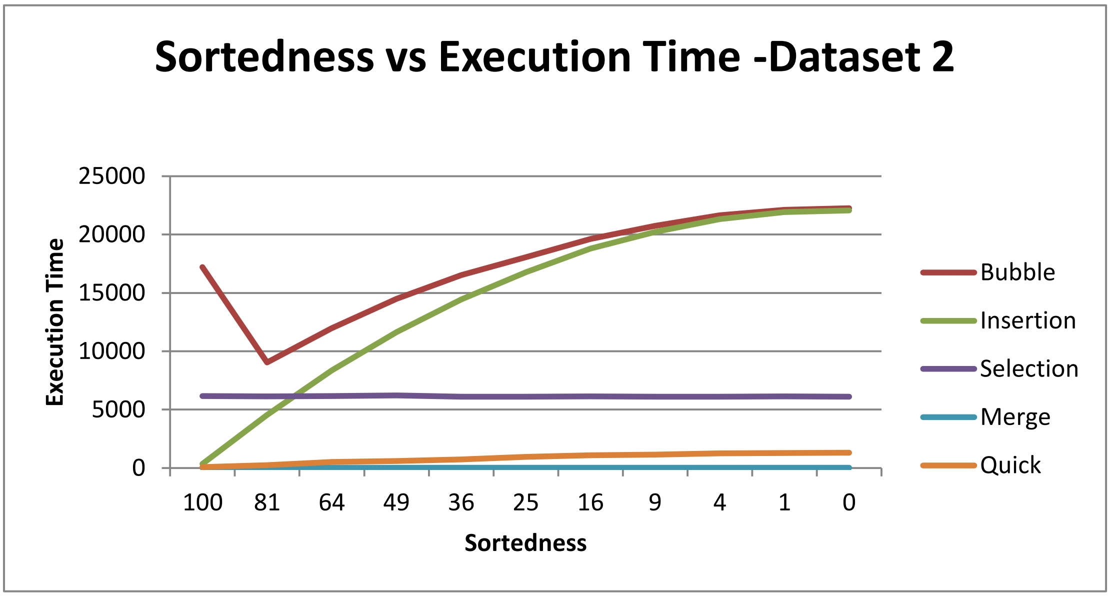
Data Set 1: This dataset is obtained using Normal Distribution with Mean =5000 and Standard deviation of 25. There are 150,000 records.

