Comp Sci 201

DNA Analysis

To generate the data for each respective hypothesis, I created a program/class called SyntheticDNA. The program generates multiple strands of DNA made up of "c"s and "gaat"s, with "gaat" being the enzyme. The strands were all created with the same length but with varying numbers of occurrences of the enzyme "gaat". The length of the strands depended on whether I was testing the hypothesis for StringStrand, StringBuilderStrand, or LinkStrand, but all of the strands for each respective hypothesis had the same length.

**note: in the benchmark program, I changed the ENZYME instance variable from "gaattc" to "gaat" so that it was compatible with the strands I created with my program.

```
public static String makeStrand(int length){

		StringBuilder inbetween = new StringBuilder();
		StringBuilder strand = new StringBuilder();
		StringBuilder enzyme = new StringBuilder("gaat");

		for (int i=0; i<length; i++){
			inbetween.append("c");
		}

		for (int i=0; i<4600000/(length+4); i++){
			strand.append(inbetween);
			strand.append(enzyme);
			}

		System.out.println(strand.length());
		return strand.toString();

	}
```

Note: when testing the hypothesis for the StringStrand class, the second for loop used 220000 rather than 4600000 because 4600000 took too much time for StringStrand.

This method is called five or six times in the SyntheticDNA class: once for each DNA strand generated. The parameter "length" of this method represents the amount of c's between each occurrence of the enzyme gat in the given strand. Once each strand is generated, it is written to a file whose title specifies how many c's are between each enzyme in that strand.

The StringBuilder "inbetween" represents the concatenation of c's in between each occurrence of gaat, the StringBuilder "strand" represents the DNA strand being generated, and the StringBuilder "enzyme" represents the enzyme gaat.

I used the first for loop to generate "inbetween" based off of the parameter "length." So, if 8 were passed into this method, the DNA strand generated would have 8 c's in between each occurrence of gaat. The second for loop creates the entire DNA strand by appending "inbetween" and then appending gat (the enzyme) to the DNA strand multiple times.

In order to ensure that each DNA strand was the same length (4600000 in this case), the "length" parameters that I used were all factors of 4600000 (or 220000 in the case of StringStrand) subtracted by 4, since 4 is the length of gaat. Because the "length" parameter is always a factor of 4600000 minus 4, joining "inbetween" and "gaat" creates a segment that factors into 4600000 without a remainder.

The rest of the code in this SyntheticDNA class consists of writing the generated DNA strands to various files saved in the "data" folder of the DNA project in eclipse, which I later used when running the

benchmark program. The LinkStrand and StringBuilder were both tested using the files labeled "SBlength.txt" and the StringStrand was tested using the files labeled "SSlengthbetween.txt," with "length" representing the parameter used to call the method shown above. I ran SyntheticDNA on all of the files before turning them into gitHub, so all of the files there contain the DNA strands generated by the class.

I submitted the class SyntheticDNA with the code used to write strands to the "SB" files, which are used for testing StringBuilderStrand and LinkStrand. The green comments throughout the class indicate what the code should say to write to the "SS" files correctly. This is explained in the comments section of the class as well.

All of my tests were run with the -Xmx8192M heap size.

Non Linked List Hypothesis (NLLH):

Part 1: The runtime of the method cutAndSplice for using a **StringStrand** is $O(b^2S)$, ignoring the O(N) time to search through the strand.
Part 2: The runtime of the method cutAndSplice for using a **StringBuilderStrand** is O(bS), ignoring the O(N) time to search through the strand.

