Project Report

TCSS598 Master Seminar LIJUN XUE

Summary

Project report of simulation of sorting techniques, including simulation and evaluation of 5 different sorting algorithms: **insertionsort, selectsort, bubblesort, mergesort, and quicksort**.

Soring Algorithms

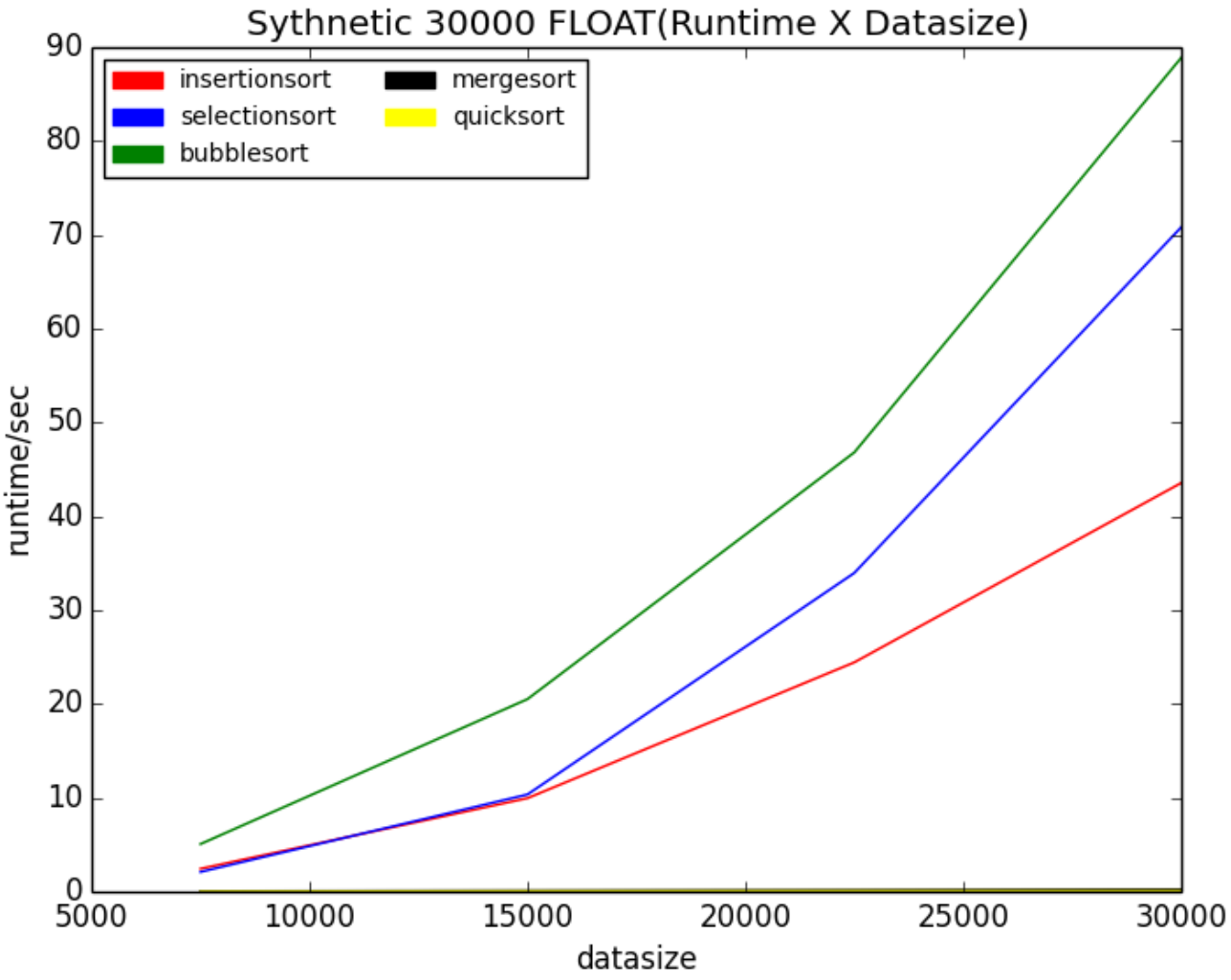
1. Insertion Sort: insertion sort is a simple algorithm that sort one item each time. The algorithm divide the array into 2 part, the sorted part, and the unsorted part, insertion sort with find on item in unsorted part and insert it into the sorted part.
2. Selection Sort: Selection sort also divide the array into sorted part and unsorted part, each time the algorithm will find the minimum(or maximum) item in unsorted part, and insert it into sorted part.
3. Bubble Sort: bubble sort do the things as its name, each time, it compare each items, and bubble the biggest one into one side of the array to form the sorted part.
4. Merge Sort: Merge Sort is a divide and conquer algorithm which divide the n length array into n single element, and merge them into bigger sorted array until there is only one sorted array left.
5. Quick Sort: Quick Sort is a powerful sorting algorithm, that each time select a random element as pivot, then separate elements bigger than pivot into a sub array, and others into another sub array

