Authors:

Samuel Chan, spc28

Vicki Lu, vl54

Ahmed Abualsaud, aka32

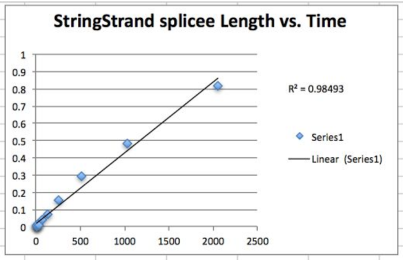
Excel Spreadsheet in the reflect directory for times (found in git repo as well)

Non Linked List Hypothesis

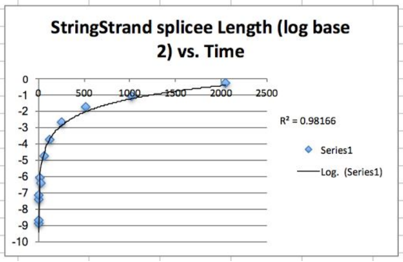
<u>Hypothesis</u>: That big Oh of StringStrand is O(b^2S) and StringBuilder is O(bS)

Both StringStrand and StringBuilder show that as S doubles, the run time doubles, hence S is incorporated into big Oh for both as O(S).

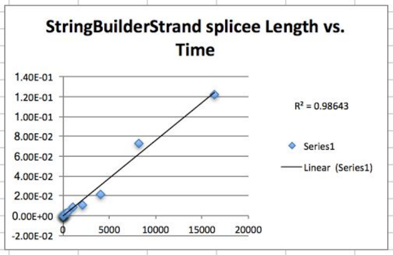
StringStrand	
S	runtime
1	0.002067
2	0.00241135
4	0.00594703
8	0.0069857
16	0.01458004
32	0.01194307
64	0.03789358
128	0.07446573
256	0.15389202
512	0.29533372
1024	0.48364374
2048	0.81598354



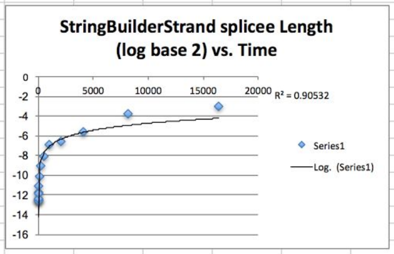
StringStrand	
S	log(runtime)
1	-8.9182459
2	-8.6959426
4	-7.3936157
8	-7.1613802
16	-6.0998611
32	-6.3876827
64	-4.7219027
128	-3.7472796
256	-2.7000097
512	-1.759582
1024	-1.0479834
2048	-0.2933881



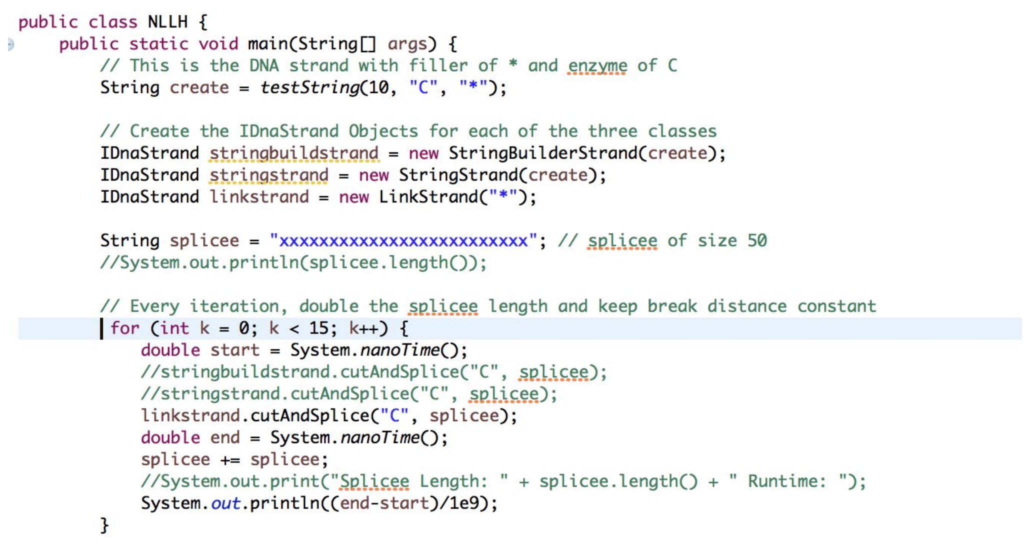
StringBuilder	
S	runtime
1	2.83E-04
2	1.58E-04
4	1.71E-04
8	1.60E-04
16	1.85E-04
32	2.74E-04
64	4.71E-04
128	8.94E-04
256	0.00187877
512	0.00368026
1024	0.00821131
2048	0.0103093
4096	0.02128333
8192	0.07244051
16384	0.12241845