Part 1 of the NLLH: ***Is the runtime $O(b^2S)$ for using a StringStrand?***

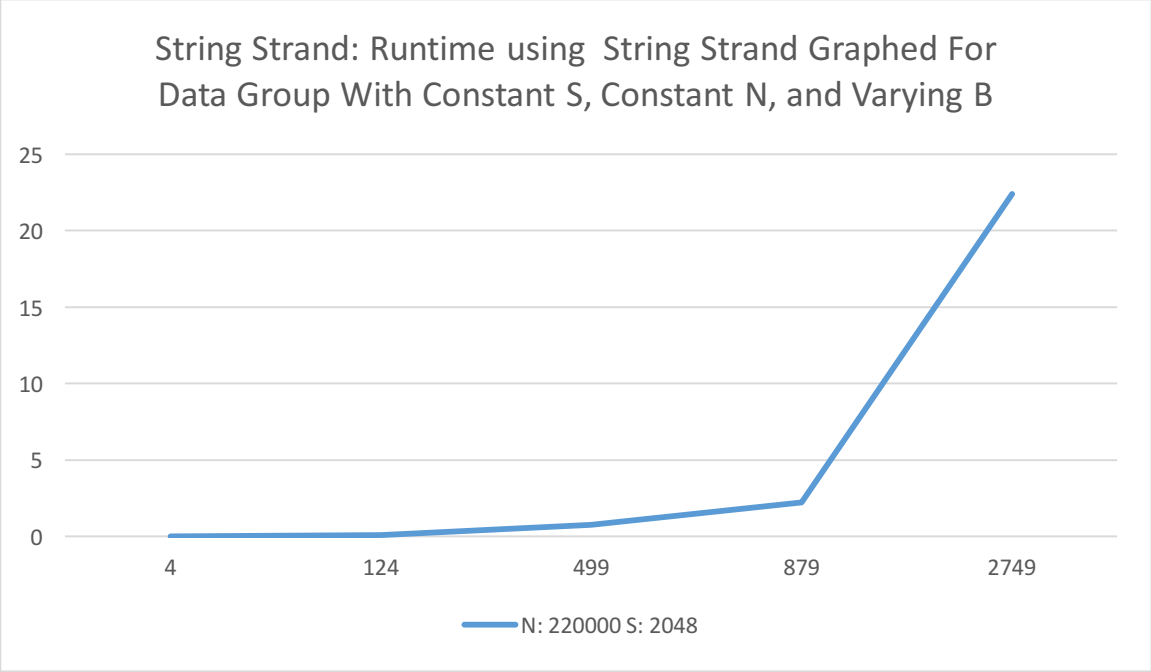

Figure 1
S = length of splicee (measured in # of characters)
N = length of original DNA strand (measured # of characters)
B = number of occurrences of the enzyme in the DNA strand
X axis: B
Y axis: Time (measured in seconds)

The data plotted above illustrates the runtime of cutAndSplice using StringStrand on multiple DNA strands (created by the code described at the top of the document) that all have length 220000 but have varying numbers of occurrences of the enzyme being spliced (b). The data represents only the runtimes

from the splicee of size 2048. Controlling the variable N (length of the DNA strand) and the variable S (length of the splicee) allowed me to generate a graph that shows the relationship between runtime and the variable b (the number of occurrences of the enzyme in the DNA strand).

From the graph above, it's clear that the relationship between runtime and b is quadratic due to the shape of the line. As b increases, the runtime increases exponentially. To prove that the runtime is proportional to S as well, the graph below illustrates the runtime of cutAndSplice using StringStrand on one DNA strand of length 220000 with 2749 occurrences of the enzyme, with varying values of S.