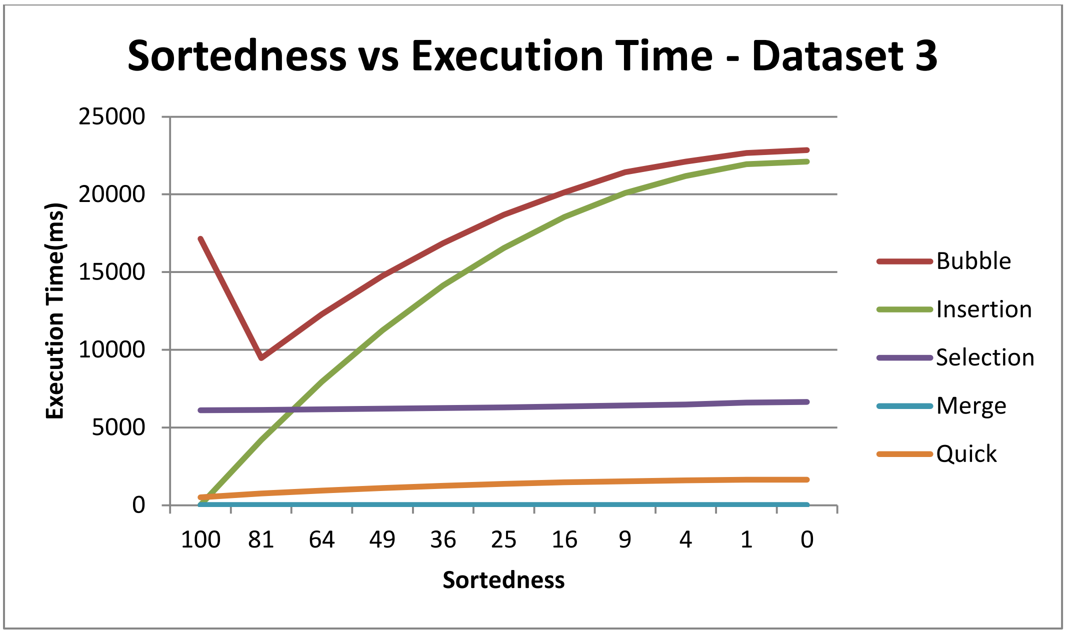
# 4. Graph

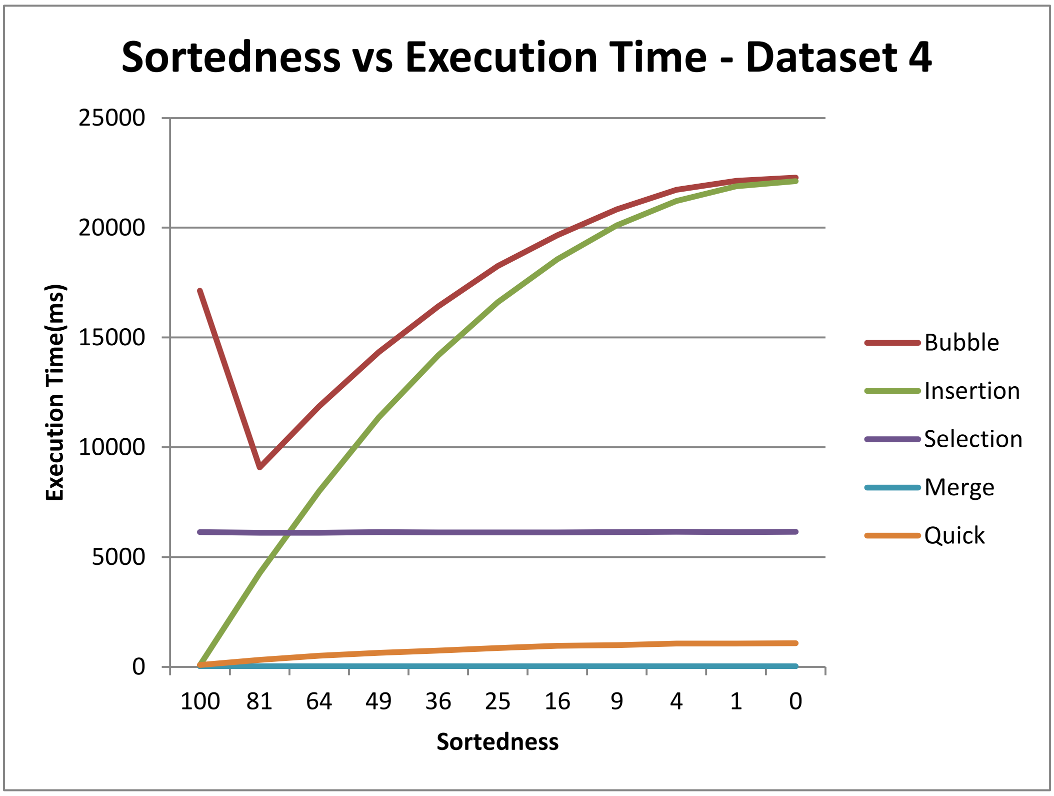
## 4.1 Sortedness vs Execution Time

This experiment was run with dataset of size 50000 and sortedness percent of 100,81,64,49,36,25,16,9,4,1and 0.



