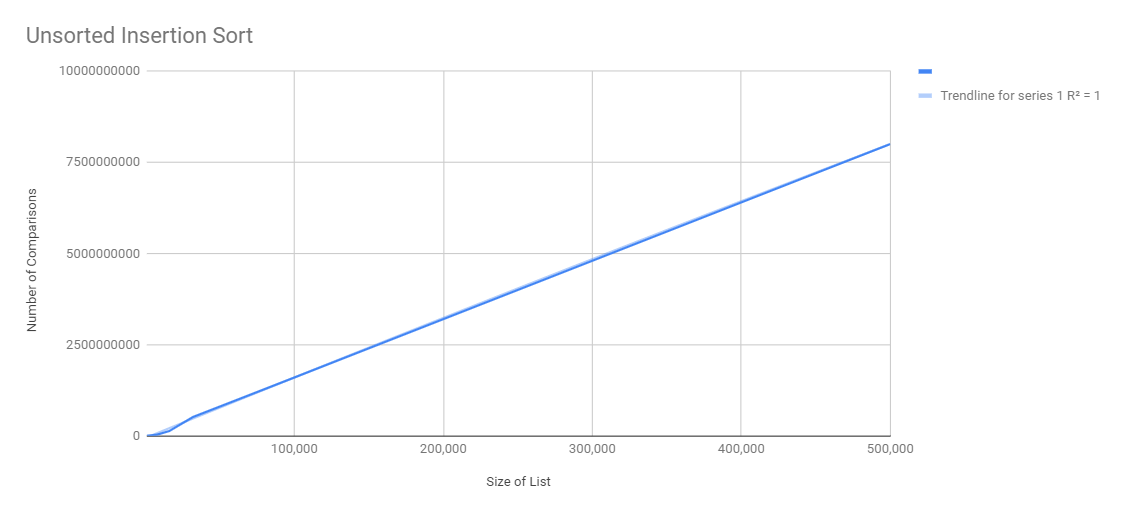
|  |  |  |
| --- | --- | --- |
| List Size | Comparisons | Time(seconds) |
| 1,000 | 251412 | 0.05385589599609375 |
| 2,000 | 1008885 | 0.21342873573303223 |
| 4000 | 3995350 | 0.8512387275695801 |
| 8000 | 16168433 | 3.4983673095703125 |
| 16000 | 63910064 | 13.819163799285889 |
| 32000 | 255686605 | 55.606541872024536 |
| 100,000(estimated) | 799020640 | 173.7704434 |
| 500,000(estimated) | 3885103203 | 868.8522168 |

