Koby Miller

Data Structures and Algorithms

Project 2: Comparing sorting algorithms

The two algorithms I decided to compare are Merge sort and Selection sort. I wanted to test one of the most basic sorting algorithms to one of the quicker ones we learned about. Selection sort is the kind of code that you write when you first begin coding. It is simple, and it gets the job done. However, it's time complexity is pretty bad. These are things you don't think about when you first start programming though. So, comparing it to Merge sort shows that if you take a little more time to think about your code and you put a little more effort into it, the result can come out better. So even though Merge sort is not as "easy" as Selection sort, the time complexity is better.

Furthermore, if we don't even look at the sorts yet but just how the size of the files effected the sorting times, it is not surprising to see that the bigger the file, the more time both algorithms took to fully run. This is because their time complexities depend on the number of elements the algorithm needs. The more elements there are, the more computations there are, meaning it needs extra time to run those extra computations.

Moreover, Merge sort's time complexity for best/worst/average case is O(n*log n). This can be seen in the graph below. In each case, every time the size of the text file increases, the time increases. It is hard to tell because we only have 4 file sizes, but the lines are slowly curving upwards (you can compare it to the graph of n*log n at the bottom). There are a couple other interesting things to point out about this graph. First, the Times of the Normal case and Backwards cases are almost identical. In a perfect world, they should be identical. They have the same amount of computations, but the reason they are different is because the computer is not perfect. So, each time it runs the algorithm, the times will vary slightly. Still, they are very close, keep in mind that the times are $1/1000^{th}$ of a second! The second thing I would like to point out is the fact that the random case takes just a little more time for each text file size. The reason for this is because it has more computations than the other 2 cases. I'll try to make a brief explanation why. The Merge algorithm splits everything up then puts the piece back in order by comparing 2 sets of numbers at a time. Well when comparing two sets, if one set ends up being used (like say all the numbers in one set are lower than the second set) than the other set is just added to the end of the merged set – there are no more comparisons. So, the algorithm only compares as long as two sets have numbers. In both the normal and backwards case, when merging sets, every time one set will get used then the other put behind it. So, say that a set has N numbers, it only has to compare to another set (of size N) N times, whereas in the random case one set is not always smaller than the other set giving it from N to 2N-1(worst case) comparisons. In addition, we know that with time complexity you do not worry about the coefficients or constants. So, if we had a time complexity of O(2n-1), that is still equal to O(n). So the algorithm first keeps dividing the list in half until one element is reached which takes O(log n) time. This is why the random case is still O(n*log n). However, the line is still shifted

due to those extra computations. The last thing I would like to cover is Merge sort's worst case. After the algorithm separates each number into its own set, the order of the worst case would make it so that each time 2 sets get merged, each number in each set is compared at least one time.

To continue, let's look at Selection sort. It's time complexity for best/worst/average case is O(n^2). This can be seen with all cases on the graph (below) as they all curve upwards with the curve getting steeper and steeper the bigger the size of the file. The algorithm works by starting at the beginning of the list and comparing that number to every other number in the list after it. Once it finds a smaller number, that number is saved temporarily as the smallest number. Now every number after the smallest number is compared to the smallest number until you get to the end of the list and you know the index of the smallest number. You now switch that number with the number at the beginning of the list. You then go to the second index of the list and do the same thing. You continue this until you have been through the whole list and it is sorted. Say we have a set of size N. We will have to run through the list N times (one time for each index) then at each index you will compare a number to the rest of the list. So, for the first run, you

compare N times. The next run through you will compare N – 1 times, then N – 2 times…etc. all the way until 1. So, you end up with:

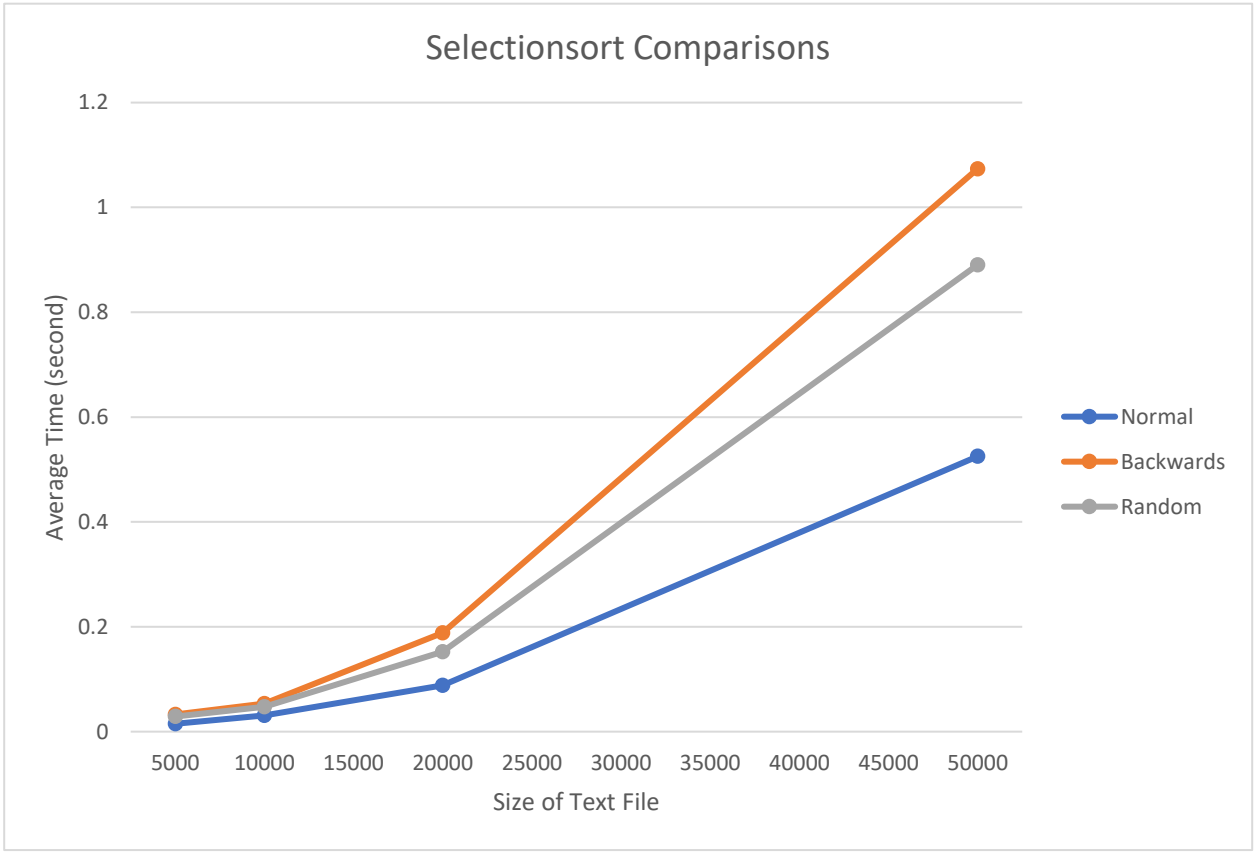
N* [(N) + (N-1) + (N-2)….+1]

Once multiplying through, the ending polynomial will end up being:

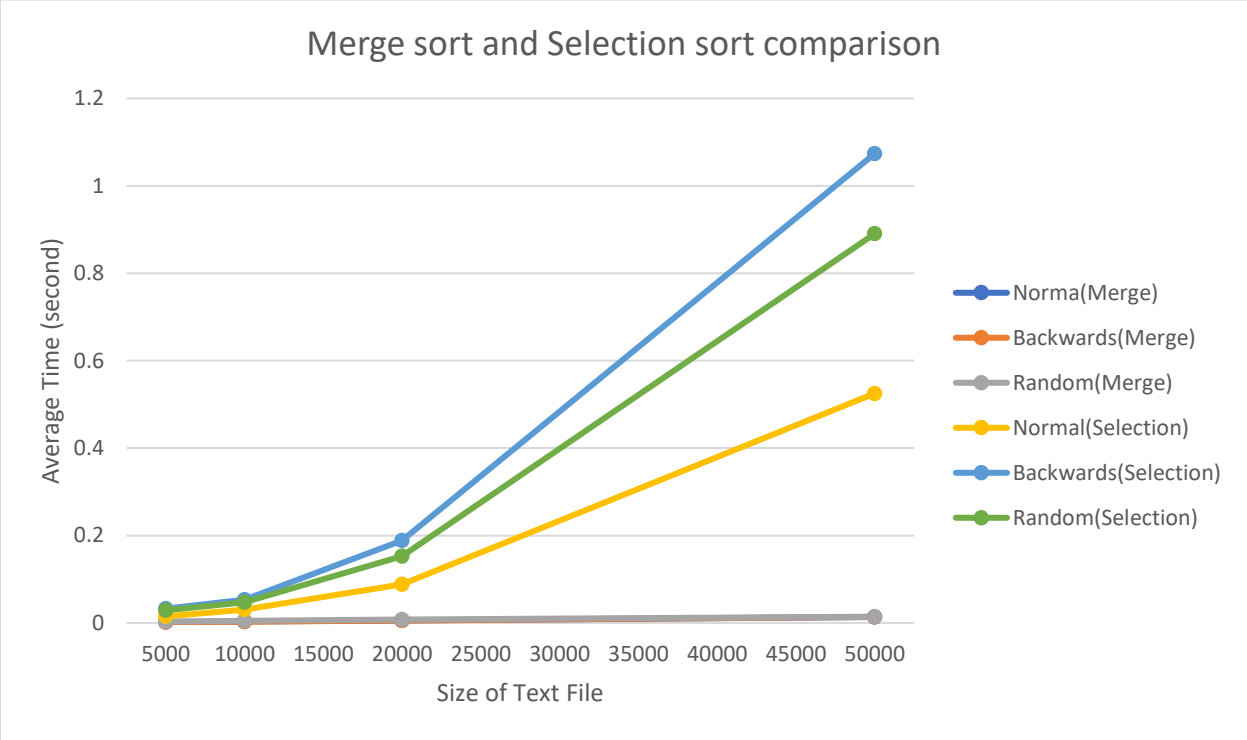
aN^2 + bN

where a and b are constants. Therefore, the time complexity is O(n^2).

