Riley Smith, Richard Hua, and Caitlin Feldewerth

Comparing Sorting Algorithms

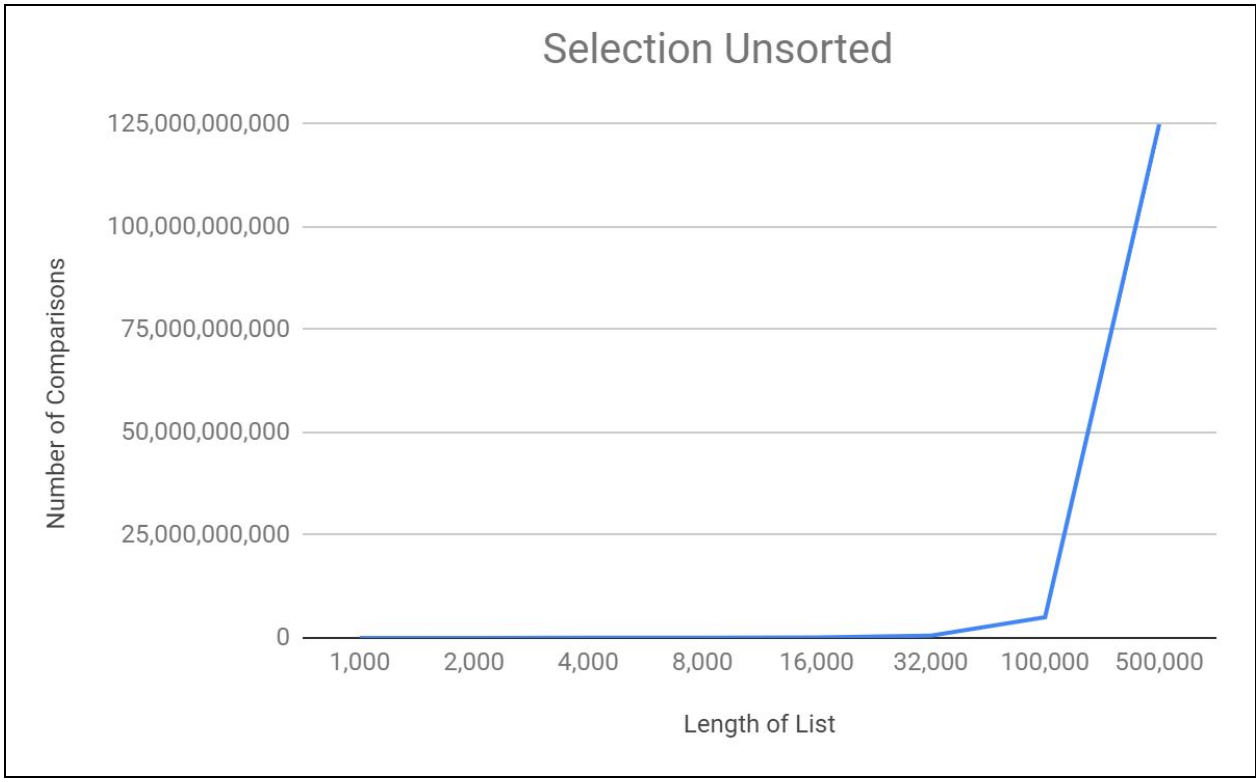
CPE 202-05

Toshihiro Kuboi

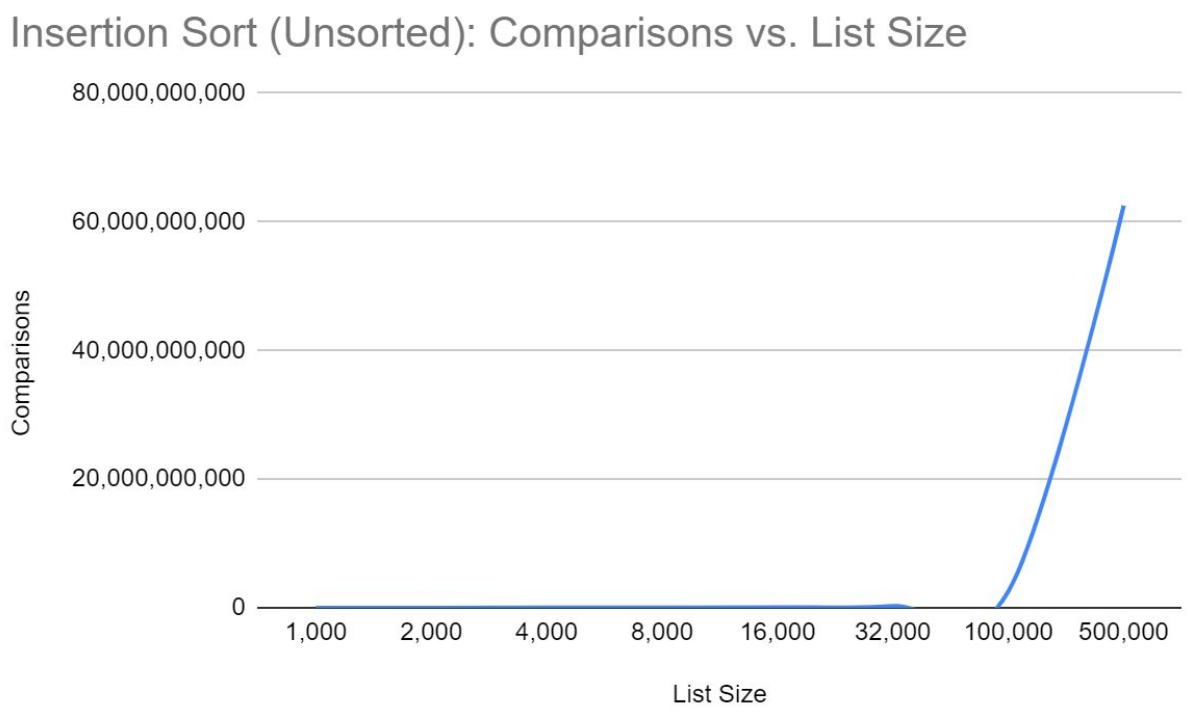
02/24/2020

Table of Contents

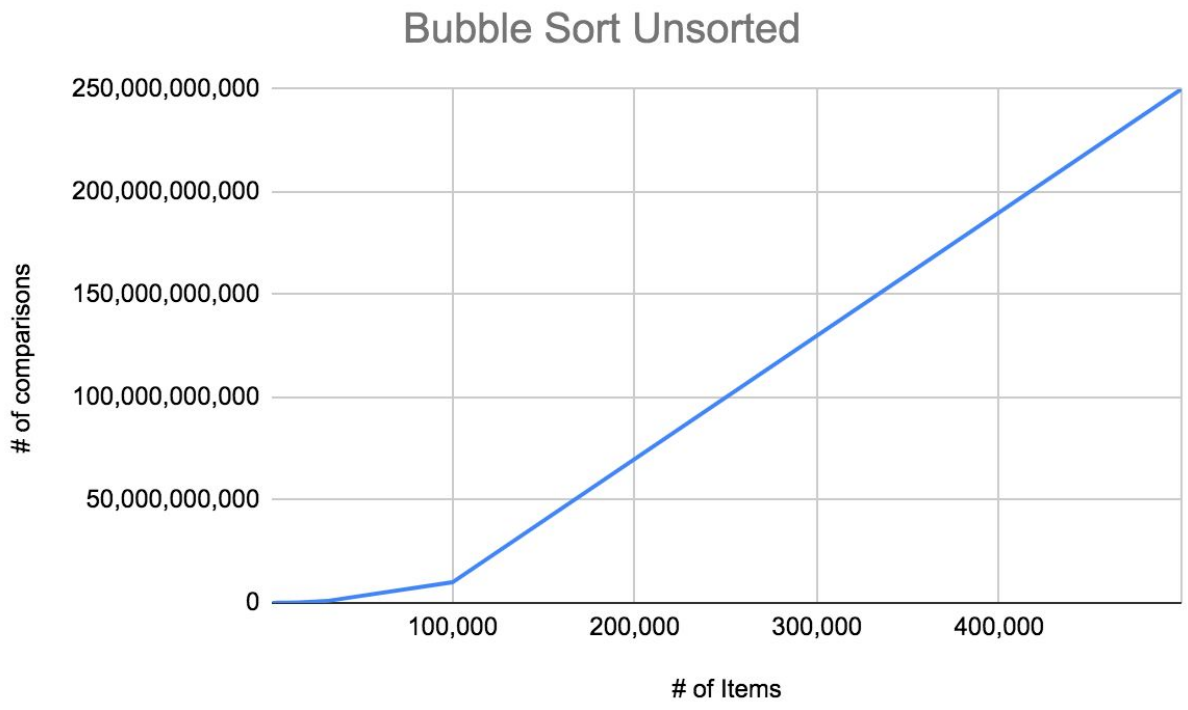
Selection Sort (Unsorted List)		
List Size	**Comparisons**	**Time (seconds)**
1,000	499,500	0.12432527542114258
2,000	1,999,000	0.5058417320251465
4,000	7,998,000	2.215327739715576
8,000	31,996,000	9.760668277740479