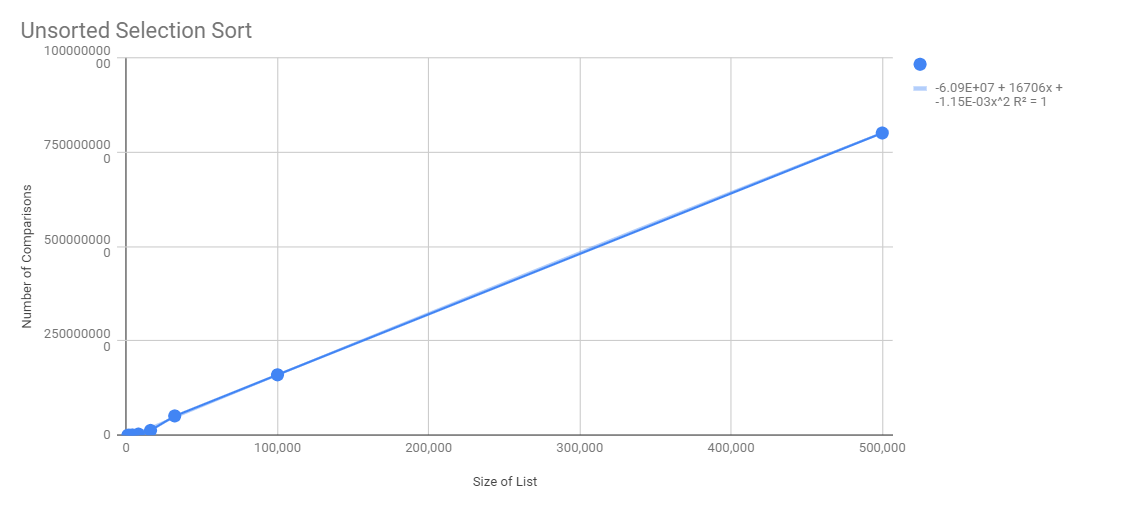
Hayden Tam

Unsorted List Insertion Sort



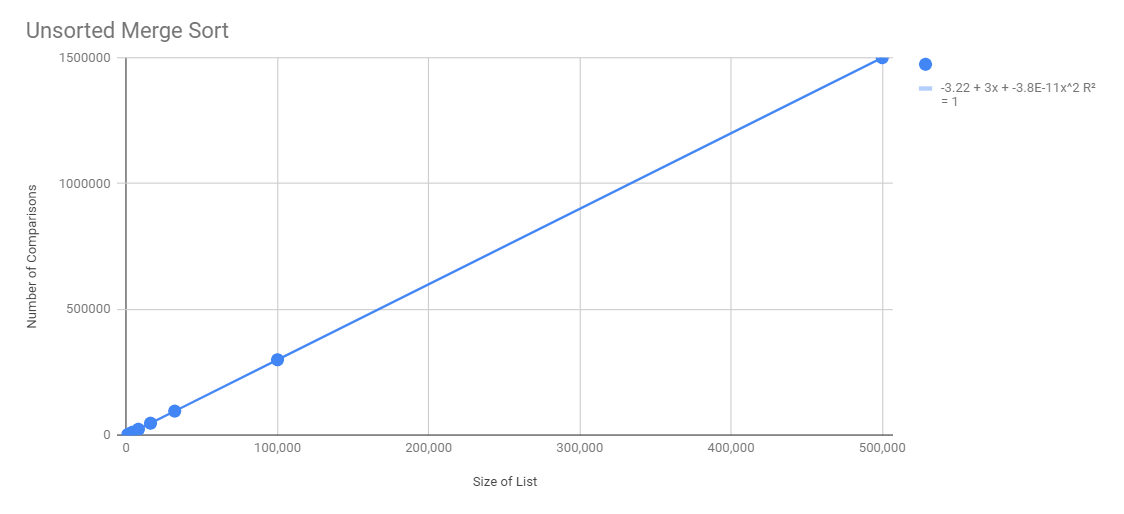
Unsorted List Selection Sort

|  |  |  |
| --- | --- | --- |
| List Size | Comparisons | Time(seconds) |
| 1,000 | 508084 | 0.05684828758239746 |
| 2,000 | 2018821 | 0.23388028144836426 |
| 4,000 | 8038281 | 0.9129457473754883 |
| 8,000 | 32078467 | 3.6937599182128906 |
| 16,000 | 128161279 | 14.743354320526123 |
| 32,000 | 512342923 | 59.335564374923706 |
| 100,000 (estimated) | 1601071634 | 185.4236375 |
| 500,000 (estimated) | 8005358172 | 927.1181875 |