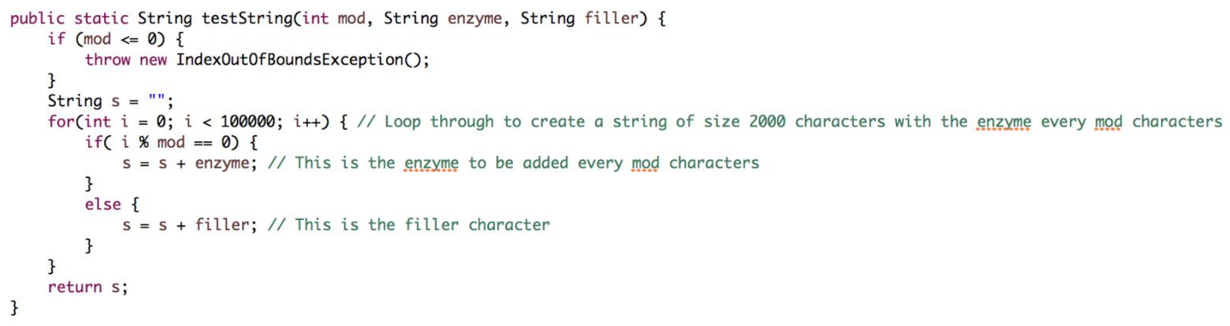
StringBuilder	
S	log(runtime)
1	-11.786778
2	-12.626255
4	-12.511819
8	-12.606173
16	-12.399369
32	-11.833521
64	-11.050605
128	-10.126863
256	-9.0559974
512	-8.0859758
1024	-6.9281719
2048	-6.5999102
4096	-5.5541322
8192	-3.7870595
16384	-3.0301071



The code for this is in my gitlab repository



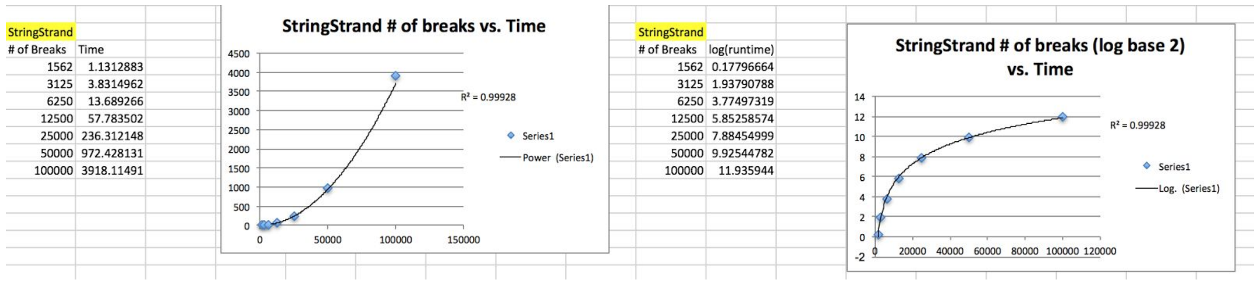
```java
public class NLLH {
    public static void main(String[] args) {
        // This is the DNA strand with filler of * and enzyme of C
        String create = testString(10, "C", "*");

        // Create the IDnaStrand Objects for each of the three classes
        IDnaStrand stringbuildstrand = new StringBuilderStrand(create);
        IDnaStrand stringstrand = new StringStrand(create);
        IDnaStrand linkstrand = new LinkStrand("*");

        String splicee = "xxxxxxxxxxxxxxxxxxxxxxxxx"; // splicee of size 50
        //System.out.println(splicee.length());

        // Every iteration, double the splicee length and keep break distance constant
        for (int k = 0; k < 15; k++) {
            double start = System.nanoTime();
            //stringbuildstrand.cutAndSplice("C", splicee);
            //stringstrand.cutAndSplice("C", splicee);
            linkstrand.cutAndSplice("C", splicee);
            double end = System.nanoTime();
            splicee += splicee;
            //System.out.print("Splicee Length: " + splicee.length() + " Runtime: ");
            System.out.println((end-start)/1e9);
        }
```



```java
public static String testString(int mod, String enzyme, String filler) {
    if (mod <= 0) {
        throw new IndexOutOfBoundsException();
    }
    String s = "";
    for(int i = 0; i < 100000; i++) { // Loop through to create a string of size 2000 characters with the enzyme every mod characters
        if( i % mod == 0) {
            s = s + enzyme; // This is the enzyme to be added every mod characters
        }
        else {
            s = s + filler; // This is the filler character
        }
    }
    return s;
}
```