16,000	127,992,000	37.508360862731934
32,000	511,984,000	151.21382403373718
100,000	4,999,950,000	1688.3581149578094
500,000	124,999,750,000	142367.876892647



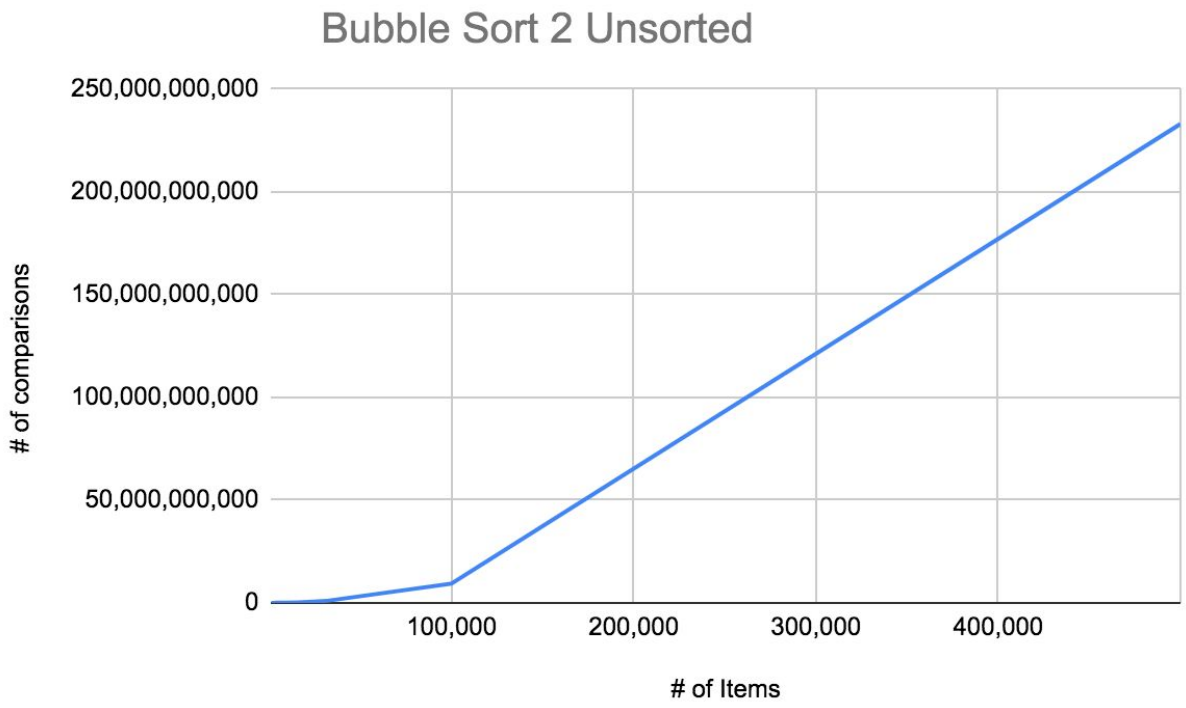
Insertion Sort (Unsorted List)		
List Size:	**Comparisons:**	**Time (seconds):**
1,000	249,655	0.258291
2,000	997,982	0.956803
4,000	4,050,919	4.302346
8,000	16,137,520	18.385138
16,000	64,488,700	109.607708
32,000	255,925,135	405.932389
100,000	2,499,656,027	3308.376714
500,000	62,508,426,921	16508.268954