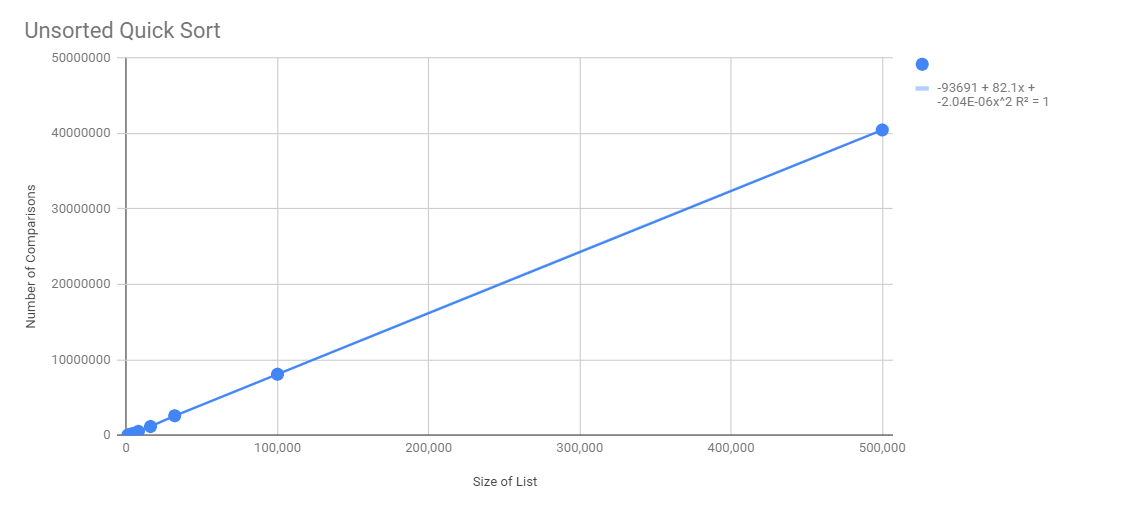
Unsorted List Merge Sort

|  |  |  |
| --- | --- | --- |
| List Size | Comparisons | Time(seconds) |
| 1,000 | 2999 | 0.003989219665527344 |
| 2,000 | 5999 | 0.008975982666015625 |
| 4,000 | 11997 | 0.01795220375061035 |
| 8,000 | 23995 | 0.03889584541320801 |
| 16,000 | 47991 | 0.08477354049682617 |
| 32,000 | 95999 | 0.18102097511291504 |
| 100,000 (estimated) | 299996 | .5656905472 |
| 500,000 (estimated) | 1499984 | 2.828452736 |



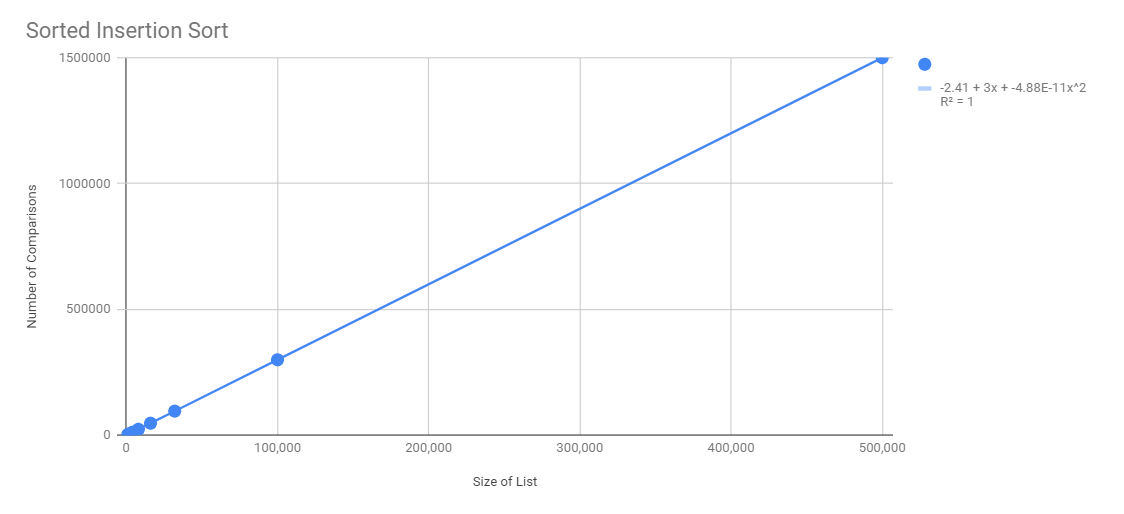
Unsorted List Quick Sort

|  |  |  |
| --- | --- | --- |
| List Size | Comparisons | Time(seconds) |
| 1,000 | 28587 | 0.002991914749145508 |
| 2,000 | 88609 | 0.005983591079711914 |
| 4,000 | 224764 | 0.013962507247924805 |
| 8,000 | 522239 | .03142094612121582 |
| 16,000 | 1159817 | 0.06590390205383301 |
| 32,000 | 2587472 | 0.151108980178833 |
| 100,000 (estimated) | 8085850 | .4722155631 |
| 500,000 (estimated) | 40429250 | 2.361077815 |