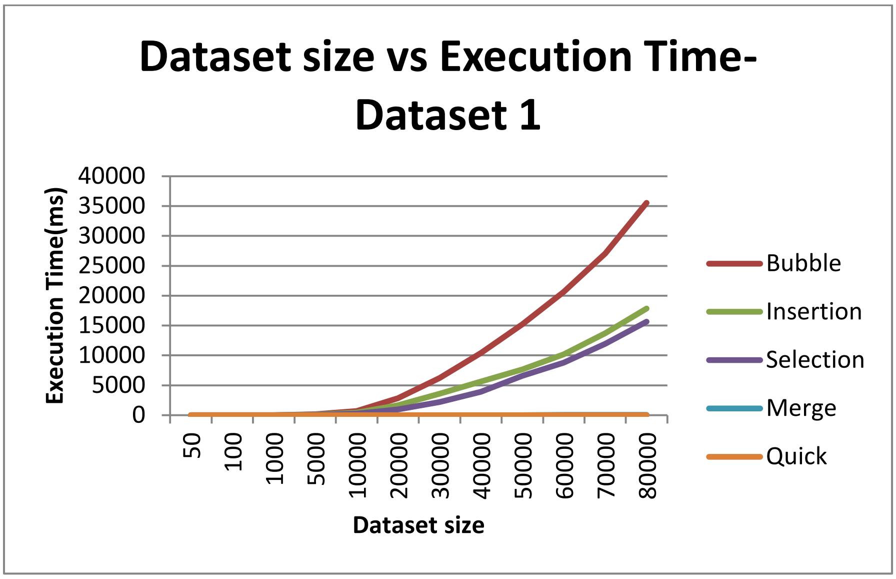


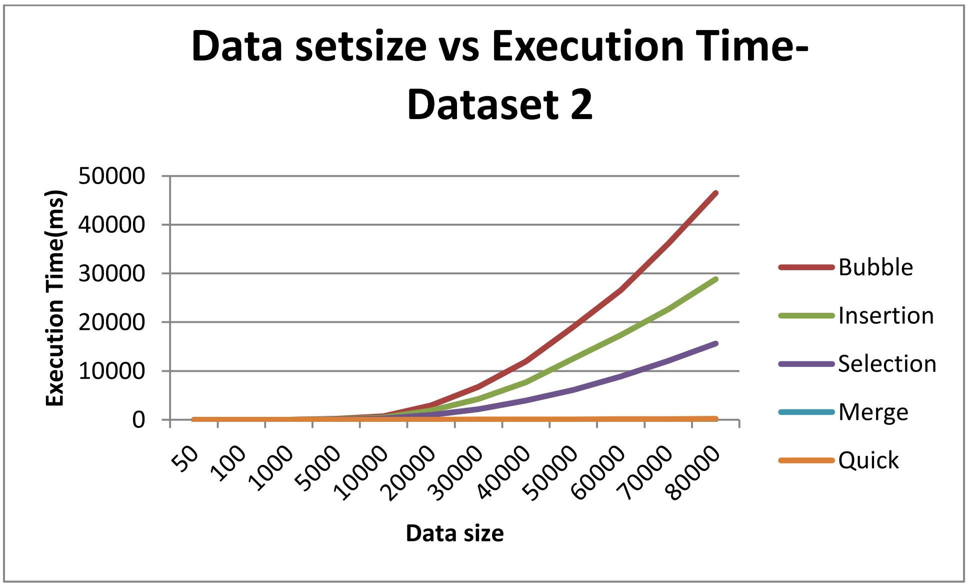

Sortedness vs Execution Time - Dataset 3
Execution Time(ms)
25000
20000
15000
10000
5000
0
100
81
64
49
36
25
16
9
4
1
0
Sortedness
Bubble
Insertion
Selection
Merge
Quick

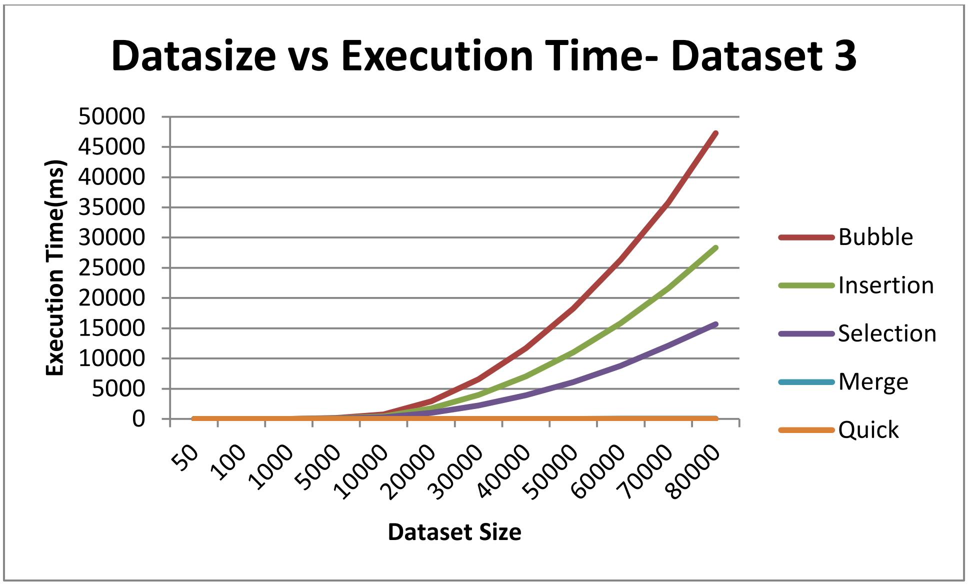

Sortedness vs Execution Time - Dataset 4
Execution Time(ms)
25000
20000
15000
10000
5000
0
100
81
64
49
36
25
16
9
4
1
0
Sortedness
Bubble
Insertion
Selection
Merge
Quick