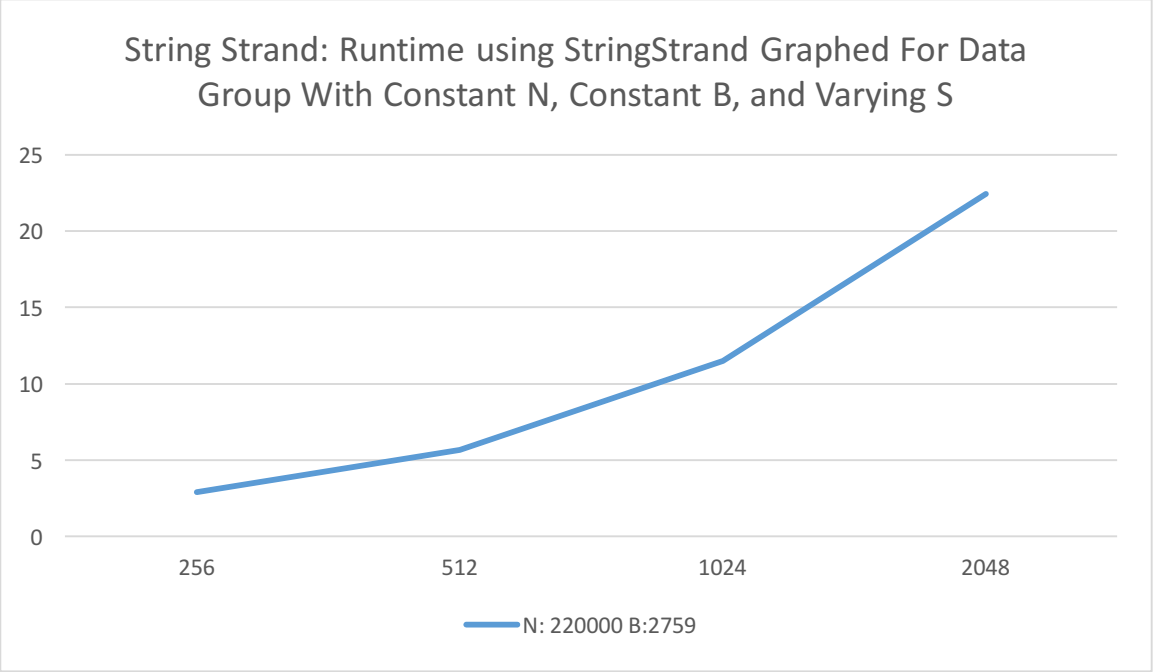

Figure 2
S = length of splicee (measured in # of characters)
N = length of original DNA strand (measured # of characters)
B = number of occurrences of the enzyme in the DNA strand
X axis: S
Y axis: Time (measured in seconds)

The graph above demonstrates a relatively linear relationship between runtime and S, supporting the hypothesis that S effects the runtime linearly since the runtime is $O(b^2S)$.

Conclusion for part 1: The data illustrated in Figure 1 and Figure 2 support part one of the NLL hypothesis that the runtime of the method cutAndSplice for using a StringStrand is $O(b^2S)$, by showing that the runtime has a quadratic relationship with b and a linear relationship with S.

Part 2 of the NLLH: ***Is the runtime O(bS) for using a StringBuilderStrand?***

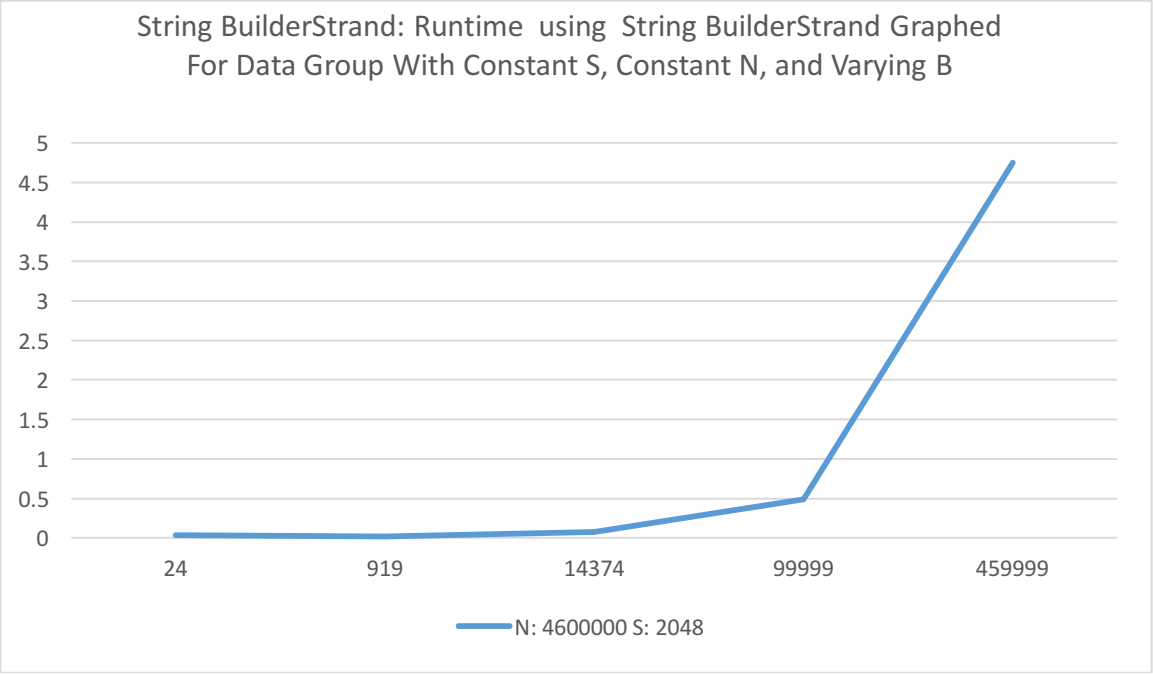

Figure 3
S = length of splicee (measured in # of characters)
N = length of original DNA strand (measured # of characters)
B = number of occurrences of the enzyme in the DNA strand
X axis: B
Y axis: Time (measured in seconds)

The data plotted above illustrates the runtime of cutAndSplice using StringBuilderStrand on multiple DNA strands (created by the code described at the top of the document) that all have length 4600000 but have varying numbers of occurrences of the enzyme being spliced. The data represents only the runtimes from the splicee of size 2048. Controlling the variable N (length of the DNA strand) and the variable S (length of the splicee) allowed me to generate a graph that shows the relationship between runtime and the variable b (the number of occurrences of the enzyme in the DNA strand).