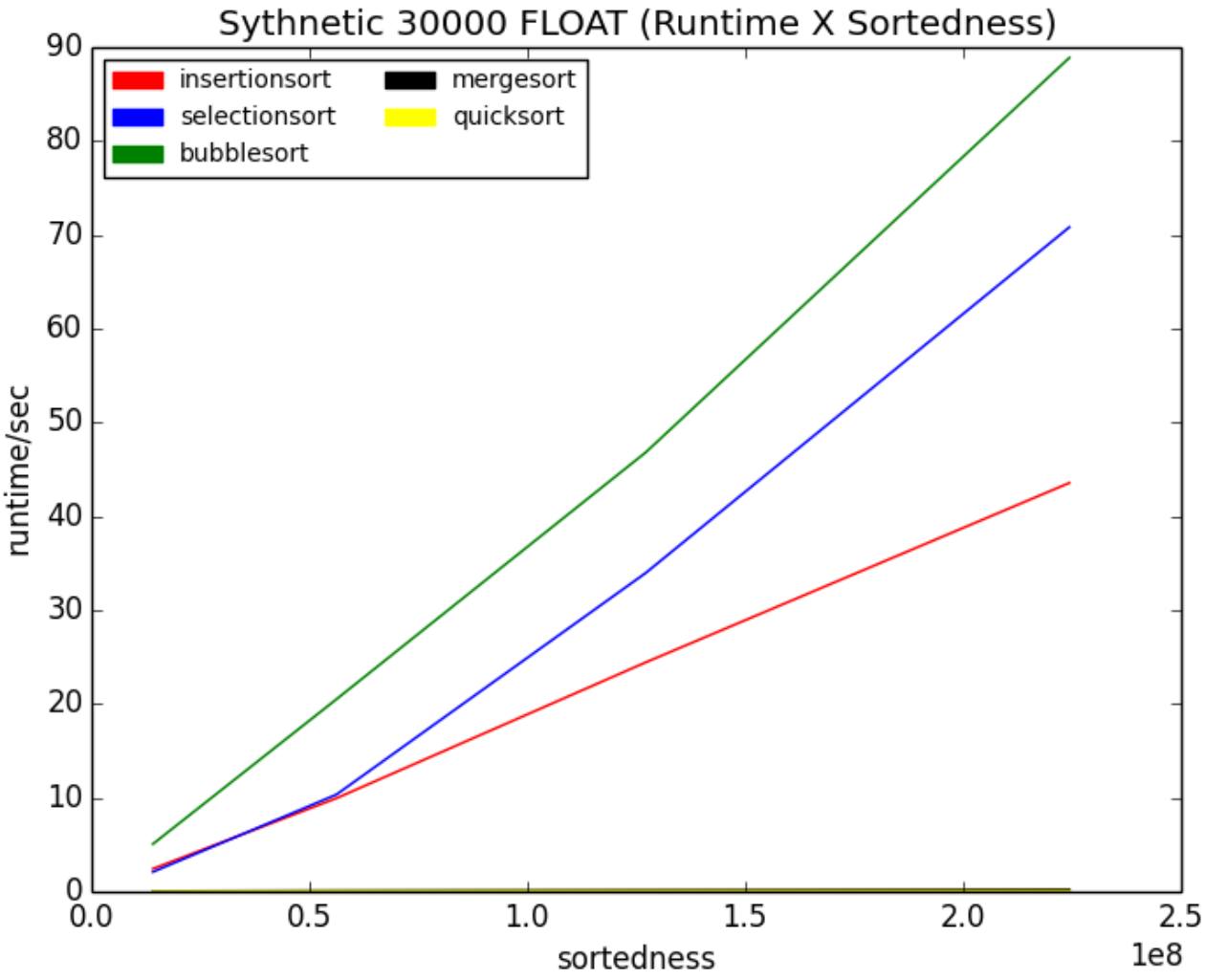
Datasets

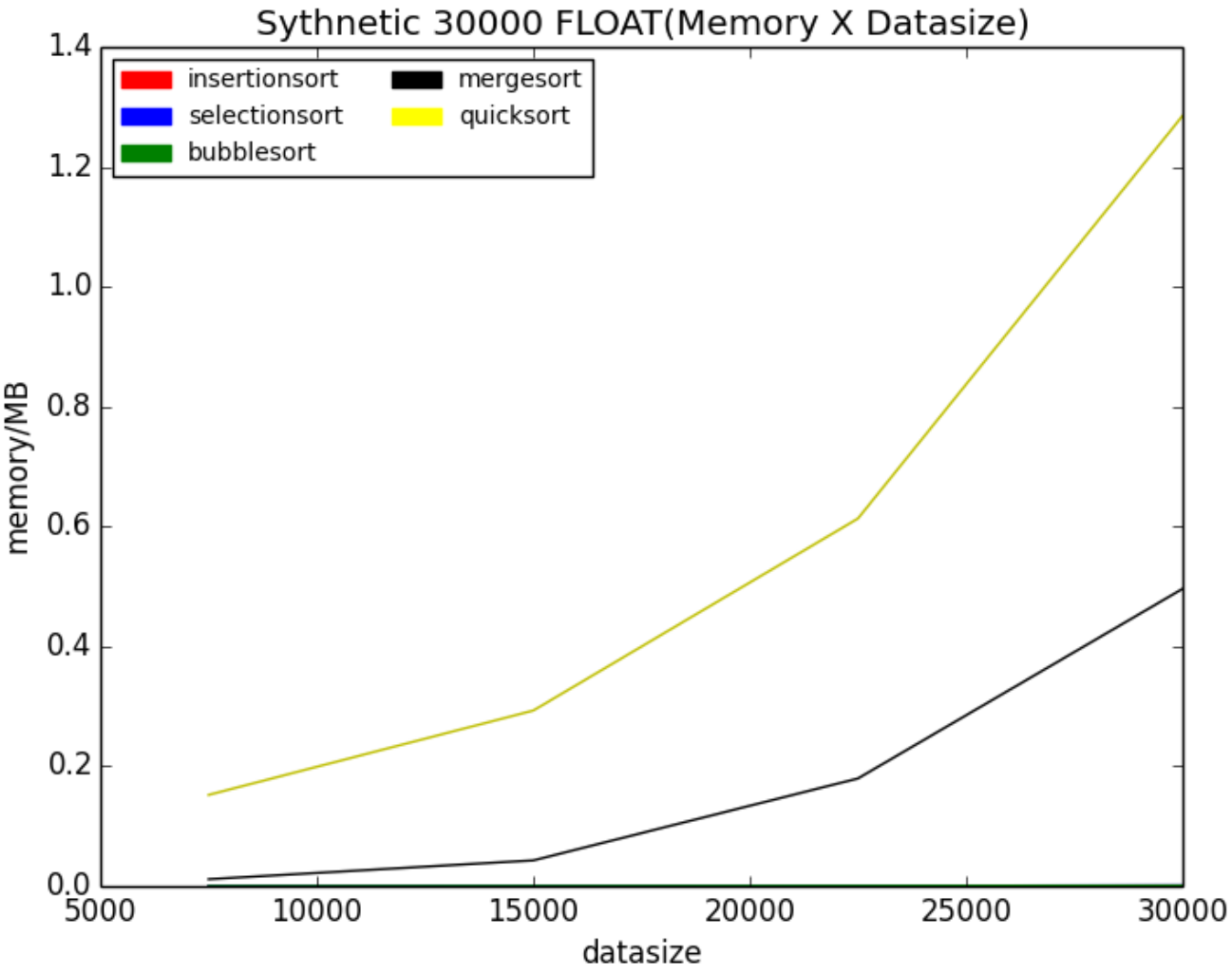
There are 4 datasets I use for the sorting algorithm simulation. The 2 synthetic datasets is one with 30000 float number range from 0.0 to 1.0, another one is with 50000 integer range from 0 to 99999. Also, for the different dataset, I generate them using different random seed in order to make datasets more random.

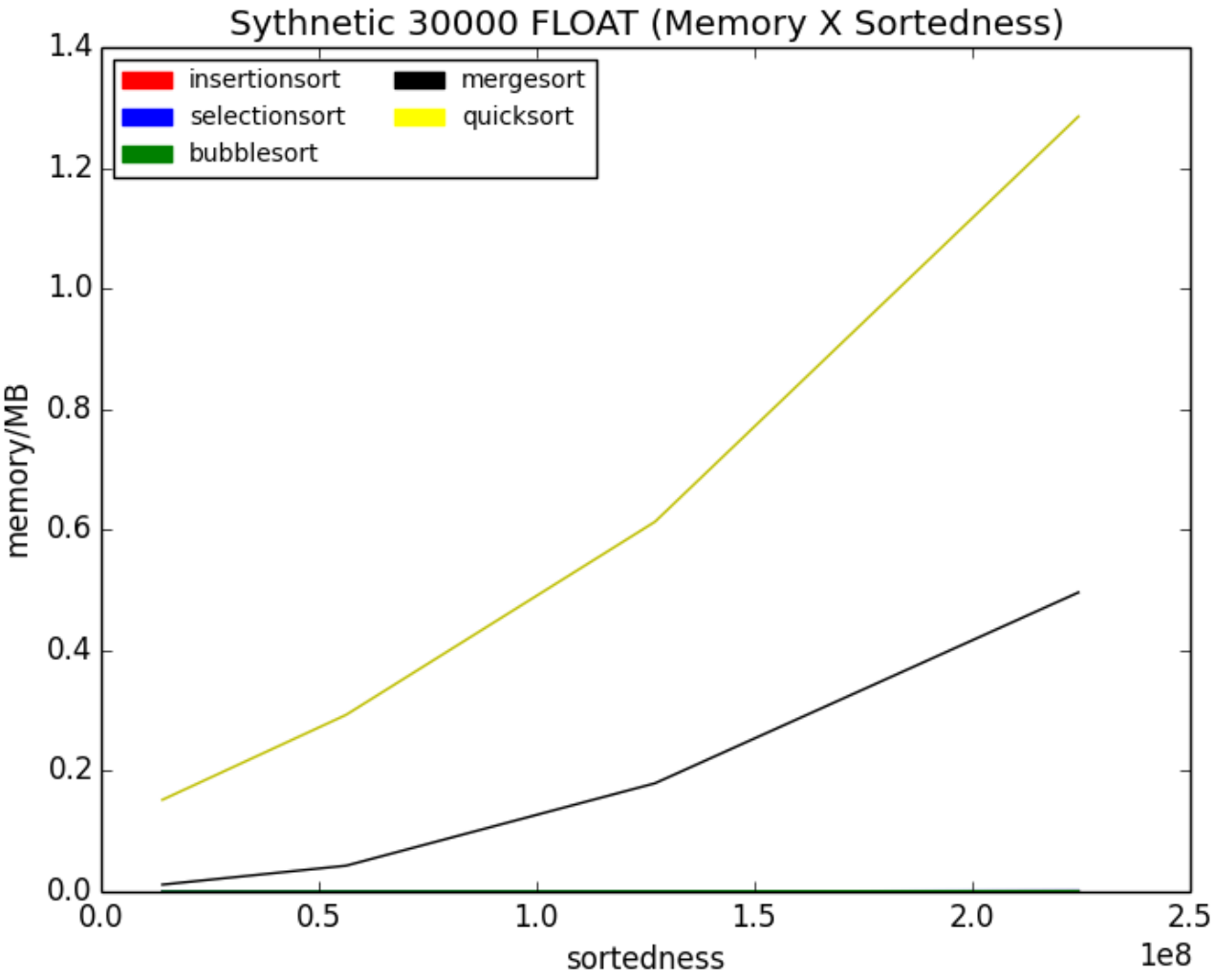
Another two real world dataset come from Quality Controlled Local Climatological Data(QCLCD), March 2015 Daily Monitoring data. The 2 columns I use is Tmax and Stn Pressure, which the first is a integer temperature value, the second is the float value range from 28 to 30. There are around 30000 data rows, however during extraction, I also need to process data, so finally i will get datasets less than 30000 rows.