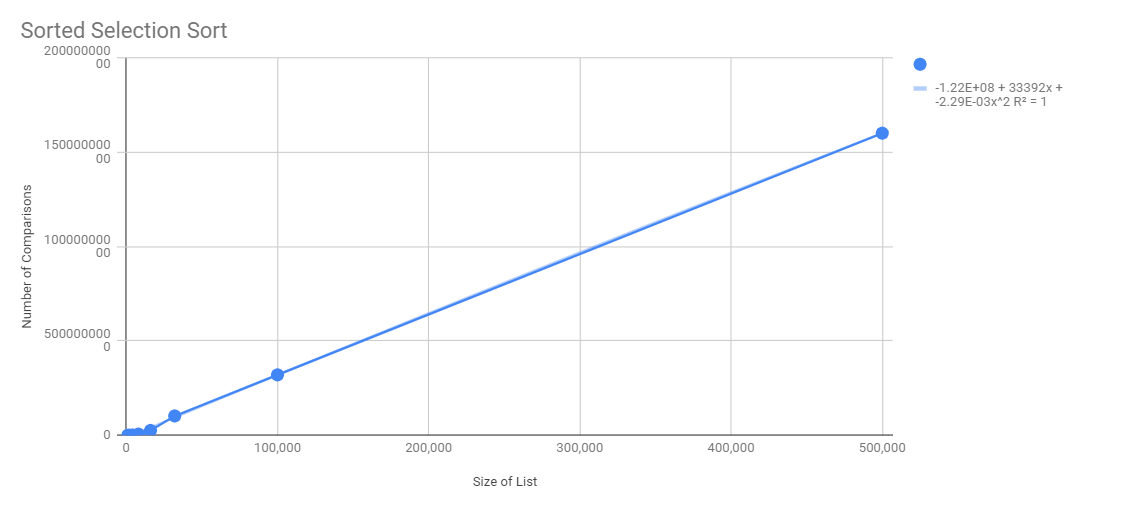
Sorted List Insertion Sort

|  |  |  |
| --- | --- | --- |
| List Size | Comparisons | Time(seconds) |
| 1,000 | 2997 | 0.0006251335144042969 |
| 2,000 | 5997 | 0.0011277198791503906 |
| 4,000 | 11997 | 0.0023038387298583984 |
| 8,000 | 23997 | 0.004436969757080078 |
| 16,000 | 47997 | 0.009141683578491211 |
| 32,000 | 95997 | 0.01764988899230957 |
| 100,000 (estimated) | 299990 | .0551559031 |
| 500,000 (estimated) | 1499953 | .2757795155 |



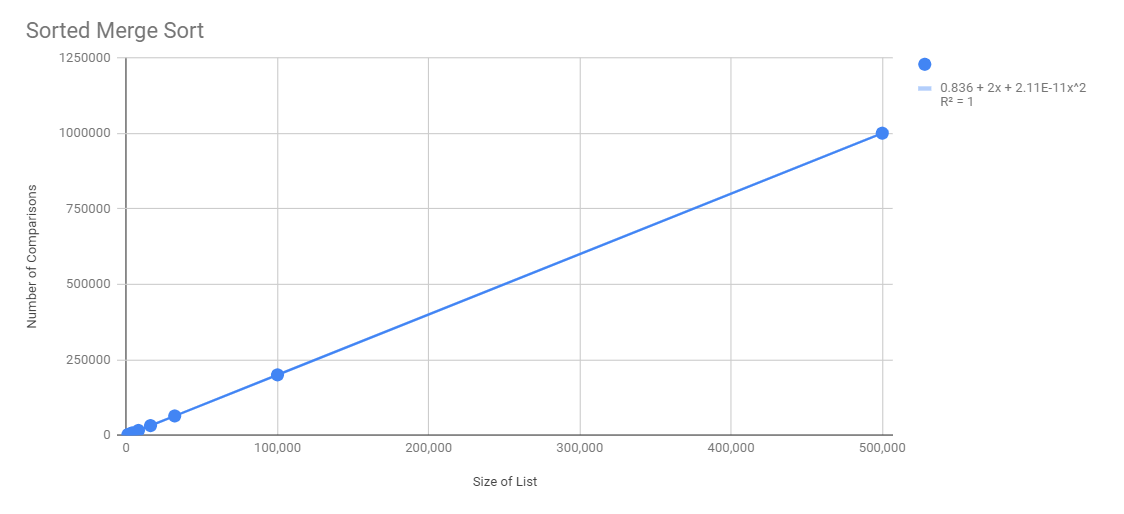
Sorted List selection Sort

|  |  |  |
| --- | --- | --- |
| List Size | Comparisons | Time(seconds) |
| 1,000 | 1001997 | 0.1791536808013916 |
| 2,000 | 4003997 | 0.7623109817504883 |
| 4,000 | 16007997 | 3.071711301803589 |
| 8,000 | 64015997 | 12.313228130340576 |
| 16,000 | 256031997 | 49.94814109802246 |
| 32,000 | 1024063997 | 198.22587251663208 |
| 100,000 (estimated) | 3200199991 | 619.4558516 |
| 500,000 (estimated) | 16000999950 | 3097.279258 |