StringStrand Results for Breaks vs. Time

```
num of breaks: 1562
1.1312883004
num of breaks: 3125
3.8314961678
num of breaks: 6250
13.689266202799999
num of breaks: 12500
57.7835022128
num of breaks: 25000
236.312148
num of breaks: 50000
972.428131
num of breaks: 100000
3918.114912
```



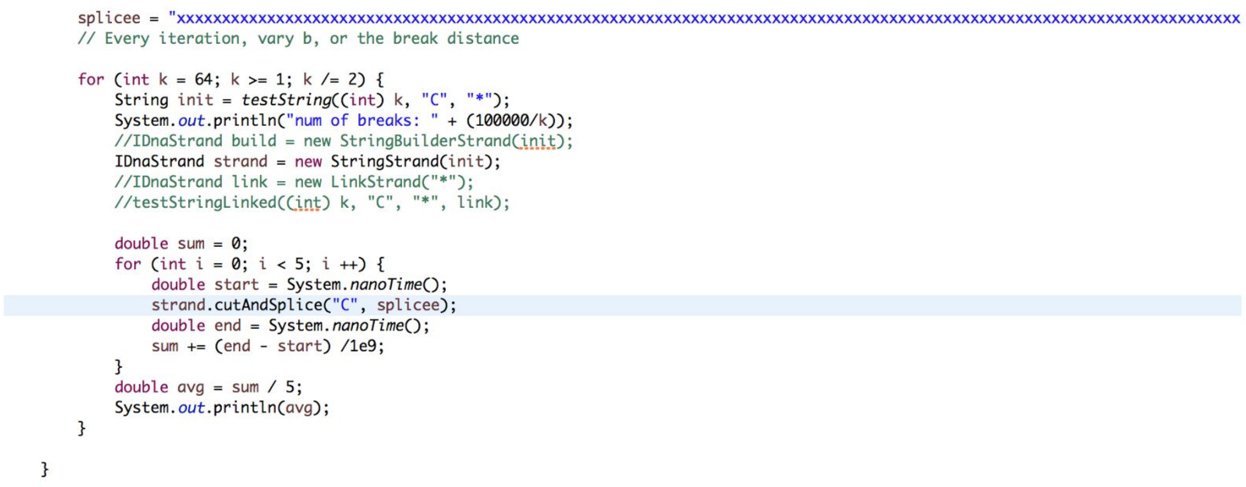
StringStrand

# of Breaks	Time
1562	1.1312883
3125	3.8314962
6250	13.689266
12500	57.783502
25000	236.312148
50000	972.428131
100000	3918.11491

StringStrand

# of Breaks	log(runtime)
1562	0.17796664
3125	1.93790788
6250	3.77497319
12500	5.85258574
25000	7.88454999
50000	9.92544782
100000	11.935944

This clearly shows a quadratic (b^2) correlation between number of breaks (b) and runtime, thus giving us a big oh complexity with respect to b of O(b^2). Putting this together with the big oh complexity with respect to S, we have a total big oh complexity of **<u>O(b^2S)</u>**, which **supports** our hypothesis



```
        splicee = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        // Every iteration, vary b, or the break distance

        for (int k = 64; k >= 1; k /= 2) {
            String init = testString((int) k, "C", "*");
            System.out.println("num of breaks: " + (100000/k));
            //IDnaStrand build = new StringBuilderStrand(init);
            IDnaStrand strand = new StringStrand(init);
            //IDnaStrand link = new LinkStrand("*");
            //testStringLinked((int) k, "C", "*", link);

            double sum = 0;
            for (int i = 0; i < 5; i ++) {
                double start = System.nanoTime();
                strand.cutAndSplice("C", splicee);
                double end = System.nanoTime();
                sum += (end - start) /1e9;
            }
            double avg = sum / 5;
            System.out.println(avg);
        }

    }
```