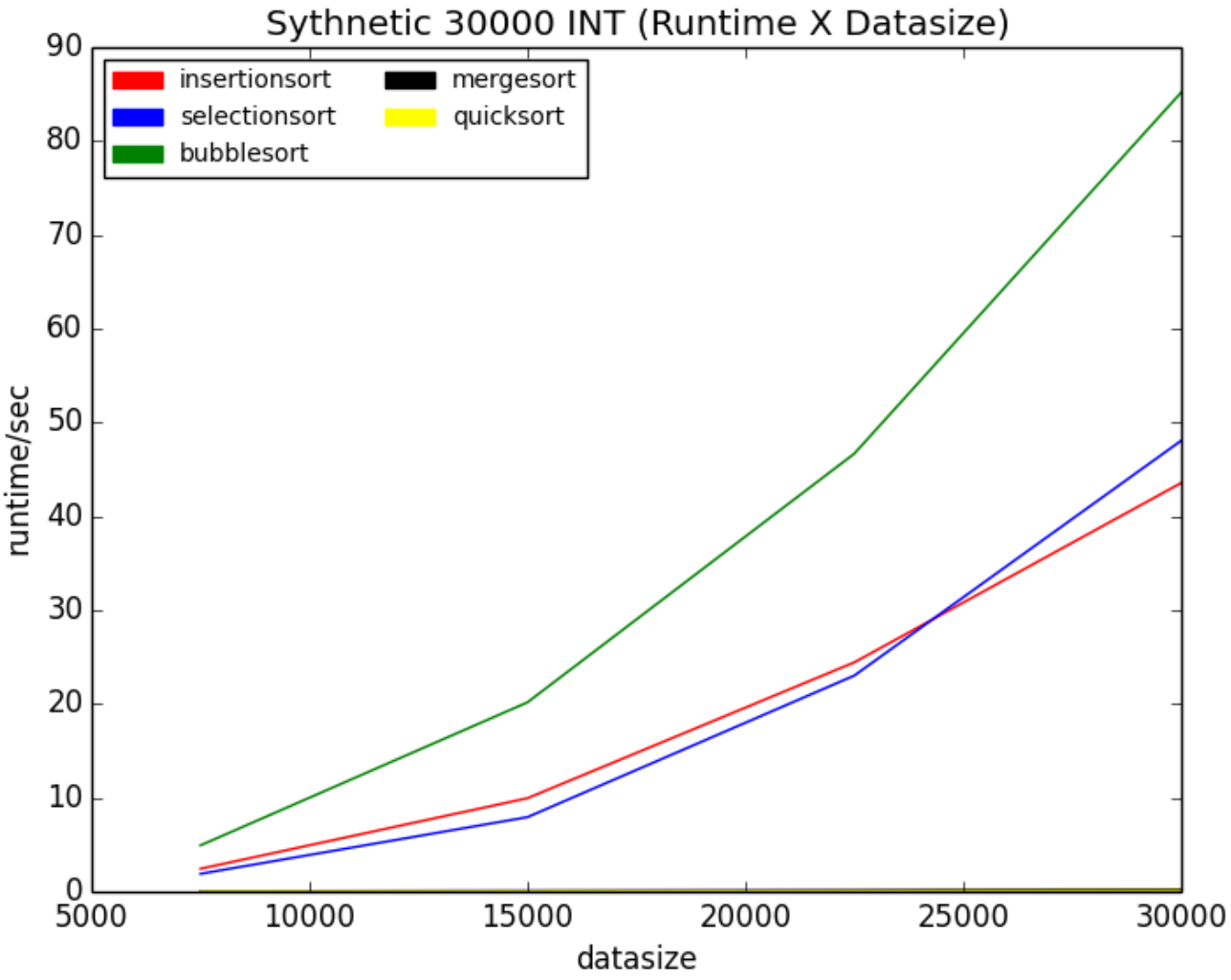
Performance Curve

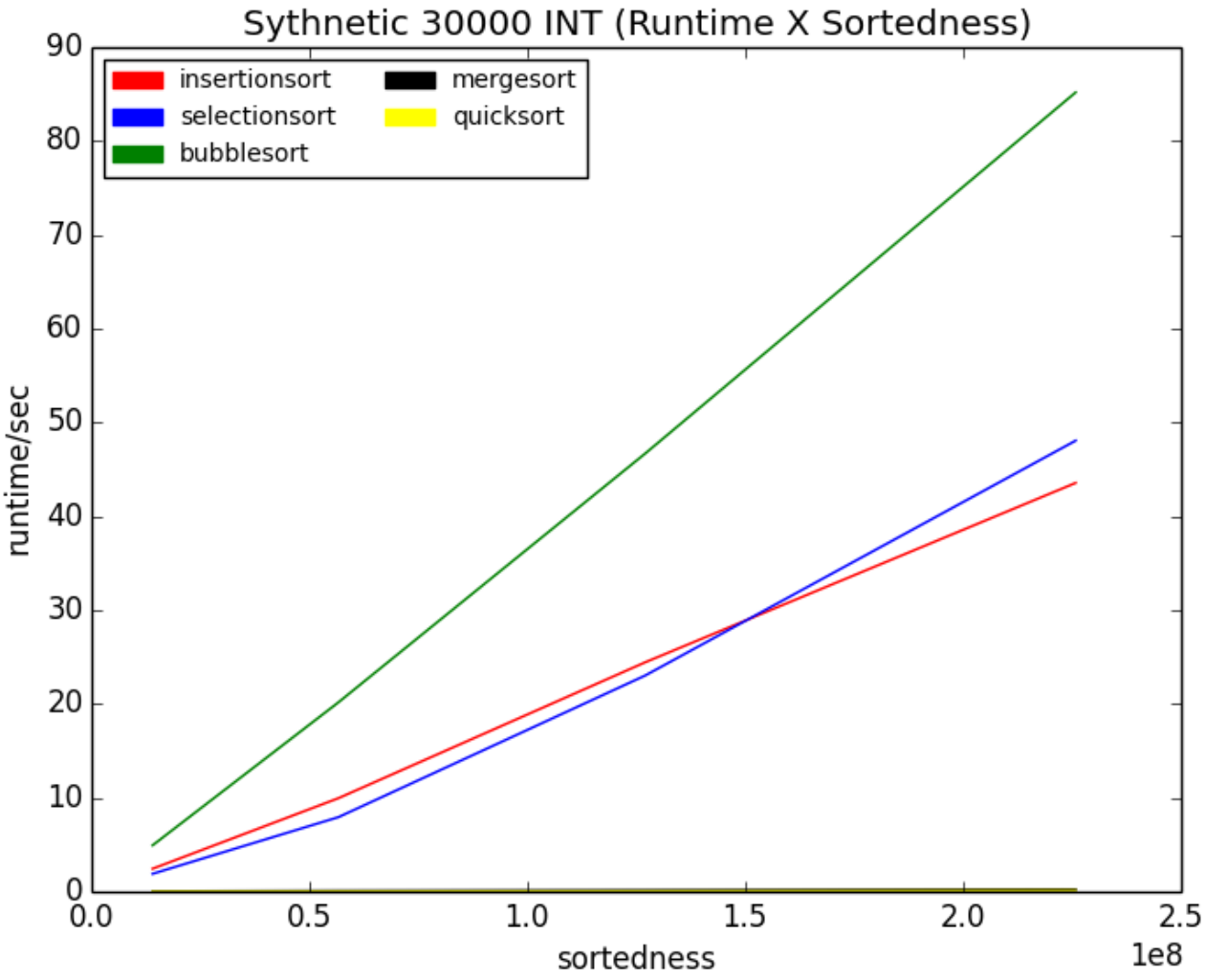

Sythnetic 30000 FLOAT(Runtime X Datasize)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
runtime/sec
datasize
0
10
20
30
40
50
60
70
80
90
5000
10000
15000
20000
25000
30000