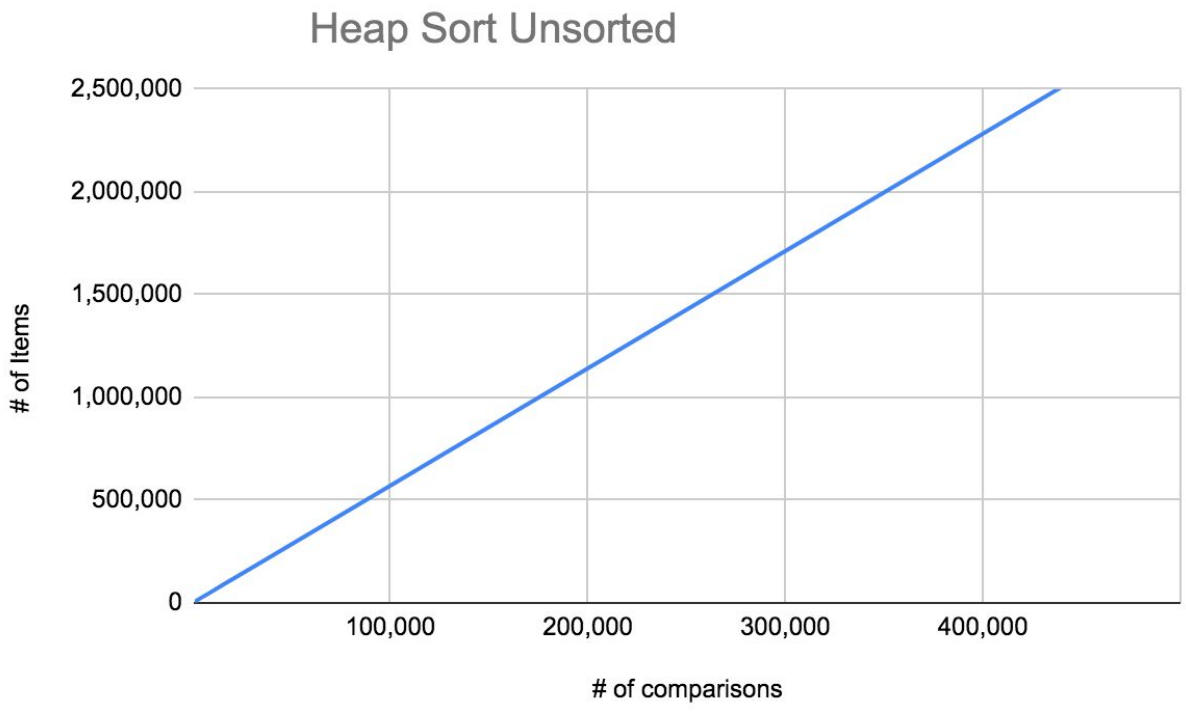
Bubble Sort (Unsorted List)		
List Size:	**Comparisons:**	**Time (seconds):**
1,000	999,000	0.4341693761935452
2,000	3,998,000	1.7618364513743854
4,000	15,996,000	7.28194356103456195
8,000	63,992,000	29.62397235690657
16,000	255,984,000	119.13540987364124
32,000	1,023,968,000	483.2349716351868
100,000	9,999,900,000	5424.1249764824312
500,000	249,999,500,000	135603.1241567658