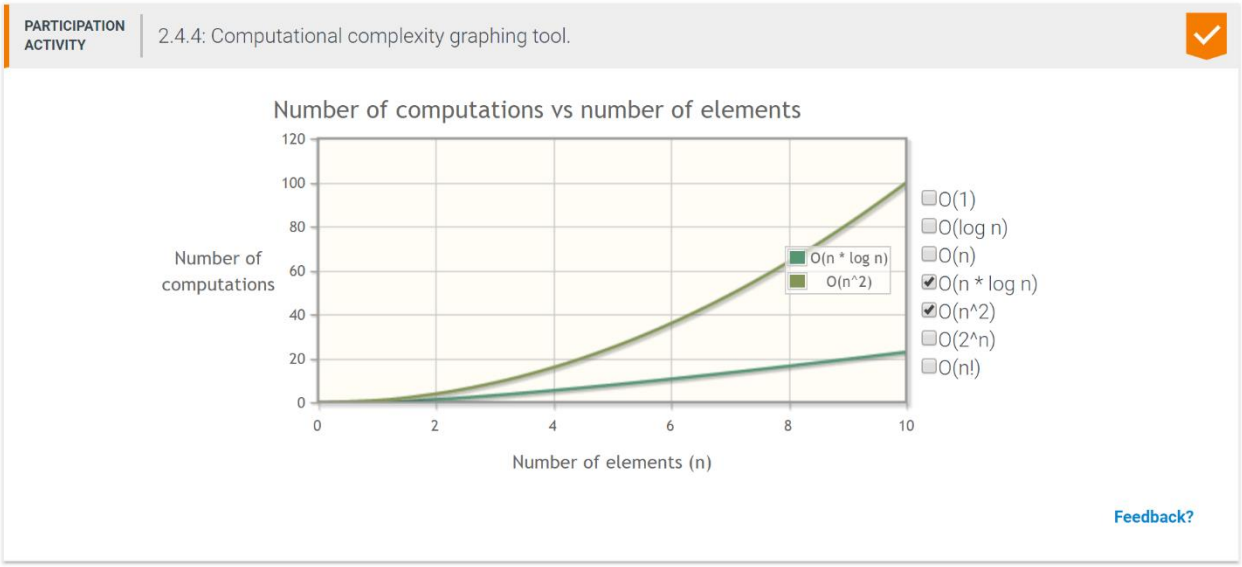
With the help of the graph you can see that the already sorted list is the best case, and the backwards is the worst case. The simple reason for this is that if the smallest number is already in front, no numbers ever need to switch. Both cases have to make the same amount of comparisons, but the best case doesn't have any of the switching number computations. This is why the random case is in the middle of the two. Sometimes the smallest number is already in the right place. The spread between each case grows bigger and bigger as the files get bigger. That shows how much effect the order of the original set has.



Finally, when comparing both sorting algorithms, as expected, you can see that Merge sort (most of the time) takes far less time than Selection sort. All three of Merge sorts cases were so similar and short that they all look like they form one small line on the x-axis. At the start, Selection sort is not too far behind Merge sort, but begins to break off at around 20000. From then on, the gaps between the two grows. This shows that selection sort is good for small list because its lack of efficiency. However, one advantage it has over merge sort is the amount of memory it takes up. Selection sort only needs to take up the memory of the original set, whereas Merge sort's memory usage varies because of the new set it makes.



Below is the comparison of the O(n^2) and O(n*log n) graphs from Zybooks. If I had more trials you could see my Merge sort rise a little more and both graphs would look identical.



Some final notes. Because the algorithm takes a different amount of time to run as said before, I took the average of 5 times. This is why the graphs are not perfect and might have a small hump or bend the wrong way. I also measured in nanoseconds to be as precise as possible. I referred to the clock right before I called the algorithm and then once it finished. I then subtracted the numbers to get the amount of nanoseconds it took to run the algorithm. I then convert the time to seconds for my graphs. When finding the average, I tried not to take any outliers of time. For example, say I was running a certain case and kept getting times around 0.2 seconds, but then out of nowhere 0.5 seconds (which would be a big leap for these times). I would re-run it and not take that outlier. Below are pictures from the excel sheet I used to keep track of times and get averages. The last thing is that the text files I made and used were all from 1 to whatever their name is. For backwards it is that same list backwards and then the random files are just the first file mixed up. This doesn't make any difference but is good to know. Because of this, there were no repeat numbers.

THESE FIRST CHARTS ARE MERGESORT
NOTES: TIME IS IN NANO SECONDS

5000.txt

Test #	Time	Average time:
1	2174540	2289411
2	2112545	
3	2471382	In Seconds:
4	2624909	0.002289411
5	2063679	

5000backwards.txt

Test#	Time	Average Time
1	1849981	2052592.8
2	1885354	
3	2219394	In Seconds:
4	2442573	0.002052593
5	1865662	

5000RANDOM.txt

Test#	Time	Average time:
1	2381308	3105255.4
2	3665683	
3	2842983	In Seconds:
4	3139827	0.003105255
5	3496476	