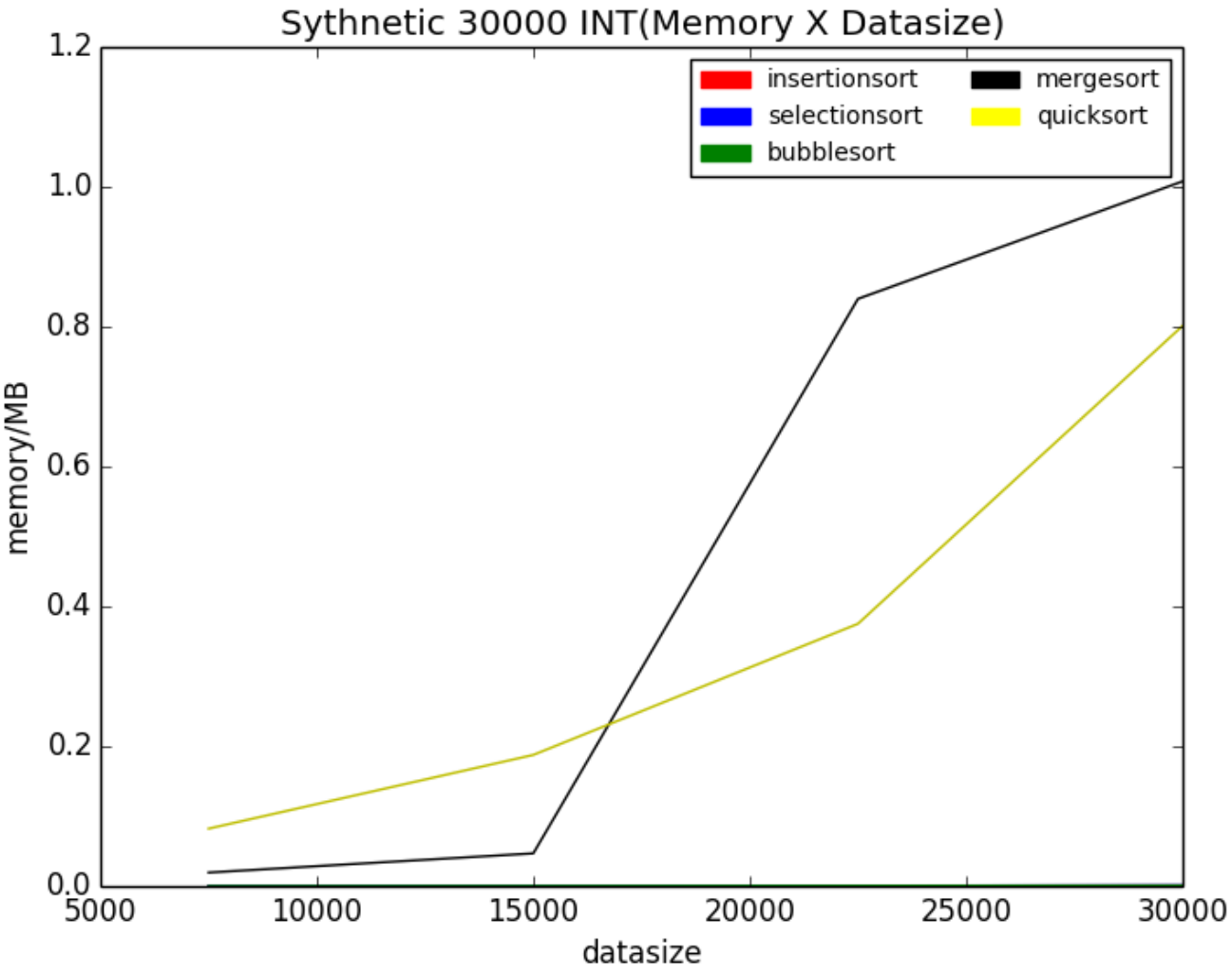

Sythnetic 30000 FLOAT (Runtime X Sortedness)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
runtime/sec
sortedness
1e8
0
10
20
30
40
50
60
70
80
90
0.0
0.5
1.0
1.5
2.0
2.5

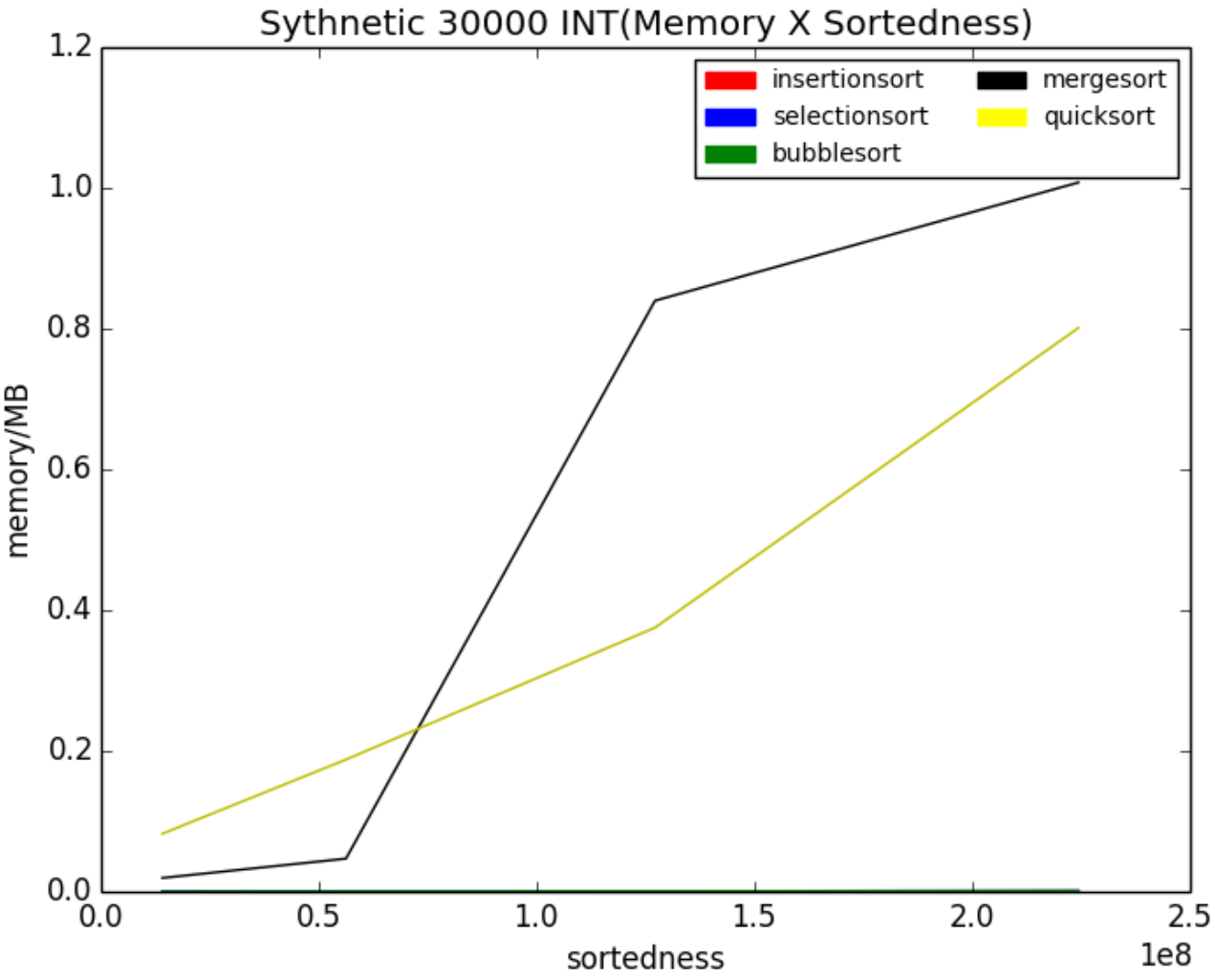

Sythnetic 30000 FLOAT(Memory X Datasize)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
memory/MB
datasize
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
5000
10000
15000
20000
25000
30000