From the graph above, it's clear that there is a relationship between runtime and b, however the graph does not reflect a linear relationship, therefore refuting part 2 of the NLL hypothesis. Rather, the graph implies more of a quadratic relationship between b and runtime. However, the small values of the Y axis indicate that StringBuilderStrand was certainly much more efficient than StringStrand, as the same b and S values were used to generate their data and the N value was much bigger for StringBuilderStrand (220000 vs. 4600000), further emphasizing its efficiency. This is important because it demonstrates that a StringBuilderStrand is much more efficient than a StringStrand because the use of a StringBuilder allows you to append a string to another rather than having to make an entirely new string. To check if the runtime is related to S as well (which the hypothesis says it should be), the graph below illustrates the runtime of cutAndSplice using StringStrandBuilder on one DNA strand of length 4600000 with 99999 occurrences of the enzyme, with varying values of S.

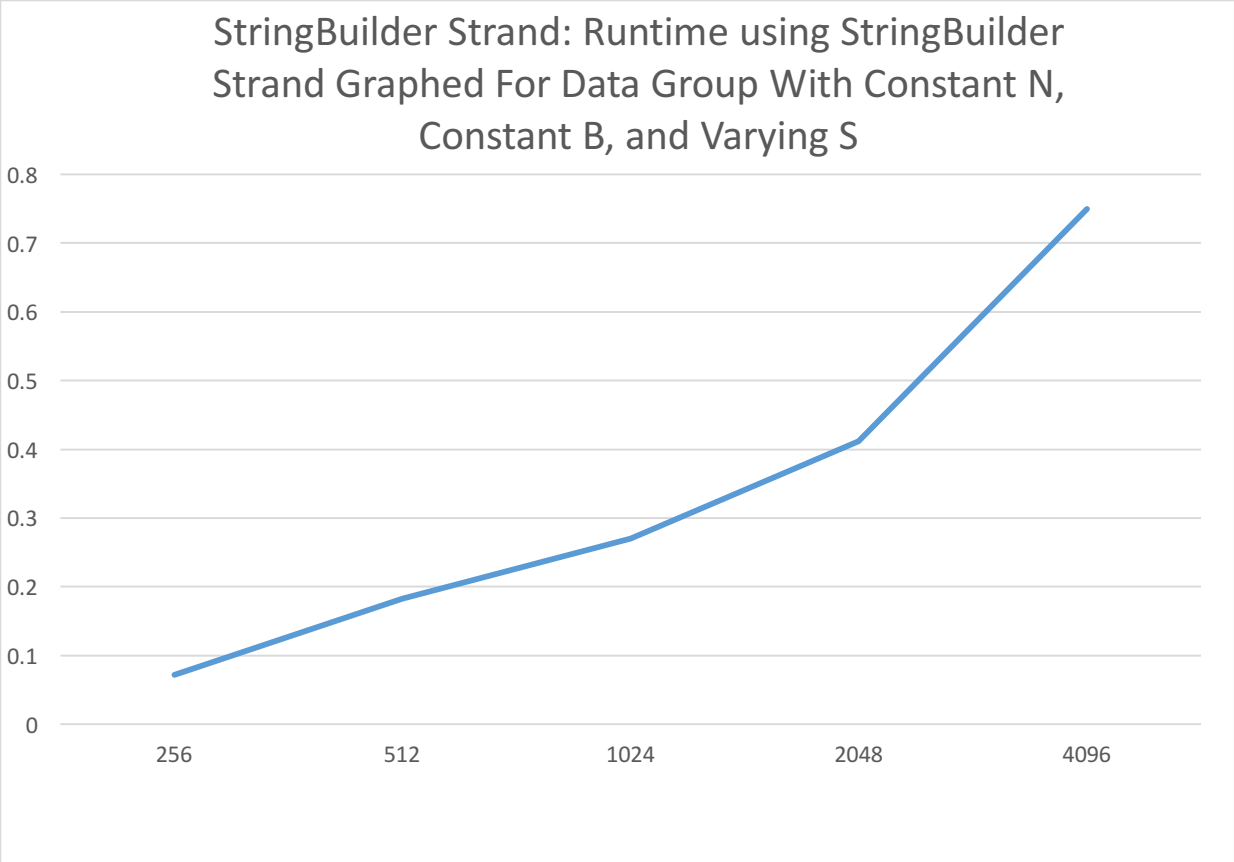

Figure 4
S = length of splicee (measured in # of characters)
N = length of original DNA strand (measured # of characters)
B = number of occurrences of the enzyme in the DNA strand
X axis: S
Y axis: Time (measured in seconds)

The graph above demonstrates a linear relationship between runtime and S, supporting the part of the hypothesis that states there should be a linear proportionality between runtime and S.