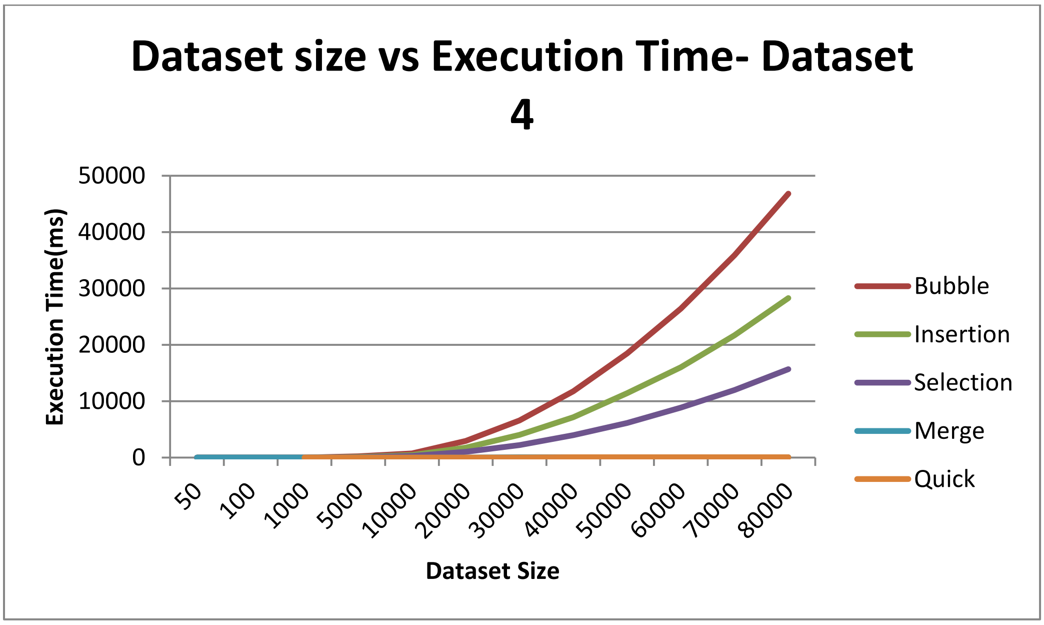
## 4.2 Dataset Size vs Execution Time

This experiment was run with dataset of below size 50,100,1000,5000,10000,20000,30000,40000,50000,60000,70000 and 80000 .