The code for this is in my gitlab repository

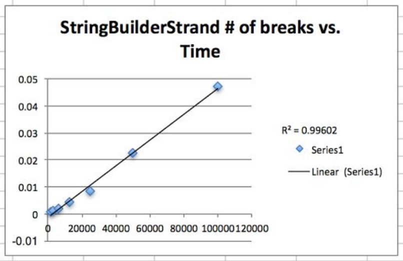
StringBuilderStrand Results for Breaks vs. Time

num of breaks: 1562
0.000783605
num of breaks: 3125
9.719522E-4
num of breaks: 6250
0.0019864088
num of breaks: 12500
0.0043608074
num of breaks: 25000

0.0085997916
num of breaks: 50000
0.0226388976
num of breaks: 10000
0.047036564

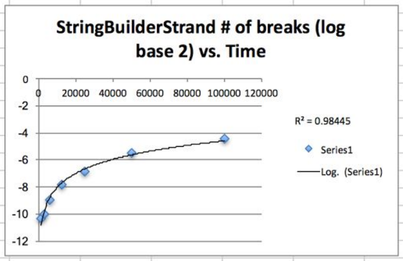
StringBuilderStrand

# of Breaks	Time
1562	0.00078361
3125	9.72E-04
6250	0.00198641
12500	0.00436081
25000	0.00859979
50000	0.0226389
100000	0.04703656

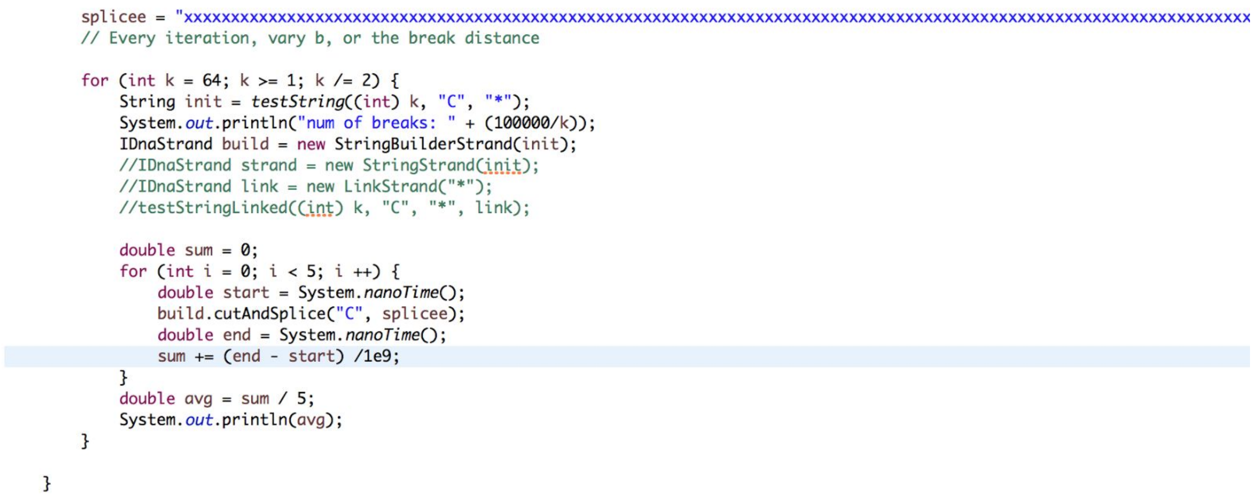


StringBuilderStrand

# of Breaks	log(runtime)
1562	-10.317587
3125	-10.006827
6250	-8.9756217
12500	-7.841189
25000	-6.8614826
50000	-5.4650525
100000	-4.4100735



This clearly shows a linear correlation between number of breaks (b) and runtime, thus giving us a big oh complexity with respect to b of O(b). Putting this together with the big oh complexity with respect to S, we have a total big oh complexity of **<u>O(bS)</u>**, which **supports** our hypothesis.