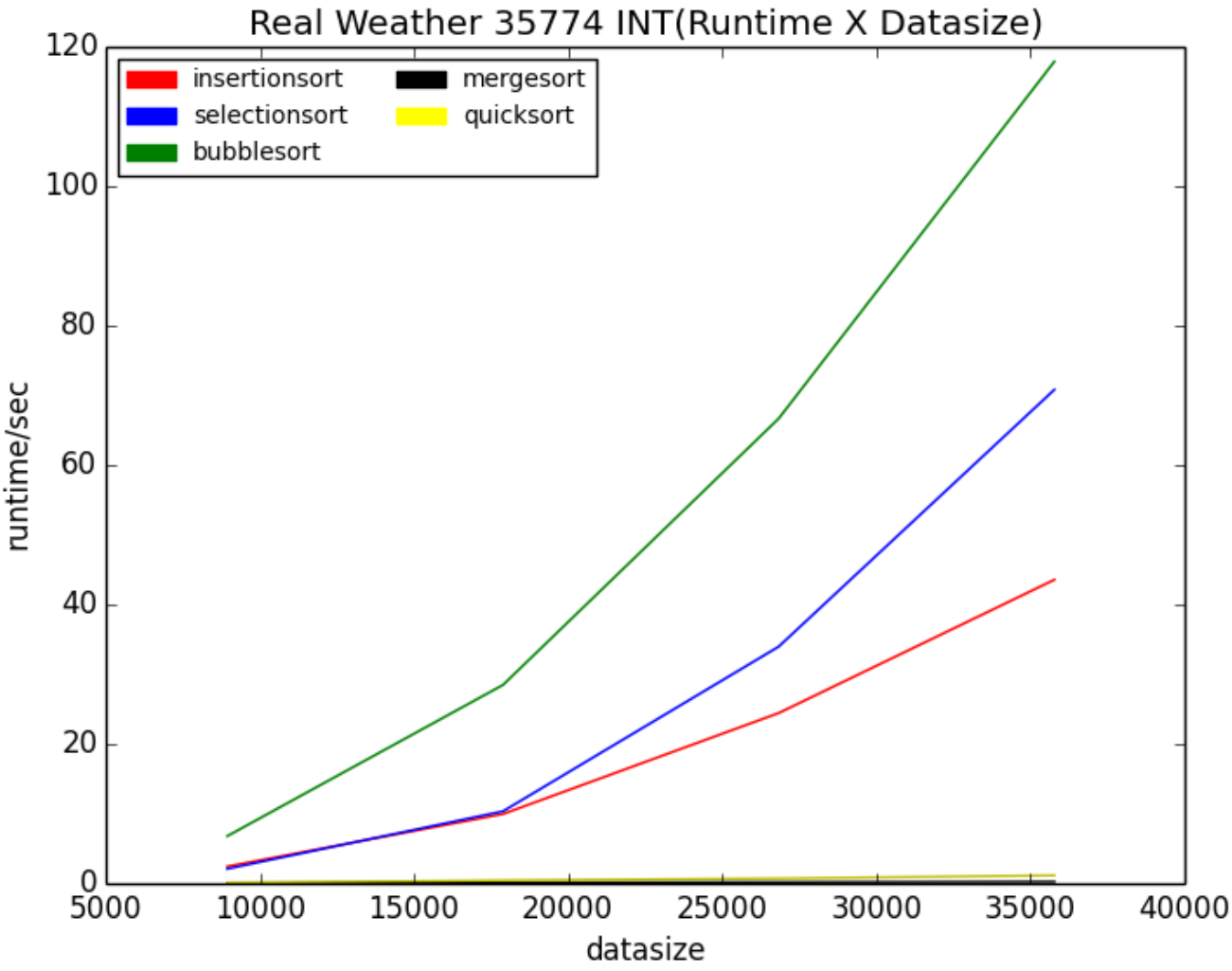

Sythnetic 30000 FLOAT (Memory X Sortedness)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
memory/MB
sortedness
1e8
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0.0
0.5
1.0
1.5
2.0
2.5

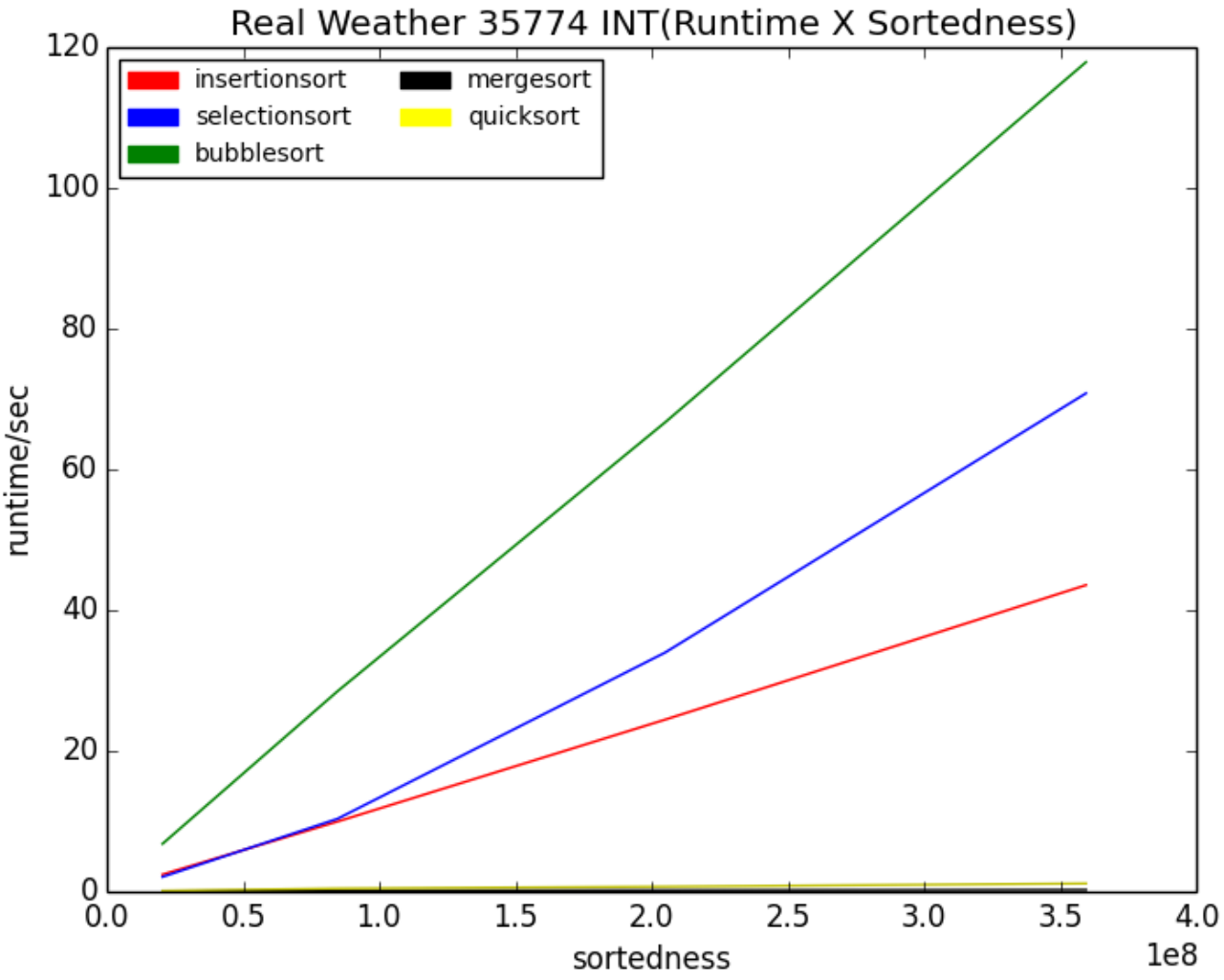

Sythnetic 30000 INT (Runtime X Datasize)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
runtime/sec
datasize
0
10
20
30
40
50
60
70
80
90
5000
10000
15000
20000
25000
30000