Conclusion for part 2: The data illustrated in Figure 3 and Figure 4 partially refutes the hypothesis that the runtime of the method cutAndSplice for using a StringStrand is O(bS). While Figure 3 demonstrates that there is indeed a relationship between runtime and b, the relationship (according to my data) is quadratic rather than linear, therefore refuting that part of the hypothesis. Figure 4 demonstrates that there is a linear relationship between runtime and S, therefore supporting that part of the hypothesis.

LinkStrand Hypothesis (LSH):

Part 1: LinkStrand runs in O(b) time when N is fixed, regardless of the size of the splicee.

The LSH: ***Is the runtime O(b) for using a LinkStrand?***

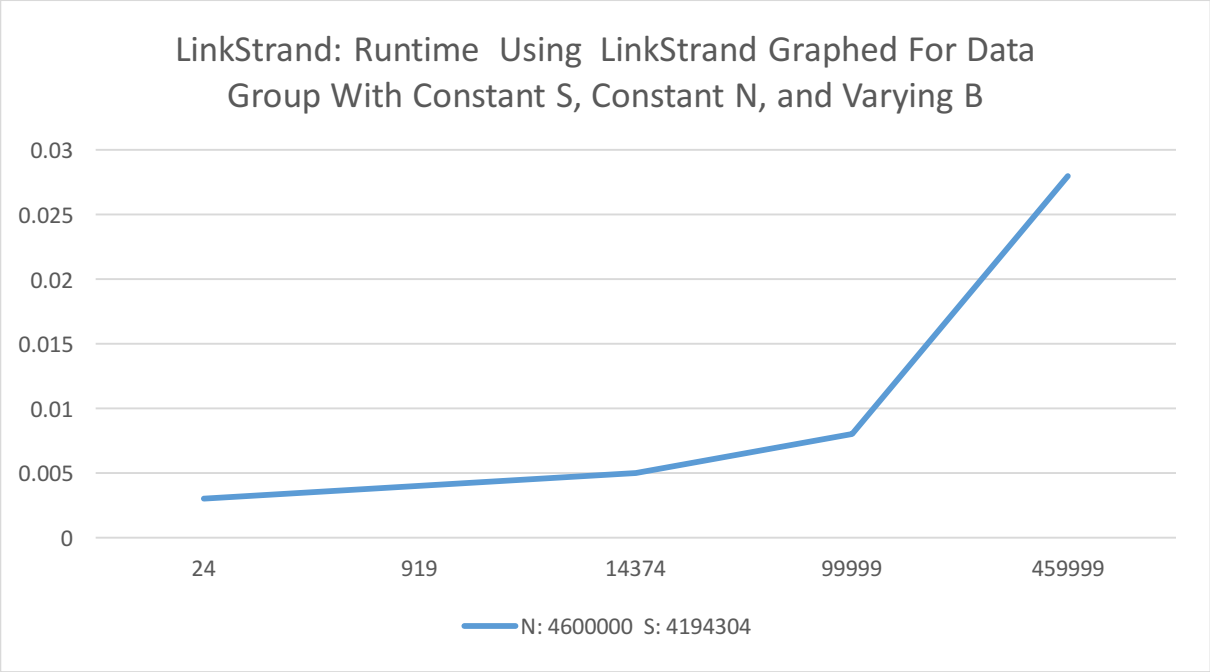

Figure 5
S = length of splicee (measured in # of characters)
N = length of original DNA strand (measured # of characters)
B = number of occurrences of the enzyme in the DNA strand
X axis: B
Y axis: Time (measured in seconds)

The data plotted above illustrates the runtime of cutAndSplice using LinkStrand on multiple DNA strands (created by the code described at the top of the document) that all have length 4600000 but have varying numbers of occurrences of the enzyme being spliced. The data represents only the runtimes from the splicee of size 4194304. Controlling the variable N (length of the DNA strand) and the variable S (length of the splicee) allowed me to generate a graph that shows the relationship between runtime and the variable b (the number of occurrences of the enzyme in the DNA strand).