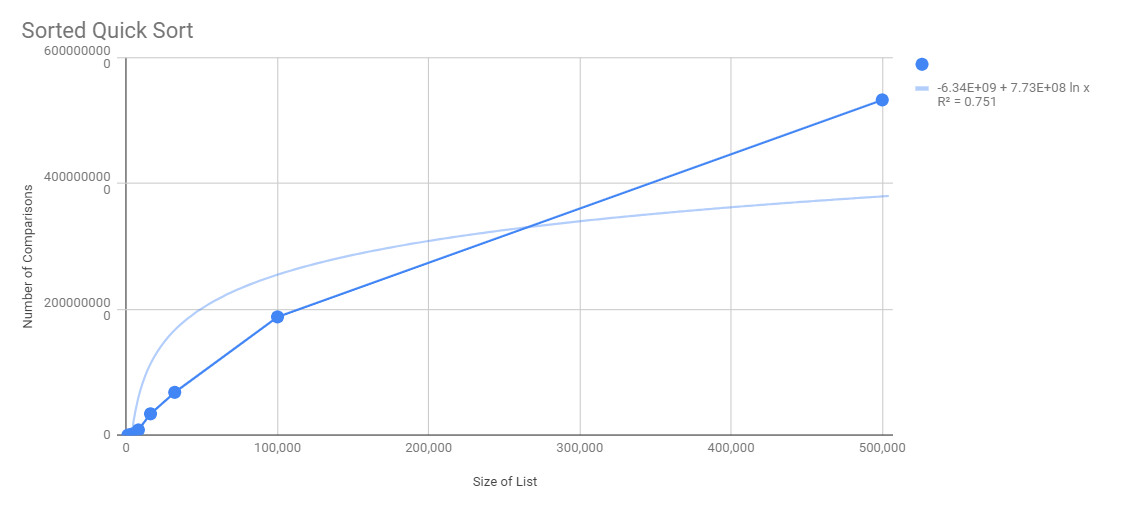
Sorted Merge Sort5

|  |  |  |
| --- | --- | --- |
| List Size | Comparisons | Time(seconds) |
| 1,000 | 2501 | 0.007691621780395508 |
| 2,000 | 5001 | 0.01641535758972168 |
| 4,000 | 10001 | 0.03570699691772461 |
| 8,000 | 20001 | 0.07871079444885254 |
| 16,000 | 40001 | 0.16773438453674316 |
| 32,000 | 80001 | 0.35271167755126953 |
| 100,000 (estimated) | 250003 | 1.102223992 |
| 500,000 (estimated) | 1250015 | 5.511119962 |



Sorted Quick Sort

|  |  |  |
| --- | --- | --- |
| List Size | Comparisons | Time(seconds) |
| 1,000 | 1001997 | 0.1808180809020996 |
| 2,000 | 5005994 | 0.7059662342071533 |
| 4,000 | 21013991 | 2.805424690246582 |
| 8,000 | 85029988 | 11.09237027168274 |
| 16,000 | 341061985 | 44.1229784488678 |
| 32,000 | 682123970 | 88.2459569 |
| 100,000 (estimated) | 1881083827 | 1.164519787 |
| 500,000 (estimated) | 5329093516 | 5.822598934 |



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

Depends on the type of list. If the list is unsorted, which most lists are, the best sort would be quick sort, as it runs as an O(nlogn) function while taking very little space in memory. With the sample size we had, we weren’t able to clearly see a O(nlogn) function produced; however, we do that the number of comparisons and the time taken is much less compared to selection and insertion sort. Compared to merge sort, it takes less memory by working with the same list, as it doesn’t stop your code because memory is filled up, which happened to me. For an unsorted list, quick sort goes divides the list into different parts, which decreases the number of comparisons needed to go through the entire list.

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

When a list is already sorted, insertion sort is the best because it has the least run time and few comparisons. However, merge sort does the least comparisons, but the run time is second compared to insertion sort. Because the list is already sorted, insertion sort does well because as it goes through the list assuming that the comparing it to the previous values, it sees that everything is already sorted which is a lot faster compared to merge or quick sort which needs a list to be partitioned and traverse through the sublists. It might take more comparisons, but it takes less time.

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

Insertion sort does half as many comparisons as selection sort. This is because for a huge list, selection sort is very inefficient as it finds the minimum value by going through the entire list to find the right position. This leads to more comparisons. However, insertion sort goes through the list the same way however the first part is already a sorted list, so all you do is pick and place into the small divided sublist. The times don’t change drastically however, since you are still going through the list and comparing numerous values to the current.