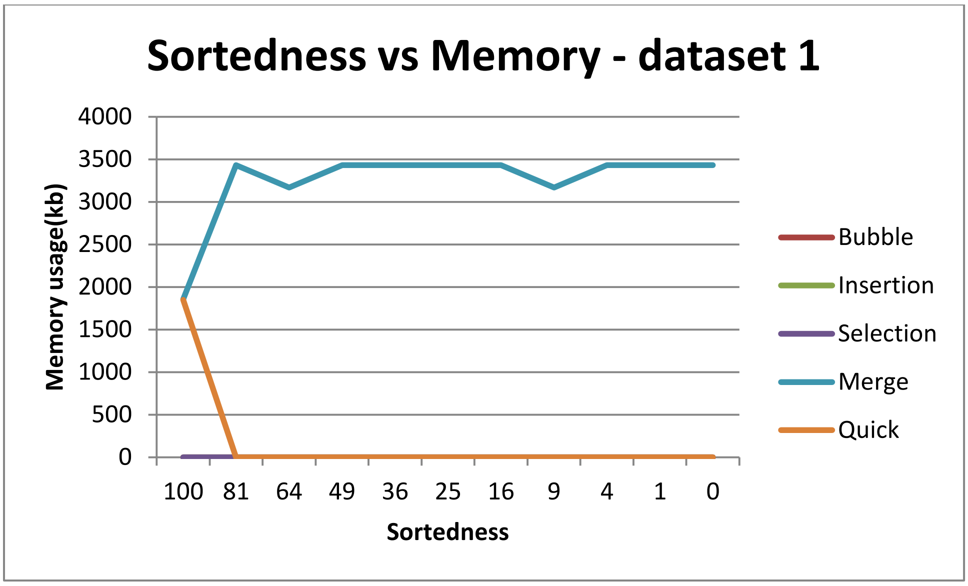


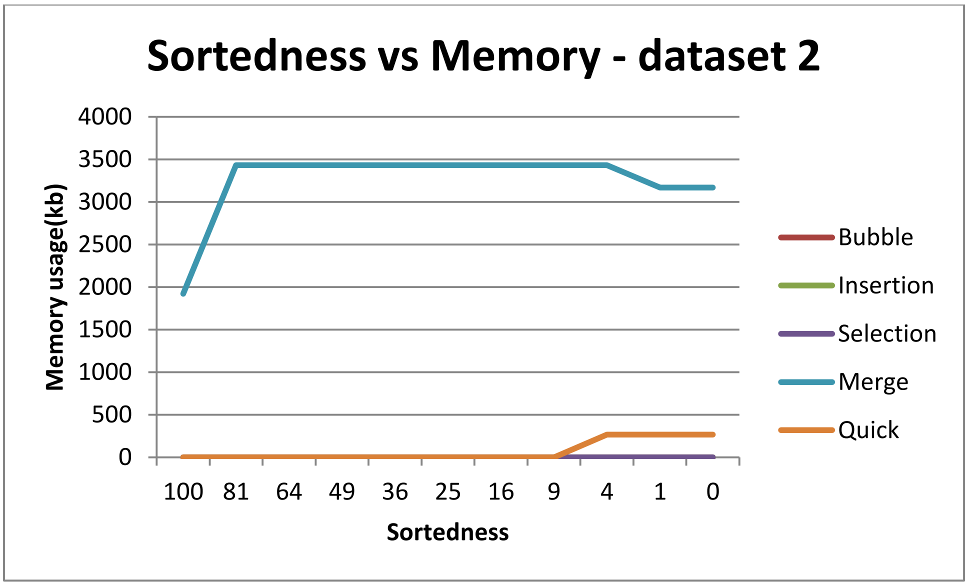


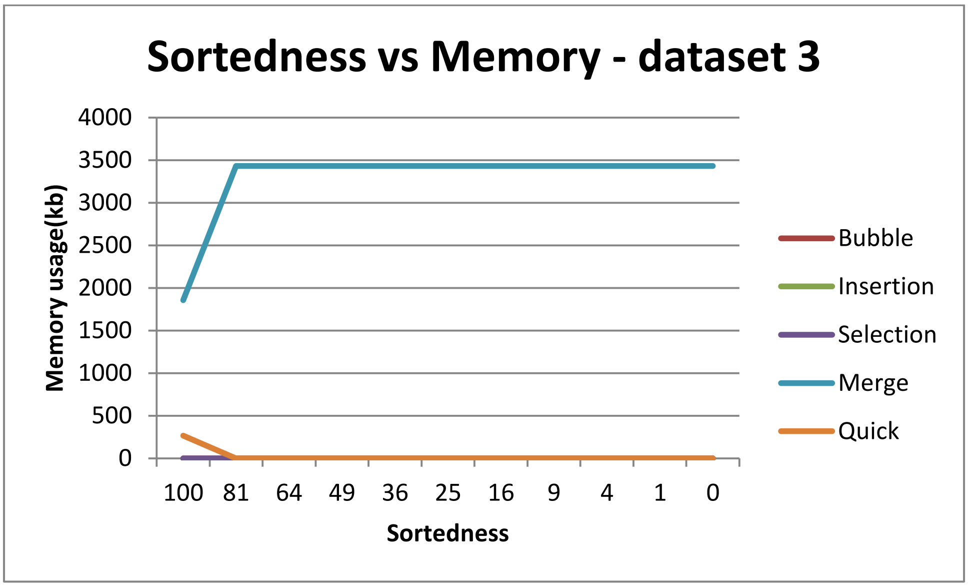


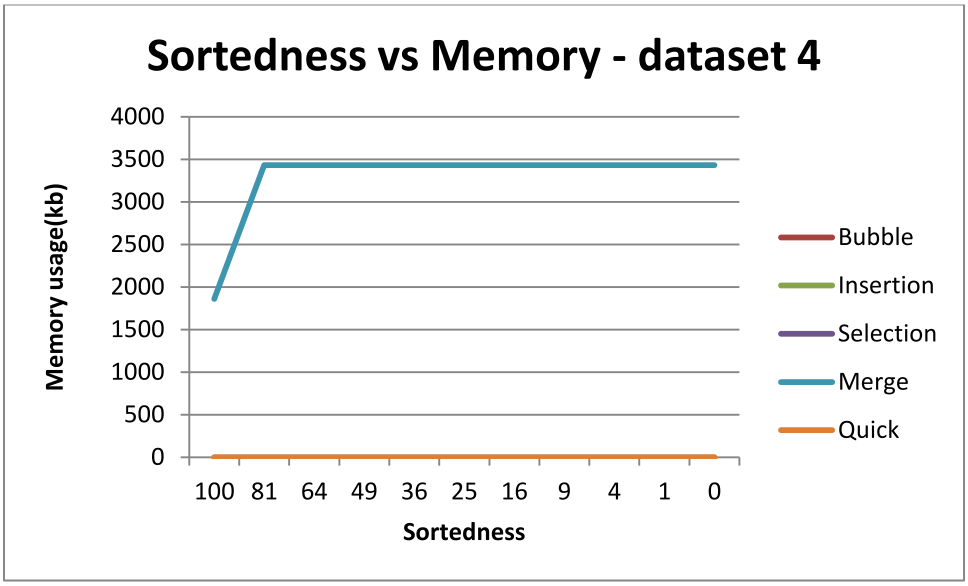
### 4.3 Sortedness vs Memory usage

This experiment was run with dataset of size 50000 and sortedness percent of 100,81,64,49,36,25,16,9,4,1and 0.