From the graph above, it's clear that there is a relationship between runtime and b, as predicted by the hypothesis. However, according to my data, that relationship is quadratic rather than linear, which refutes the hypothesis. This may be because the times (Y value) are so small and so close together that they may not be sufficient to truly show the relationship between runtime and b.

It's important to note that the smallness of the times indicates that LinkStrand is much more efficient than StringBuilderStrand, as the same N and B values were used (although the S value was much bigger for LinkStrand, further emphasizing its efficiency). This is important because it demonstrates the extreme increase in efficiency that results from using a LinkedList rather than a StringBuilder, as a LinkedList saves all the time that it takes to append a string to the end of another. To check if the runtime is unaffected by S, as the hypothesis says it should be, the graph below illustrates the runtime of cutAndSplice using the same DNA strand length as Figure 5 (4600000) and the same varying B values,

but this time with varying S values too (starting at S=4194304, the S value is doubled for each increased B value shown on the X axis). If runtime is unaffected by S, the graph below should look nearly identical to Figure 5.

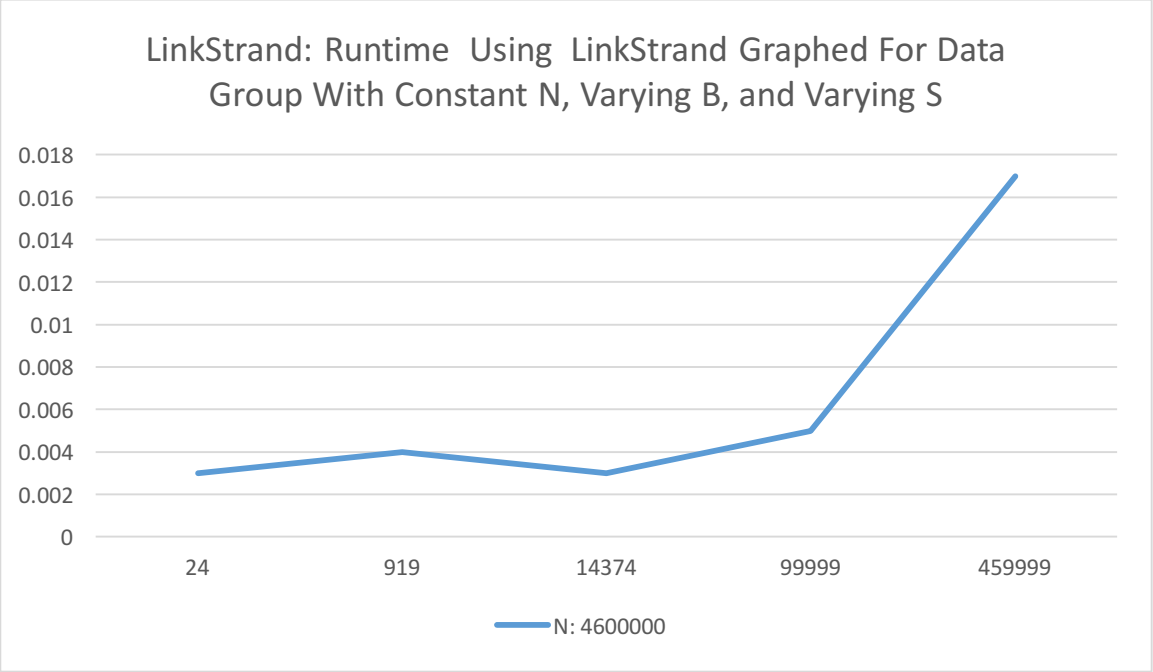

Figure 6
S = length of splicee (measured in # of characters)
N = length of original DNA strand (measured # of characters)
B = number of occurrences of the enzyme in the DNA strand
X axis: B
Y axis: Time (measured in seconds)

Because of the huge difference in size between the S values and the runtime values, I wasn't able to include a separate line to show the growing values of S because it threw off the scale of the graph, making the blue line appear horizontal. Despite that, the data from the graph seems to support the hypothesis's claim that S does not affect the runtime, as this graph is quite similar to Figure 6 except for the slight upwards curve at the x value of 919. That slight curve may be an irregularity/flaw in the data. To look at the relationship between runtime and S more closely, the graph below illustrates the runtime of cutAndSplice using LinkStrand with an N value of 4600000, a B value of 919, and a varying S value.