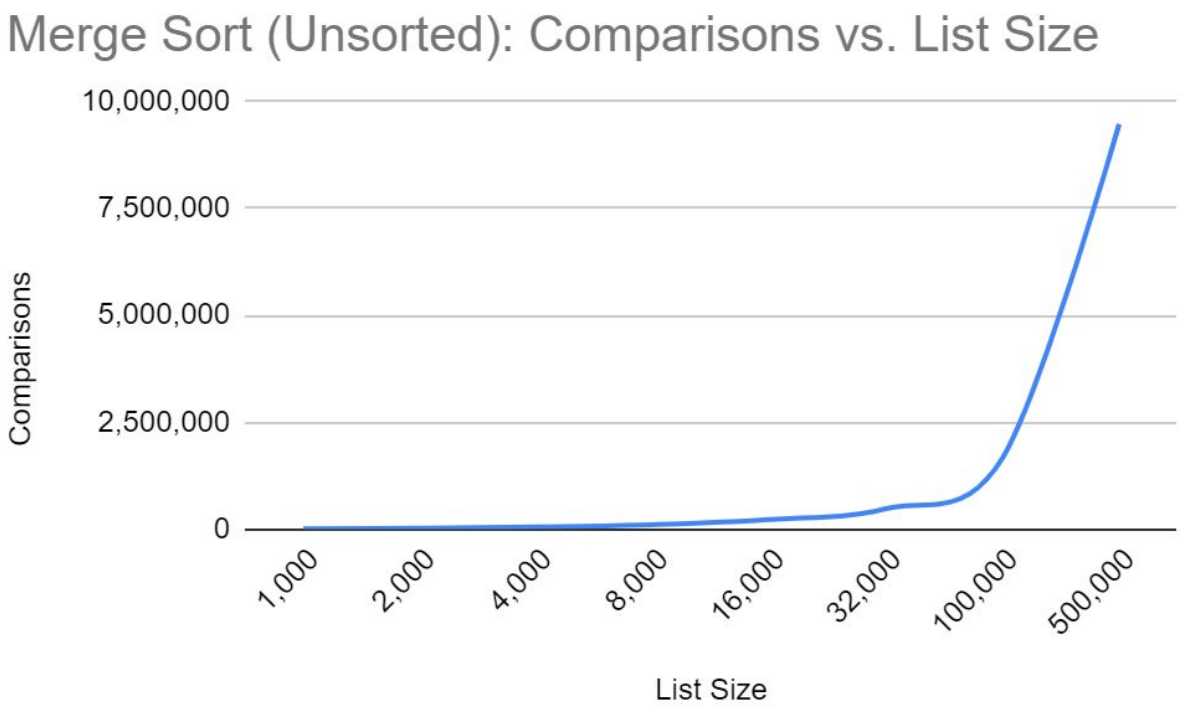
Bubble Sort 2 (Unsorted List)		
List Size:	**Comparisons:**	**Time (seconds):**
1,000	986,013	0.3227822780609131
2,000	3,886,056	1.2701349258422852
4,000	15,568,107	5.180327415466309
8,000	63,760,029	20.74586009979248
16,000	255,312,042	87.31421685218811
32,000	1,021,247,832	345.6104579543056
100,000	4,094,992,935	1392.5671208943568
500,000	16,732,556,256	5642.139756032345



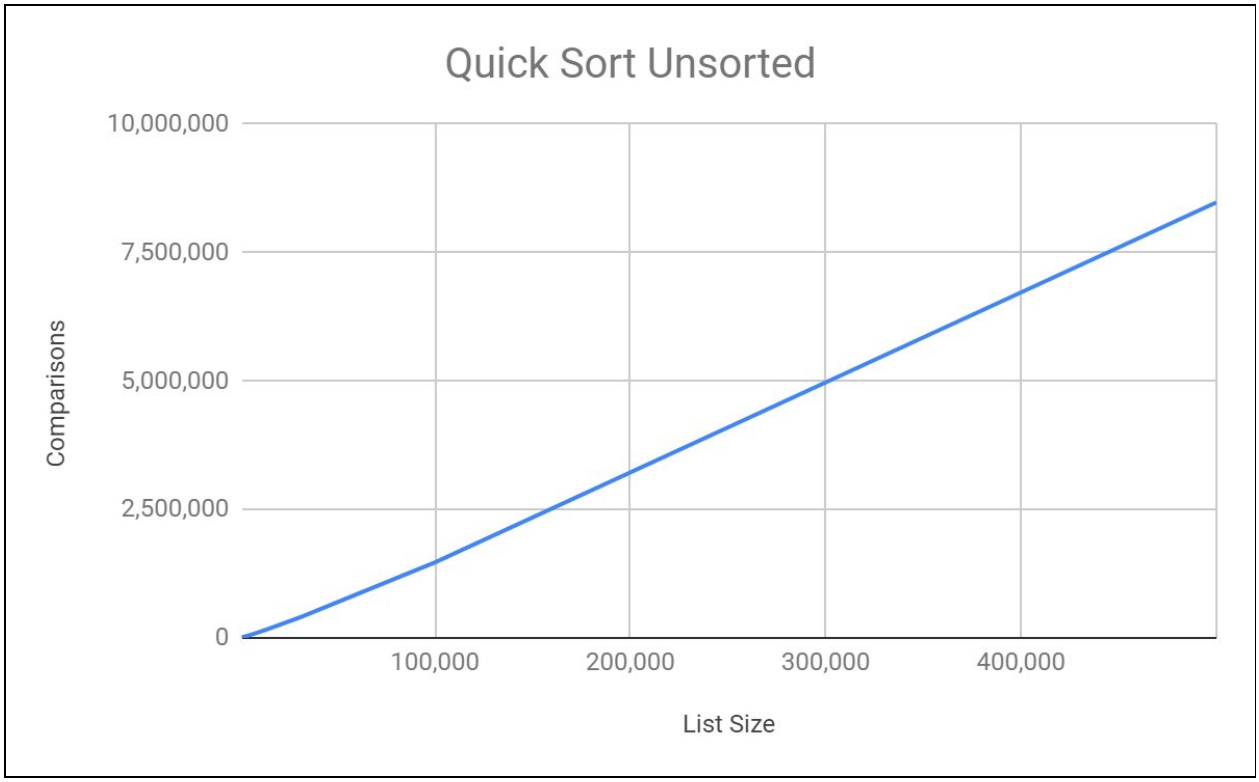
Heap Sort (Unsorted List)		
List Size:	**Comparisons:**	**Time (seconds):**
1,000	5,606	0.00319818450141411
2,000	11,350	0.0065862148754608
4,000	22,752	0.01308333083451245
8,000	45,476	0.02637082376846453
16,000	91,399	0.05358694974779544
32,000	182,732	0.1168546086964944
100,000	570,396	0.3683962813614809
500,000	2,853,444	1.9228098076857565