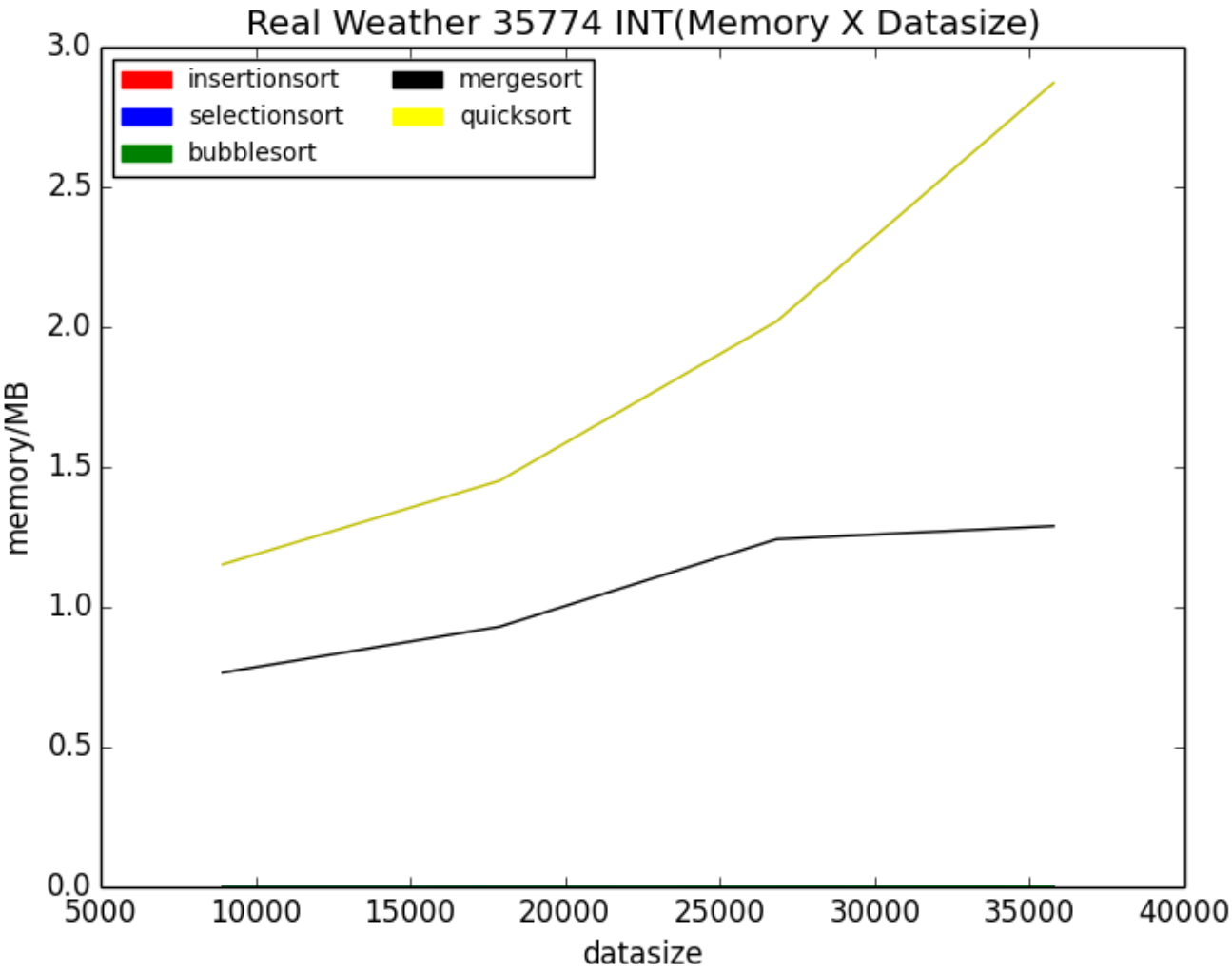

Sythnetic 30000 INT (Runtime X Sortedness)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
runtime/sec
sortedness
1e8
0
10
20
30
40
50
60
70
80
90
0.0
0.5
1.0
1.5
2.0
2.5

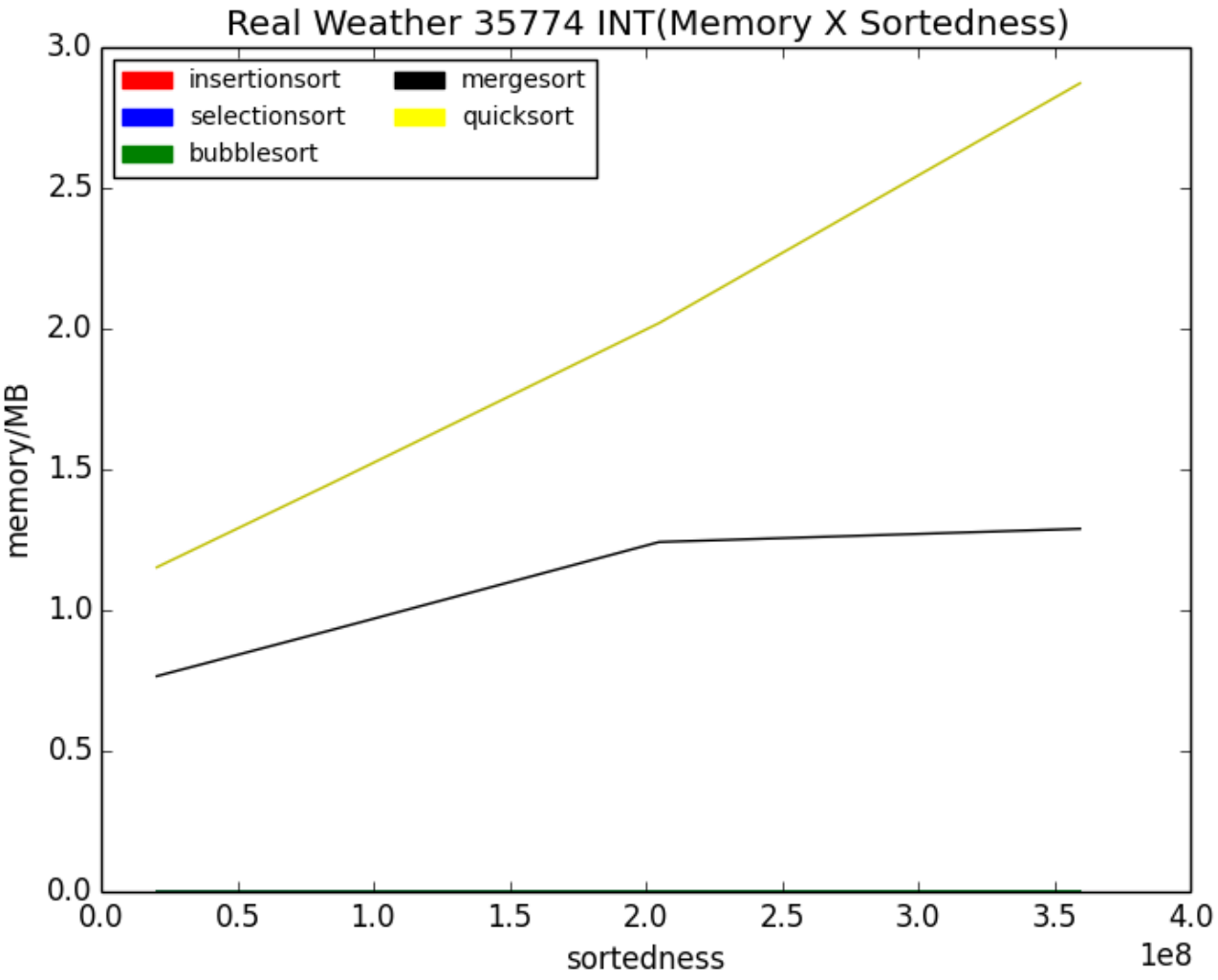

Sythnetic 30000 INT(Memory X Datasize)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
memory/MB
datasize
0.0
0.2
0.4
0.6
0.8
1.0
1.2
5000
10000
15000
20000
25000
30000