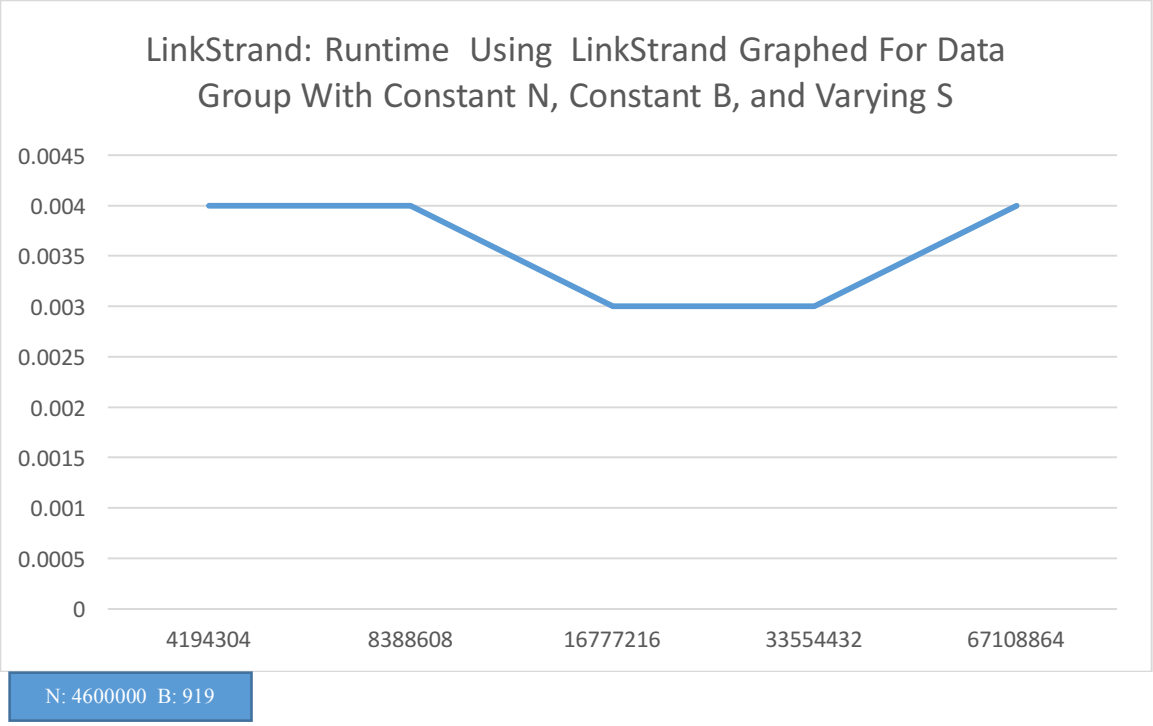

Figure 7
S = length of splicee (measured in # of characters)
N = length of original DNA strand (measured # of characters)
B = number of occurrences of the enzyme in the DNA strand
X axis: S
Y axis: Time (measured in seconds)

It's clear from the graph above that there is no discernable relationship between runtime and S. The graph does not demonstrate any regular pattern between S and runtime, thereby verifying the hypothesis's claim that S does not affect the runtime of ListStrand.

Conclusion: The data illustrated in Figure 5 refutes the hypothesis that the runtime of the method cutAndSplice for using a LinkStrand is O(b), but the data in Figure 7 supports the part of the hypothesis that S does not affect the runtime. Figure 6 also helps to demonstrate that S does not have an impact on runtime.

Other notes:

When implementing my **reverse** method, I used the strategy suggested in the FAQ section of the DNA write up that uses a map with the keys being the un-reversed strings and the values being the reverse of the key. I did this in order to avoid having to re-reverse the enzyme b times—instead, I could just check if the string in the node I was currently looking at was a kay in the map, and if it was, I set the new Node's info to that key's value. This way, I didn't have to use the reverse method built into the StringBuilder an unnecessary amount of times.