Merge Sort (Unsorted List)		
List Size:	**Comparisons:**	**Time (seconds):**
1,000	9,976	0.008014
2,000	21,952	0.013962
4,000	47,904	0.028996
8,000	103,808	0.063829
16,000	223,616	0.134565
32,000	479,232	0.321771
100,000	1,668,928	1.102915
500,000	9,475,712	6.330691

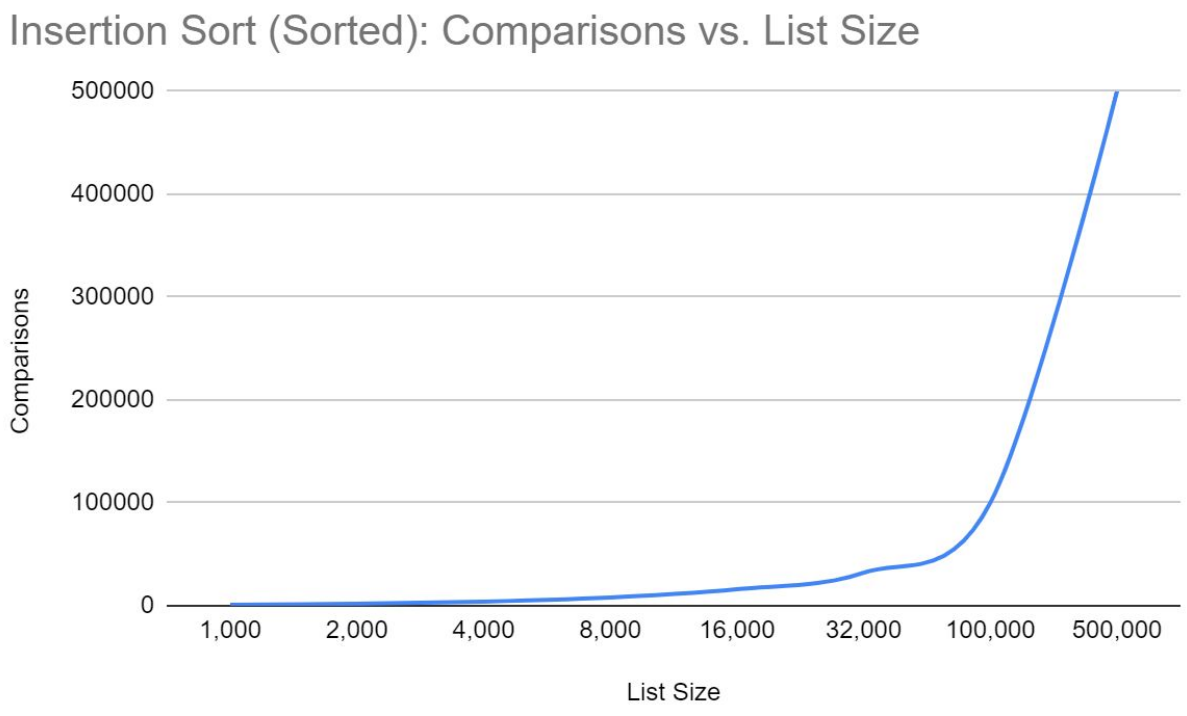


Quick Sort (Unsorted List)		
List Size	**Comparisons**	**Time (seconds)**
1,000	7,987	0.010088920593261719
2,000	17,964	0.028272628784179688
4,000	39,917	0.039319515228271484
8,000	87,822	0.09403586387634277
16,000	191,631	0.19470667839050293
32,000	415,248	0.45737528800964355
100,000	1,468,946	1.4908151626586914
500,000	8,475,732	7.952672243118286