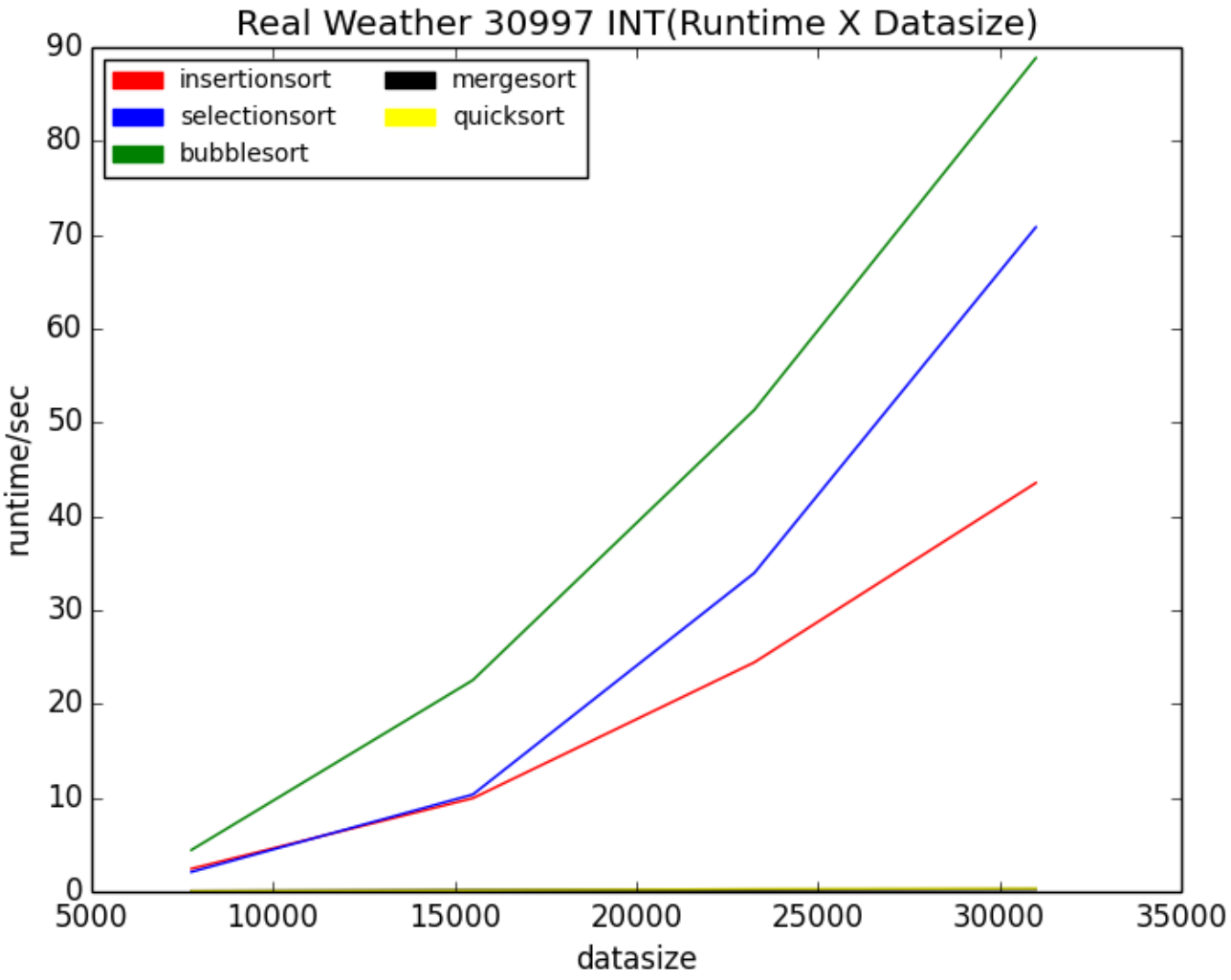

Sythnetic 30000 INT(Memory X Sortedness)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
memory/MB
sortedness
1e8
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.0
0.5
1.0
1.5
2.0
2.5

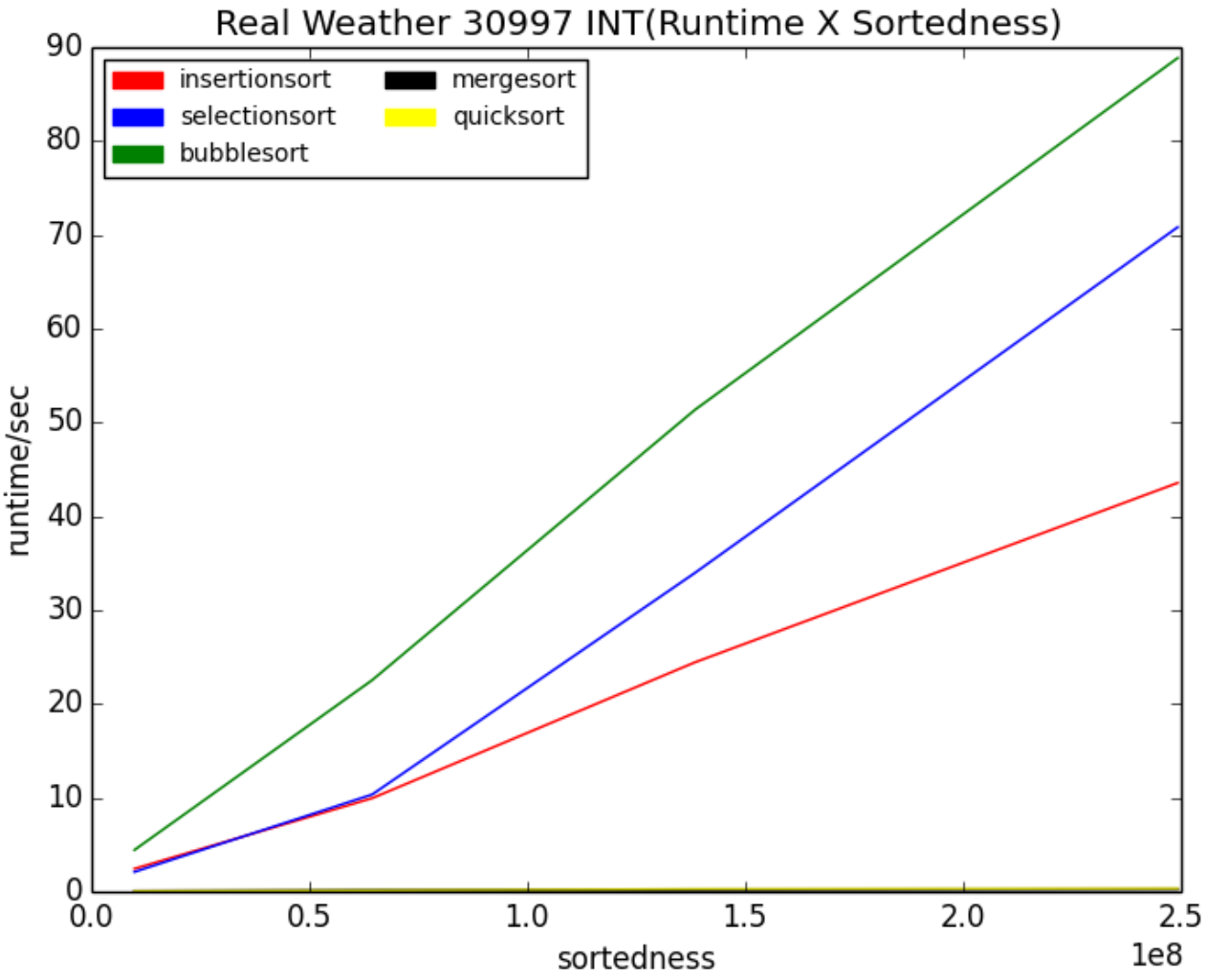

Real Weather 35774 INT(Runtime X Datasize)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
runtime/sec
datasize
0
20
40
60
80
100
120
5000
10000
15000
20000
25000
30000
35000
40000