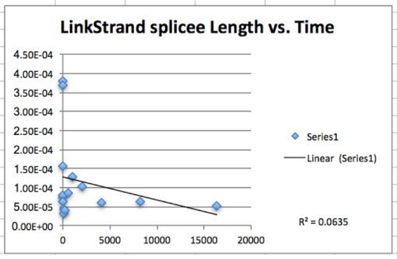
```
        splicee = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        // Every iteration, vary b, or the break distance

        for (int k = 64; k >= 1; k /= 2) {
            String init = testString((int) k, "C", "*");
            System.out.println("num of breaks: " + (100000/k));
            IDnaStrand build = new StringBuilderStrand(init);
            //IDnaStrand strand = new StringStrand(init);
            //IDnaStrand link = new LinkStrand("*");
            //testStringLinked((int) k, "C", "*", link);

            double sum = 0;
            for (int i = 0; i < 5; i ++) {
                double start = System.nanoTime();
                build.cutAndSplice("C", splicee);
                double end = System.nanoTime();
                sum += (end - start) /1e9;
            }
            double avg = sum / 5;
            System.out.println(avg);
        }

    }
```

The code for this is in my gitlab repository

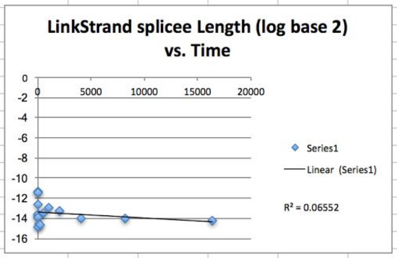
Linked List Hypothesis

<u>Hypothesis</u>: That the big Oh of LinkedStrand is O(b)

LinkStrand	
S	runtime
1	3.81E-04
2	3.68E-04
4	1.56E-04
8	7.48E-05
16	8.13E-05
32	6.39E-05
64	3.31E-05
128	4.24E-05
256	4.00E-05
512	8.67E-05
1024	1.28E-04
2048	1.02E-04
4096	5.98E-05
8192	6.21E-05
16384	5.26E-05

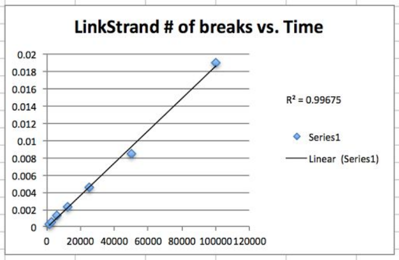


LinkStrand	
S	log(runtime)
1	-11.357085
2	-11.406161
4	-12.64442
8	-13.705696
16	-13.586847
32	-13.933599
64	-14.881154
128	-14.526836
256	-14.611012
512	-13.493575
1024	-12.926607
2048	-13.25708
4096	-14.030388
8192	-13.975535
16384	-14.213865

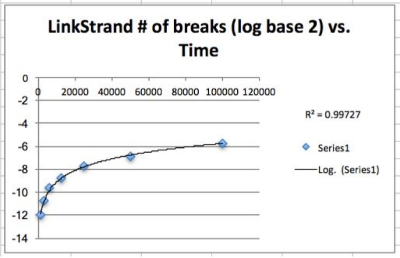


As can be seen from this data, when S is varied, the overall runtime does not change, hence the big oh complexity does not include S

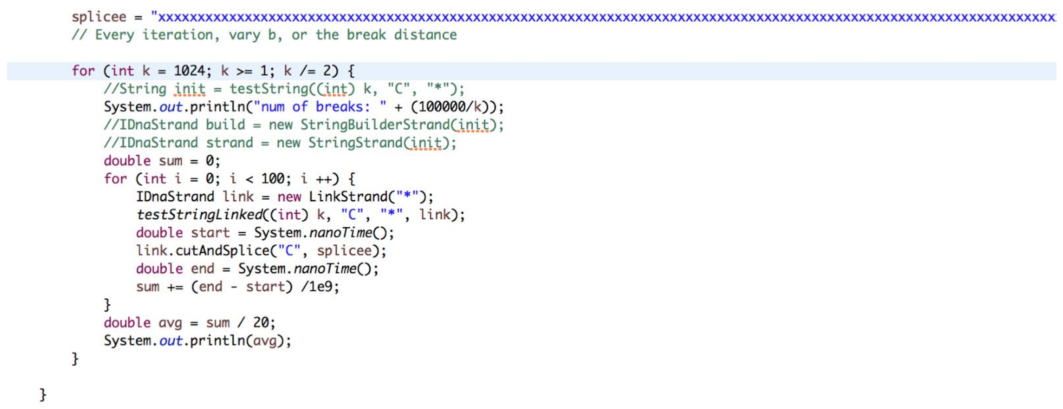
LinkStrand	
# of Breaks	Time
1562	0.00024726
3125	0.00057275
6250	0.0012739
12500	0.0023315
25000	0.00459795
50000	0.00842954
100000	0.01897881



LinkStrand	
# of Breaks	log(runtime)
1562	-11.98167
3125	-10.76981
6250	-9.6165383
12500	-8.7445268
25000	-7.7647939
50000	-6.890331
100000	-5.7194667