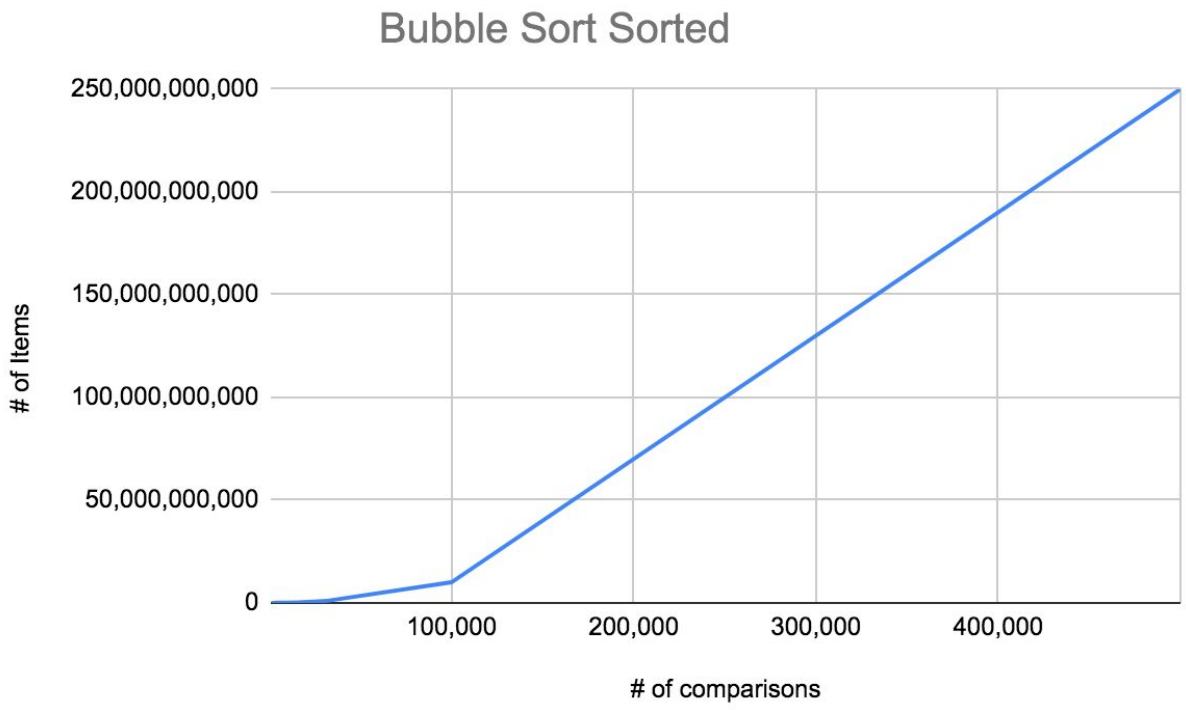


Selection Sort (Sorted List)		
List Size	**Comparisons**	**Time (seconds)**
1,000	499,500	0.1512000560760498
2,000	1,999,000	0.482633113861084
4,000	7,998,000	2.4250333309173584
8,000	31,996,000	8.644640684127808
16,000	127,992,000	36.27250647544861
32,000	511,984,000	139.73194479942322
100,000	4,999,950,000	1413.14169383049
500,000	124,999,750,000	16375.4563721901

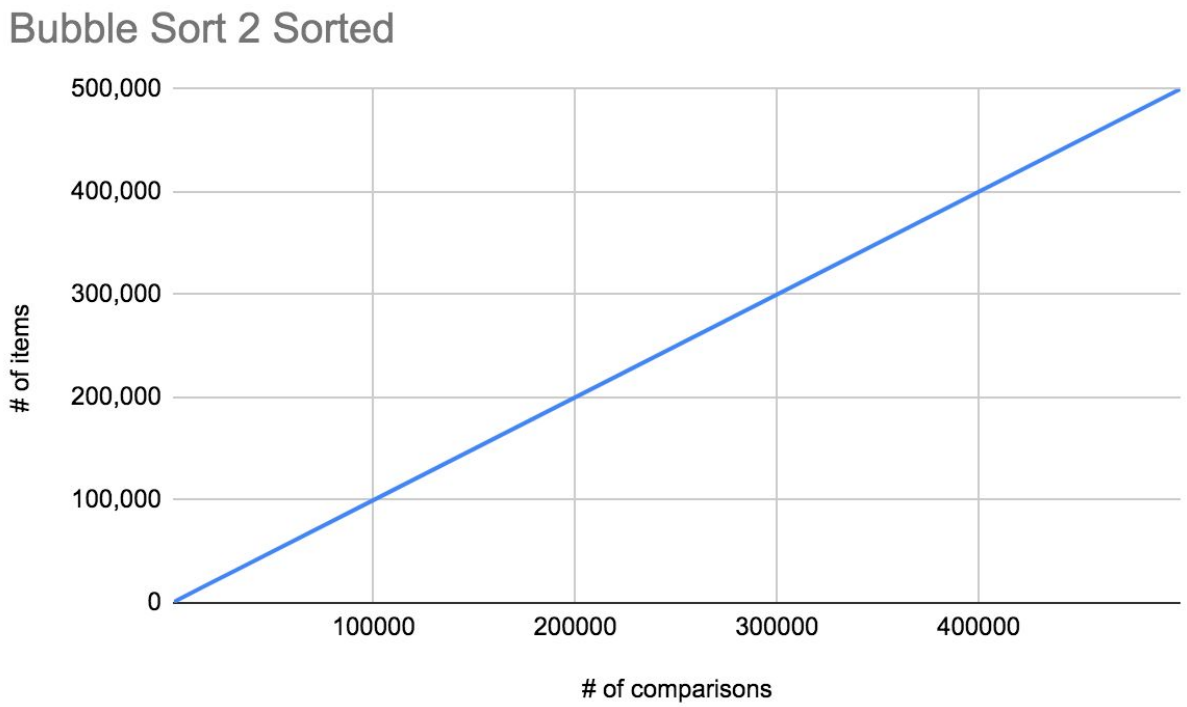
Insertion Sort (Sorted List)		
List Size:	**Comparisons:**	**Time (seconds):**
1,000	999	0.001137
2,000	1,999	0.000997
4,000	3,999	0.001597
8,000	7,999	0.002609
16,000	15,999	0.006981
32,000	31,999	0.015958
100,000	99,999	0.026966
500,000	499,999	0.173655