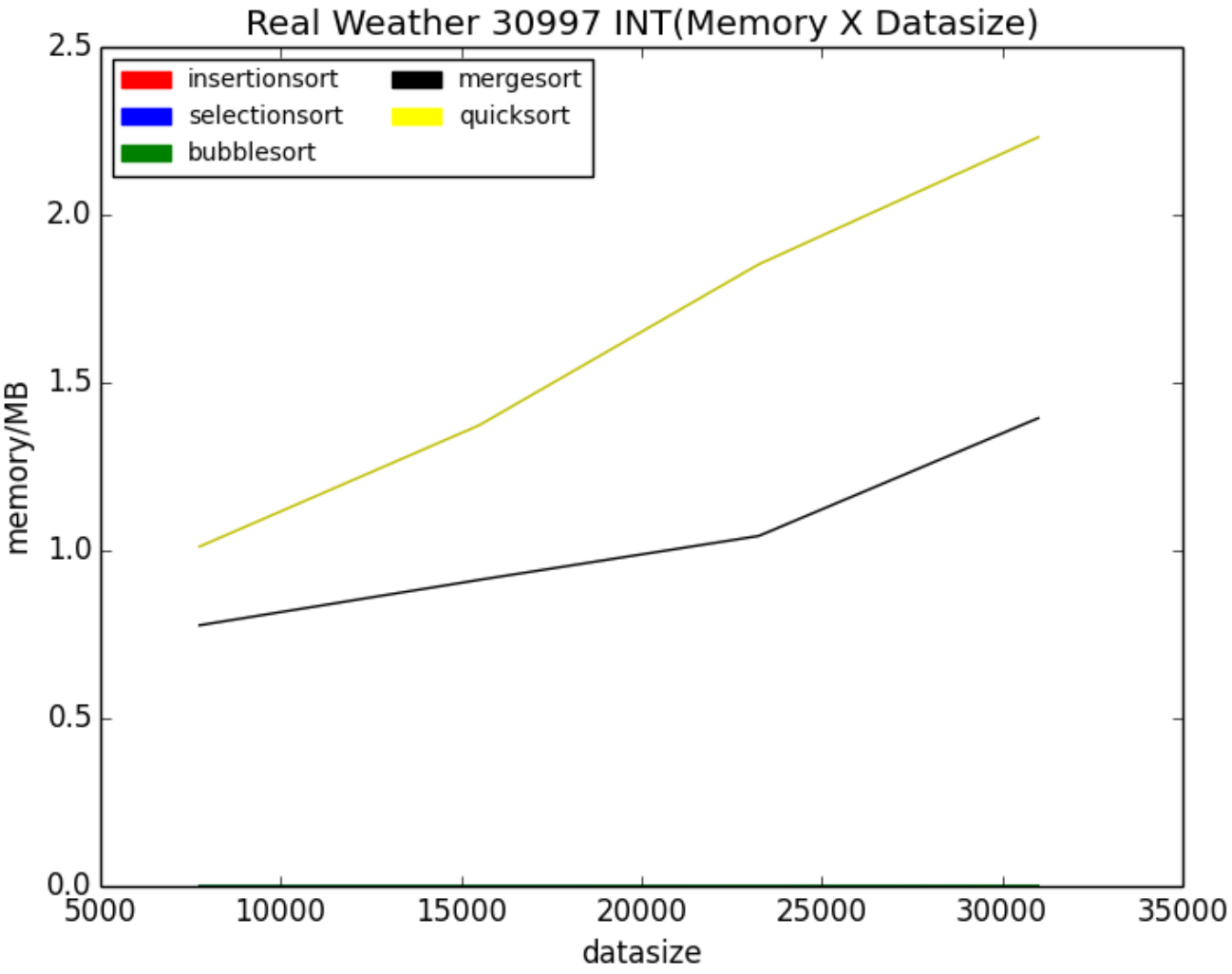

Real Weather 35774 INT(Runtime X Sortedness)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
runtime/sec
sortedness
1e8
0
20
40
60
80
100
120
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

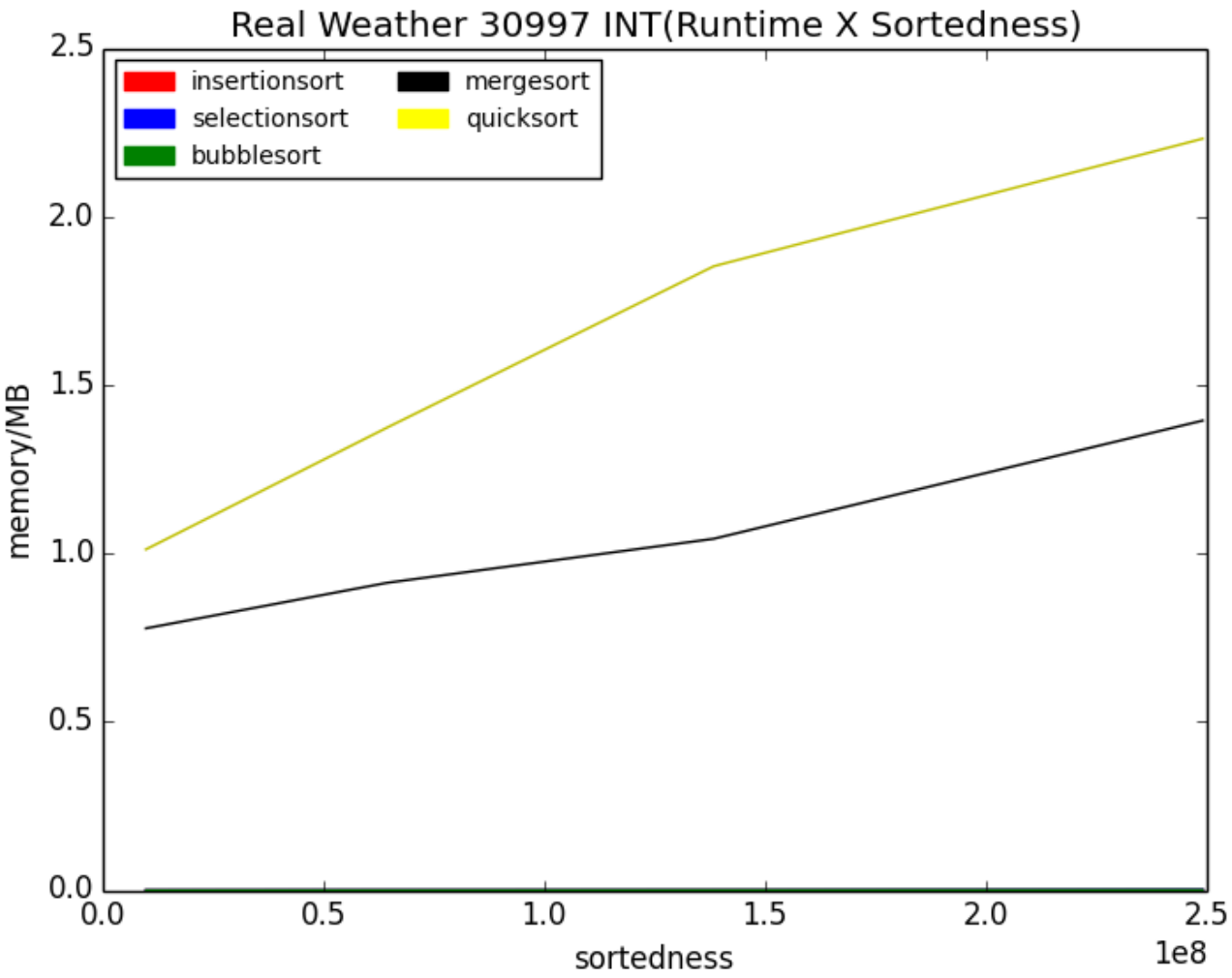

Real Weather 35774 INT(Memory X Datasize)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
memory/MB
datasize
3.0
2.5
2.0
1.5
1.0
0.5
0.0
5000
10000
15000
20000
25000
30000
35000
40000


Real Weather 35774 INT(Memory X Sortedness)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
memory/MB
sortedness
1e8
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Real Weather 30997 INT(Runtime X Datasize)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
runtime/sec
datasize
0
10
20
30
40
50
60
70
80
90
5000
10000
15000
20000
25000
30000
35000


Real Weather 30997 INT(Runtime X Sortedness)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
runtime/sec
sortedness
1e8
0
10
20
30
40
50
60
70
80
90
0.0
0.5
1.0
1.5
2.0
2.5


Real Weather 30997 INT(Memory X Datasize)
insertionsort
selectionsort
bubblesort
mergesort
quicksort
memory/MB
datasize
0.0
0.5
1.0
1.5
2.0
2.5
5000
10000
15000
20000
25000
30000
35000



Discussion

From the performance curve we can easily classify these sorting algorithm into 2 groups(or 3, if we put quick sort into quick but unstable algorithm). One is slow, but low memory usage algorithms, it has insertion sort, seletion sort, and bubble sort. Another one is quick but extra memory space needed algorithms, it has merge sort and quick sort. Different group has different features, for the first group, they all have O(n^2) runtime, and use O(1) space, which we can see on the runtime curves, they three rises rapidly, especially the bubble sort, but on the memory usage curves, they are almost a straight line near the x-axis. For another group is totally opposite, with runtime O(nlogn) and O(n)(mergesort), O(nlogn)(quicksort). Their runtime curves are the straght line close to x-axis, but for memory usage, rise very rapidly.

How to decide when to use which algorithm is due to the device performance, mainly related to the memory size, for small dataset you can use merge sort or quick sort in order to get result as soon as possible. However, if your dataset is huge, and device has low memory, it's better to use insertion sort, selection sort or bubble sort to get result.

Measure of Sortedness

The measure of sortedness is using the inversion number. Formally, let A be a sequence of n distinct numbers. If i < j and A(i) > A(j), then the pair (i, j) is called an inversion of A. The inversion number of a sequence is one common measure of its sortedness.

So the measure of sortedness I used improved merge sort function to calculate the sortedness in order to make the algorithm can finish in O(nlogn) time.

Reference: https://en.wikipedia.org/wiki/Inversion_(discrete_mathematics)

Extended Resources

Language: Python 2.7

Graph Drawing: Matplotlib 1.4.2

Memory Usage Counting: Memory Profiler 0.37

Feedbacks

For the graph, because of the hugh dataset, and different algorithm features, it makes some curves don't look very significantly(the value too low, nearly straight close to x-axis). I think if using smaller size database would be better for showing the result.