Sortedness vs Memory - dataset 1
Memory usage(kb)
4000
3500
3000
2500
2000
1500
1000
500
0
100
81
64
49
36
25
16
9
4
1
0
Sortedness
Bubble
Insertion
Selection
Merge
Quick

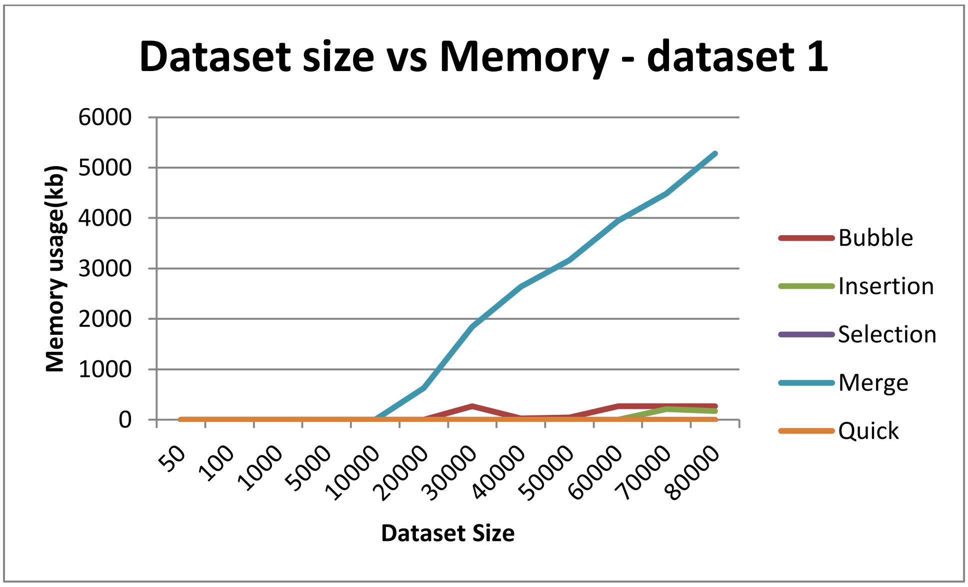

Sortedness vs Memory - dataset 2
Memory usage(kb)
4000
3500
3000
2500
2000
1500
1000
500
0
100
81
64
49
36
25
16
9
4
1
0
Sortedness
Bubble
Insertion
Selection
Merge
Quick

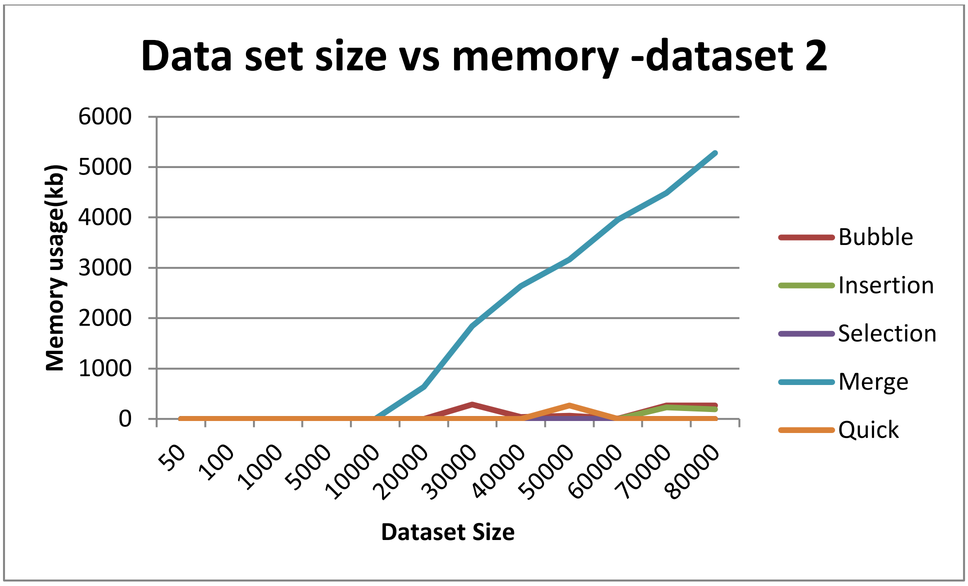

Sortedness vs Memory - dataset 3
Memory usage(kb)
4000
3500
3000
2500
2000
1500
1000
500
0
100
81
64
49
36
25
16
9
4
1
0
Sortedness
Bubble
Insertion
Selection
Merge
Quick