As can be seen from the data and the graphs, as # of breaks doubles, so does the time (roughly), which implies that with respect to b (# of breaks), the big oh complexity is **O(b)**. Adding this to the big oh with respect to S, we get that the big Oh of Linked Strand is **O(b)**, which **supports** the hypothesis



```
        splicee = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        // Every iteration, vary b, or the break distance

        for (int k = 1024; k >= 1; k /= 2) {
            //String init = testString((int) k, "C", "*");
            System.out.println("num of breaks: " + (100000/k));
            //IDnaStrand build = new StringBuilderStrand(init);
            //IDnaStrand strand = new StringStrand(init);
            double sum = 0;
            for (int i = 0; i < 100; i ++) {
                IDnaStrand link = new LinkStrand("*");
                testStringLinked((int) k, "C", "*", link);
                double start = System.nanoTime();
                link.cutAndSplice("C", splicee);
                double end = System.nanoTime();
                sum += (end - start) /1e9;
            }
            double avg = sum / 20;
            System.out.println(avg);
        }

    }
```

The code for this is in my gitlab repository

Timing Results

StringBuilderStrand Results

```
dna length = 4,639,221
cutting at enzyme gaattc
-----
Class                    splicee              recomb        time    appends
-----
StringBuilderStrand:         256          4,800,471         0.087   1290
StringBuilderStrand:         512          4,965,591         0.064   1290
StringBuilderStrand:       1,024          5,295,831         0.087   1290
StringBuilderStrand:       2,048          5,956,311         0.033   1290
StringBuilderStrand:       4,096          7,277,271         0.062   1290
StringBuilderStrand:       8,192          9,919,191         0.056   1290
StringBuilderStrand:      16,384         15,203,031         0.146   1290
StringBuilderStrand:      32,768         25,770,711         0.137   1290
StringBuilderStrand:      65,536         46,906,071         0.080   1290
StringBuilderStrand:     131,072         89,176,791         0.532   1290
StringBuilderStrand:     262,144        173,718,231         0.562   1290
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

StringStrand Results

```
dna length = 4,639,221
cutting at enzyme gaattc
-----
Class              splicee         recomb           time            appends
-----
StringStrand:          256       4,800,471          7.199           1290
StringStrand:          512       4,965,591          5.876           1290
StringStrand:        1,024       5,295,831          5.345           1290
StringStrand:        2,048       5,956,311          6.124           1290
StringStrand:        4,096       7,277,271          8.119           1290
StringStrand:        8,192       9,919,191          9.480           1290
StringStrand:       16,384      15,203,031          23.025          1290
StringStrand:       32,768      25,770,711          33.619          1290
StringStrand:       65,536      46,906,071          75.316          1290
StringStrand:      131,072      89,176,791          133.669         1290
StringStrand:      262,144     173,718,231          337.695         1290
```

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

LinkStrand Results

```
dna length = 4,639,221
cutting at enzyme gaattc
-----
Class           splicee         recomb              time    appends
-----
LinkStrand:         256         4,800,471           0.050   1290
LinkStrand:         512         4,965,591           0.052   1290
LinkStrand:       1,024         5,295,831           0.051   1290
LinkStrand:       2,048         5,956,311           0.034   1290
LinkStrand:       4,096         7,277,271           0.042   1290
LinkStrand:       8,192         9,919,191           0.101   1290
LinkStrand:      16,384         15,203,031          0.033   1290
LinkStrand:      32,768         25,770,711          0.040   1290
LinkStrand:      65,536         46,906,071          0.041   1290
LinkStrand:     131,072         89,176,791          0.033   1290
LinkStrand:     262,144         173,718,231         0.033   1290
LinkStrand:     524,288         342,801,111         0.035   1290
LinkStrand:   1,048,576         680,966,871         0.049   1290
LinkStrand:   2,097,152         1,357,298,391       0.072   1290
LinkStrand:   4,194,304         2,709,961,431       0.034   1290
LinkStrand:   8,388,608         5,415,287,511       0.033   1290
LinkStrand:  16,777,216         10,825,939,671      0.052   1290
LinkStrand:  33,554,432         21,647,243,991      0.032   1290
LinkStrand:  67,108,864         43,289,852,631      0.039   1290
LinkStrand: 134,217,728          86,575,069,911     0.029   1290
LinkStrand: 268,435,456         173,145,504,471     0.040   1290
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```