10000.txt

Test #	Time	Average time:
1	3793683	3460883.4
2	3441410	
3	3761592	In Seconds:
4	2937433	0.003460883
5	3370299	

10000backwards.txt

Test#	Time	Average time:
1	3367382	2785365
2	2886744	
3	2389331	In Seconds:
4	2368545	0.002785365
5	2914823	

10000RANDOM.txt

Test#	Time	Average time:
1	4384452	4712802.4
2	4231654	
3	5882524	In Seconds:
4	5144793	0.004712802
5	3920589	

20000.txt

Test #	Time	Average time:
1	5254194	5514205.6
2	5248360	
3	5894923	In Seconds:
4	5812143	0.005514206
5	5361408	

20000backwards.txt

Test#	Time	Average time:
1	5595892	5468694.6
2	5044873	
3	5845692	In Seconds:
4	5291391	0.005468695
5	**5565625**	

20000RANDOM.txt

Test#	Time	Average time:	(random values from 5 to 12)
1	8066180	7460824.8	
2	7545794		
3	7294899	In Seconds:	
4	7153771	0.007460825	
5	7243480		

50000.txt

Test #	Time	Average time:
1	11125414	13446260.2
2	12662872	
3	12969561	In Seconds:
4	16662230	0.01344626
5	13811224	

50000backwards.txt

Test#	Time	Average time:
1	13871395	13880949.2
2	15989410	
3	14958117	In Seconds:
4	12490382	0.013880949
5	12095442	

50000RANDOM.txt

Test#	Time	Average time:
1	13647121	14253425
2	13437070	
3	14284203	In Seconds:
4	14962494	0.014253425
5	14936237	

THE BOTTOM IS SELECTION SORT

5000.txt

Test #	Time	Average time:
1	14641582	15015079.2
2	12961174	
3	14380112	In Seconds:
4	16616281	0.015015079
5	16476247	

5000backwards.txt

Test#	Time	Average Time
1	36175830	32831350
2	30547482	
3	29171939	In Seconds:
4	28488544	0.03283135
5	39772955	

5000RANDOM.txt

Test#	Time	Average time:
1	26789902	28859342.6
2	27228602	
3	29536247	In Seconds:
4	33395206	0.028859343
5	27346756	

10000.txt

Test #	Time	Average time:
1	33563684	30678180.2
2	36660479	
3	27800772	In Seconds:
4	26499622	0.03067818
5	28866344	

10000backwards.txt

Test#	Time	Average time:
1	49324809	53474048.2
2	64469639	
3	57690386	In Seconds:
4	44254046	0.053474048
5	51631361	

10000RANDOM.txt

Test#	Time	Average time:
1	39722630	47288262.2
2	50024251	
3	45164996	In Seconds:
4	43874057	0.047288262
5	57655377	

20000.txt

Test #	Time	Average time:
1	84676846	88591964.6
2	88536170	
3	89170699	In Seconds:
4	91586651	0.088591965
5	88989457	

20000backwards.txt

Test#	Time	Average time:
1	185696104	188712306.4
2	183690043	
3	188307156	In Seconds:
4	190294983	0.188712306
5	195573246	

20000RANDOM.txt

Test#	Time	Average time:
1	147097028	152530641.6
2	151414381	
3	159998760	In Seconds:
4	150561048	0.152530642
5	153581991	

50000.txt

Test #	Time	Average time:
1	521898556	525128383
2	526638198	
3	520431480	In Seconds:
4	533147594	0.525128383
5	523526087	

50000backwards.txt

Test#	Time	Average time:
1	1.119E+09	1073813776
2	1.015E+09	
3	1.034E+09	In Seconds:
4	1.145E+09	1.073813776
5	1.057E+09	

50000RANDOM.txt

Test#	Time	Average time:
1	922300789	890555504.8
2	896882054	
3	872634281	In Seconds:
4	884501803	0.890555505
5	876458597	