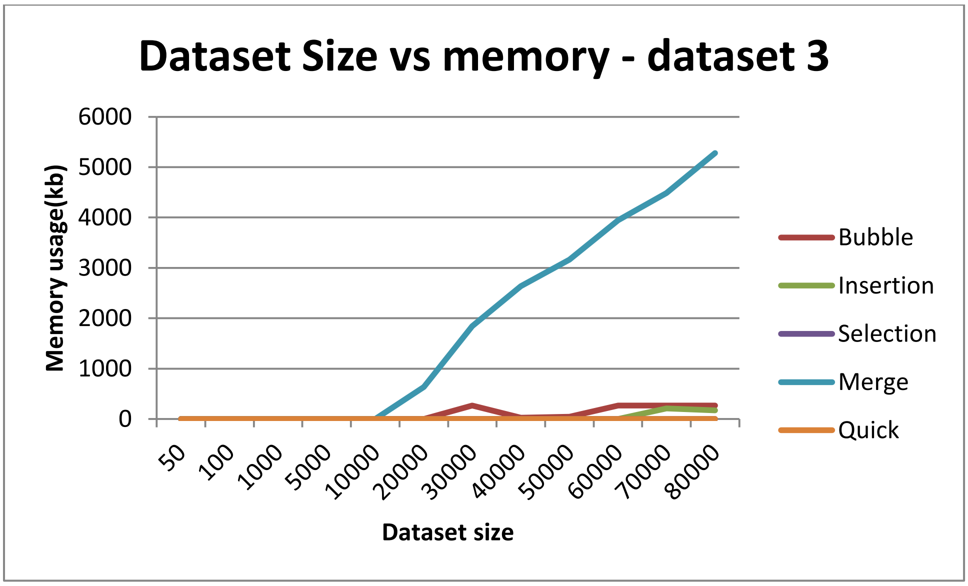


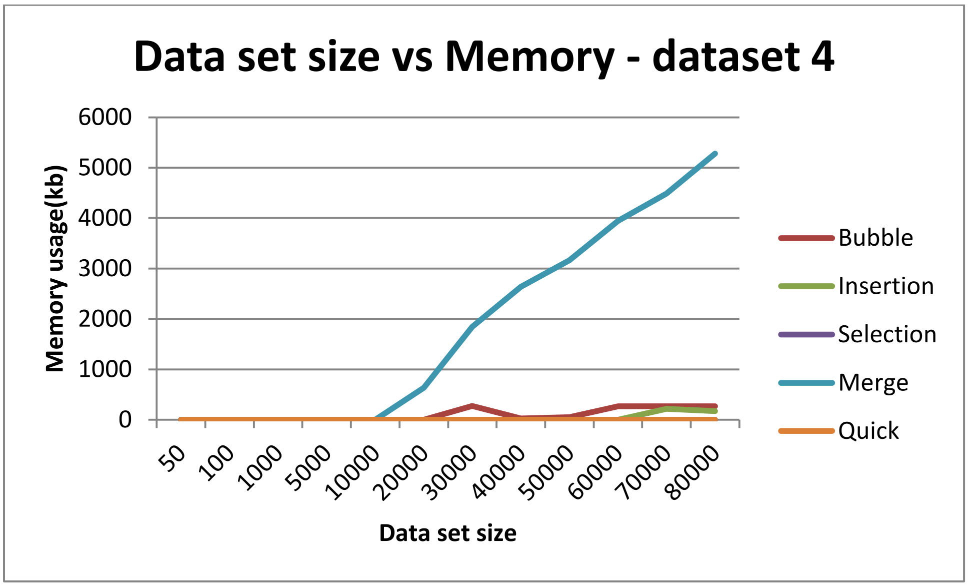
## 4.4 Dataset Size vs Memory usage

This experiment was run with dataset of below size 50,100,1000,5000,10000,20000,30000,40000,50000,60000,70000 and 80000 .