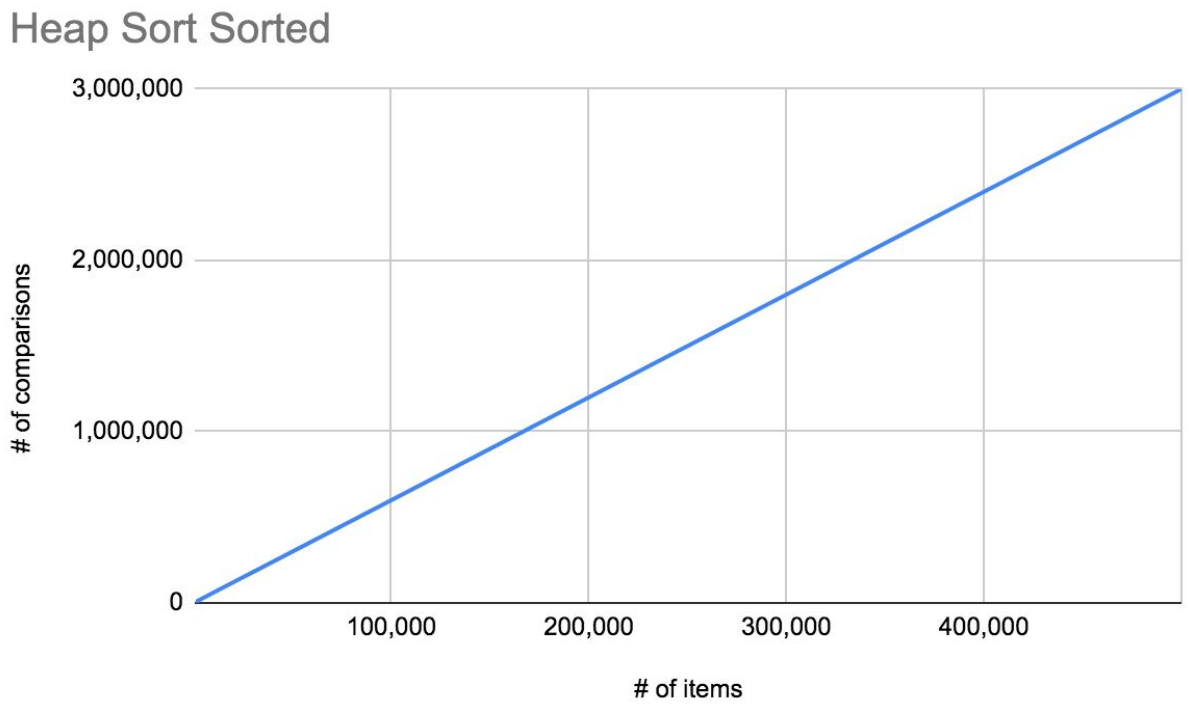
Bubble Sort (Sorted List)		
List Size:	**Comparisons:**	**Time (seconds):**
1,000	999,000	0.29463624954223633
2,000	3,998,000	1.2923145294189453
4,000	15,996,000	5.601490020751953
8,000	63,992,000	20.136396884918213
16,000	255,984,000	81.1761257648468
32,000	1,023,968,000	327.3061249256134
100,000	9,999,900,000	3196.347854857544
500,000	249,999,500,000	79908.696544569032



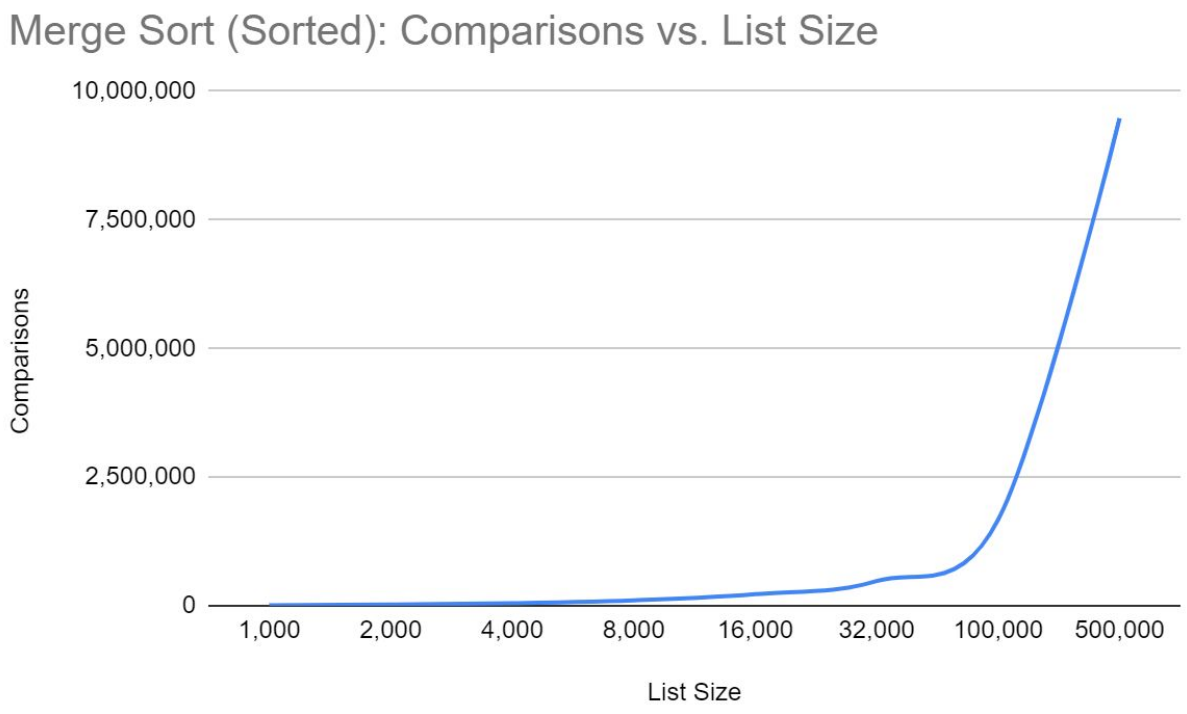
Bubble Sort 2 (Sorted List)		
List Size:	**Comparisons:**	**Time (seconds):**
1,000	999	0.0002503395080566406
2,000	1,999	0.0004491806030273437 5
4,000	3,999	0.0009319782257080078
8,000	7,999	0.0017287731170654297
16,000	15,999	0.0036623477935791016
32,000	31,999	0.011517524719238281
100,000	99,999	0.03772878646850586
500,000	499,999	0.30599188804626465



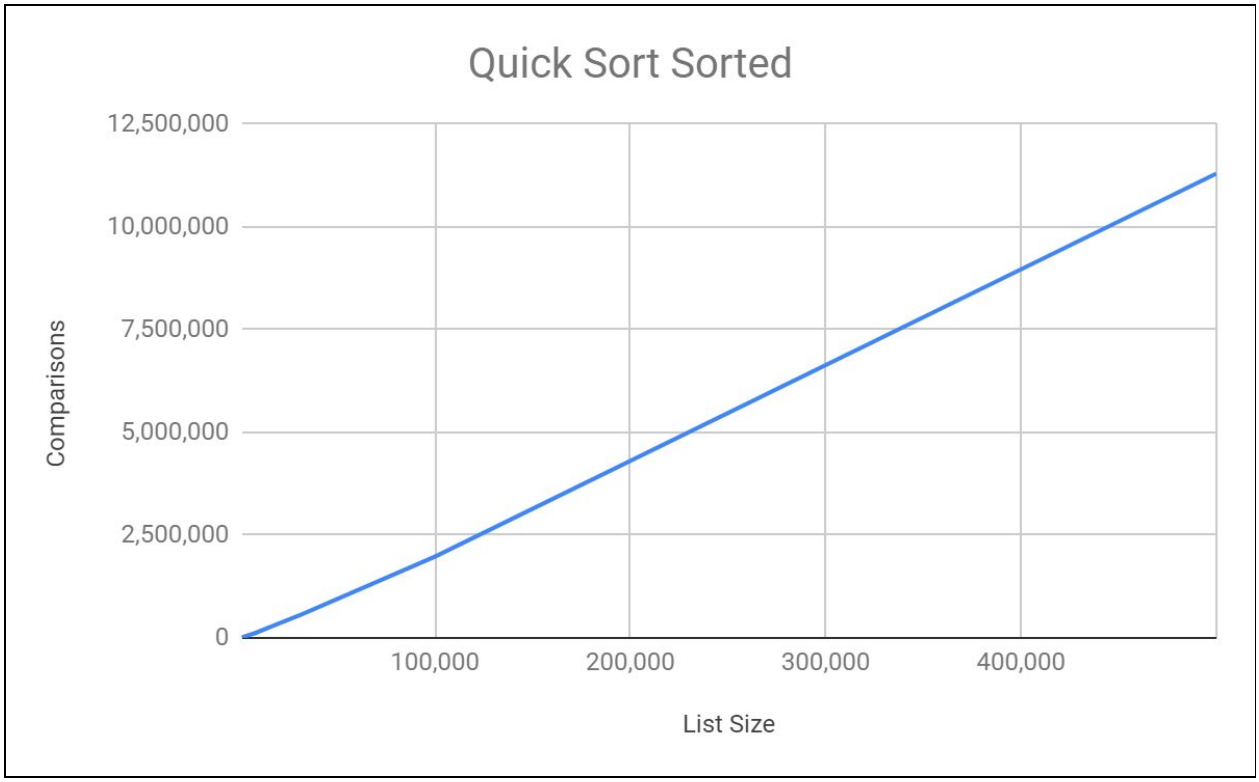
Heap Sort (Sorted List)		
List Size:	**Comparisons:**	**Time (seconds):**
1,000	5,947	0.0035741329193115234
2,000	11,941	0.005722999572753906
4,000	23,935	0.011286497116088867
8,000	47,929	0.022029876708984375
16,000	95,923	0.0473935604095459
32,000	191,917	0.09520626068115234
100,000	599,905	0.27698397636413574
500,000	2,999,893	1.8897888660430908



Merge Sort (Sorted List)		
List Size:	**Comparisons:**	**Time (seconds):**
1,000	9,976	0.004986
2,000	21,952	0.011969
4,000	47,904	0.024452
8,000	103,808	0.048869
16,000	223,616	0.107174
32,000	479,232	0.230814
100,000	1,668,928	0.938532
500,000	9,475,712	5.310837



Quick Sort (Sorted List)		
List Size	**Comparisons**	**Time (seconds)**
1,000	11,515	0.009360074996948242
2,000	23,235	0.019350290298461914
4,000	56,534	0.036003828048706055
8,000	117,266	0.0812985897064209
16,000	271,525	0.2079460620880127
32,000	573,524	0.3598630428314209
100,000	1,976,867	1.3532264232635498
500,000	11,296,579	7.721241235733032



1. **Which sort do you think is better? Why?**

   I think that Heap sort is the best because it is O(nlogn) in both worst and best cases and it also has a space complexity of O(1). In our trails it was a few seconds faster than both Merge sort and Quick sort in the 500,000 item trial. The only drawback that I see with Heap sort is that it is not stable, although in this lab stability was not something that was important since we were only sorting simple integers.

2. **Which sort is better when sorting a list that is already sorted (or mostly sorted)? Why?**

   When a list is nearly sorted completely sorted, insertion sort and bubble sort 2 are the best algorithms to use. Insertion sort is better for a nearly sorted list because it does not need to run through n numbers to sort each item, so when the list is nearly sorted it can be closer to O(n) time than its worst case of O(n^2). When we have a sorted list, the best algorithm to use would be bubble sort 2. This is because bubble sort 2 can recognize when it goes through an iteration and no swaps were made. This means that it can sort the list in O(n) time.

3. **You probably found that insertion sort had about as many comparisons as selection sort. Why? Why are the times for insertion sort not half what they are for selection sort?**

   The reason why insertion sort can finish sorting with so many less comparisons than selection sort is because insertion sort does not always go through the n numbers in the array when it picks a number to insert into the sorted portion of the list. It will on average only go through n/2 items before inserting an item to a sorted portion of the list.