Data set size vs memory -dataset 2
Memory usage(kb)
6000
5000
4000
3000
2000
1000
0
50
100
1000
5000
10000
20000
30000
40000
50000
60000
70000
80000
Dataset Size
Bubble
Insertion
Selection
Merge
Quick


Dataset Size vs memory - dataset 3
Memory usage(kb)
6000
5000
4000
3000
2000
1000
0
50
100
1000
5000
10000
20000
30000
40000
50000
60000
70000
80000
Dataset size
Bubble
Insertion
Selection
Merge
Quick


Data set size vs Memory - dataset 4
Memory usage(kb)
6000
5000
4000
3000
2000
1000
0
50
100
1000
5000
10000
20000
30000
40000
50000
60000
70000
80000
Data set size
Bubble
Insertion
Selection
Merge
Quick

# 5. Sorting Analysis

**Bubble Sort analysis :**

- Bubble sort is usually used for smaller datasets. This algorithms works on O(n^2) complexity on best ,average and worst case.
- Bubble sort works best if the array is almost sorted but has a little overhead than insertion sort.

**Insertion Sort analysis:**

- Insertion sort is usually used for smaller datasets. This algorithms works on O(n) complexity on best case and O(n^2) for average and worst case.
- Insertion sort works best choice if the array is almost sorted.

**Selection Sort analysis:**

- Selection sort does not adapt to the data anyways. So this algorithms always works on O(n^2) complexity on best ,average and worst case.
- Selection sort has the property of minimizing the number of swaps. In applications where the cost of swapping items is high, selection sort may be the algorithm of choice.

**Merge Sort analysis:**

- Merge sort behaves the same irrespective of the sortedness of the array and data size.This algorithms always works on O(n log n) complexity on best ,average and worst case.
- Merge sort is the algorithm of choice when stability is required and the data set is huge and is stored on external devices.
- If memory space is not available merge sort is not preferred because merge sort uses temporary arrays for sorting and it overflows memory. Hence this sorting occupies more extra memory.

**Quick Sort analysis:**

- Quicksort's performance is dependent on pivot selection algorithm.This algorithms works on O(n log n) complexity on best and average and O(n^2) on worst case.
- Quick sort is typically faster than merge sort when the data is stored in memory. Quicksort works best when the array is randomly sorted.
- Worst case behaviour happens when the array is already sorted and the first element is chosen as the pivot element. Worst case is usually rare scenario.
- Merge sort and Quick sort work best on large datasets in order of O(n log n) in average case and best case complexity.

# 6. Choice of Algorithm

Below listed are ideal choice of algorithms:

| *Sortedness* | *Best Sorting Algorithm* | *Worst Sorting Algorithm* |
|---|---|---|
| Fully sorted array | Insertion Sort - O(n)<br>Bubble Sort - O(n^2) | Selection sort - O(n^2)<br>Quicksort O(n^2) - Rare case |

| | | |
|---|---|---|
| Inversely sorted array | Merge Sort - O(n log n) | Bubble Sort - O(n^2)<br>Insertion Sort - O(n^2)<br>Selection Sort - O(n^2) |
| Randomly sorted array | Quick Sort - O(n log n)<br>Merge Sort - O(n log n) | Bubble Sort - O(n^2)<br>Insertion Sort - O(n^2)<br>Selection Sort - O(n^2) |

## 7. Measure of Sortedness

- Inversion count is measured to determine the measure of sortedness.
- The number of inversions is the number of items that would have to switch places in order to sort the list which gives the inversion count.
  Eg : Consider array with below elements:

  1 2 3 4 5 6
  Here the inversion count is zero as all the elements are in their position already.

  In this array 6 5 4 3 2 1
  6 has to be moved 5 places to right
  5 has to be moved 4 places to right
  4 has to be moved 3 places to right
  3 has to be moved 2 places to right
  2 has to be moved 1 places to right

  Inversion count is 15. This array will be sorted after 15 inversions.

- So when the array is fully sorted inversion count will be 0 and when it is reversely sorted count is largest which is n*(n-1)/2
- Inversion count depends on the largest possible value depends on the length, so inversion count obtained is normalized in range between (0 to 1)
- So if the array is sorted inversion count will be less and if unsorted inversion count will be greater which is unsortedness measurement.
- In Order to compute sortedness, we do sortedness percent = (1-unsortedness)*100
- Fully sorted array will have = 100% sortedness and unsorted array will have = 0% sortedness

## 8. Results and Observations

Thus the ideal choice of algorithm would depend on the size of the data set and measure of sortedness of the array.
Merge sort and quick sort are usually preferred if the data size is huge and for smaller data sets Bubble,Insertion or selection sort can be used.

# 9. References

For Datasets: http://archive.ics.uci.edu/ml/datasets/

For Sortedness Techniques:

http://jeshua.me/blog/MeasuringSortedness

http://stackoverflow.com/questions/16994668/is-there-a-way-to-measure-how-sorted-a-list-is