Kenneth Marenco and Jeffrey Cheng

Analysis

Question 1:
Part 1: are the benchmark timings for StringStrand consistent with the explanation below that the time to execute cutAndSplice is O(b^2S)?

Note that the value of b is half the number of calls to append since each cut (except the first) is modeled by two calls of append in the method cutAndSplice -- see the code. This means that b^2 will be constant in the benchmark, but S will vary.

Answer 1 Part 1:
Knowing that b^2 is constant we can analyze how the runtime changes with respect to S changing. S is the size of the splicee, and we see very clearly when the splicee is hundreds of thousands in size that when it doubles the runtime doubles which is keeping consistent with our expectations. This is because Strings are immutable objects which cannot be changed, so to when splicing a new String (more memory) must be used and then populated every time a break occurs. This leads to the most inefficient function out of the three tested.

Answer 1 Part 2:
After modifying the Benchmark program to triple the mySource, we expect b to be tripled which we see from the first few lines of the new Benchmark. This should result in our new runtime of O((3b)^2s). Below are those times which match with our expectations of what would happen when increasing b in an O(b^2s) runtime function.

Results from DNABenchmark using mySource = mySource+mySource+mySource;

```
-----
Class               splicee      recomb     time appends
-----
StringStrand:          256     14,401,413 7.402     3870
StringStrand:          512     14,896,773 8.397     3870
StringStrand:        1,024     15,887,493 6.960     3870
StringStrand:        2,048     17,868,933 9.516     3870
StringStrand:        4,096     21,831,813 11.768    3870
```

Results from DNABenchmark using StringStrand on ecoli.txt

```
dna length = 4,639,221
cutting at enzyme gaattc
-----
Class               splicee      recomb     time appends
-----
StringStrand:          256      4,800,471  0.384     1290
StringStrand:          512      4,965,591  0.356     1290
StringStrand:        1,024      5,295,831  0.359     1290
StringStrand:        2,048      5,956,311  0.418     1290
```

```
StringStrand:           4,096           7,277,271  0.526       1290
StringStrand:           8,192           9,919,191  0.892       1290
StringStrand:          16,384          15,203,031 1.319        1290
StringStrand:          32,768          25,770,711 2.629        1290
StringStrand:          65,536          46,906,071 5.173        1290
StringStrand:         131,072          89,176,791 9.774        1290
StringStrand:         262,144         173,718,231        19.623      1290
StringStrand:         524,288         342,801,111        38.633      1290
StringStrand:       1,048,576         680,966,871        74.557      1290
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
	at java.base/jdk.internal.misc.Unsafe.allocateUninitializedArray(Unsafe.java:1269)
	at
java.base/java.lang.invoke.StringConcatFactory$MethodHandleInlineCopyStrategy.newArray(StringConcatFactory.jav
a:1633)
	at java.base/java.lang.invoke.DirectMethodHandle$Holder.invokeStatic(DirectMethodHandle$Holder)
	at java.base/java.lang.invoke.LambdaForm$MH/0x0000000800149840.invoke(LambdaForm$MH)
	at java.base/java.lang.invoke.Invokers$Holder.linkToTargetMethod(Invokers$Holder)
	at StringStrand.append(StringStrand.java:70)
	at IDnaStrand.cutAndSplice(IDnaStrand.java:37)
	at DNABenchmark.lambda$strandSpliceBenchmark$0(DNABenchmark.java:76)
	at DNABenchmark$$Lambda$163/0x000000080025f840.run(Unknown Source)
	at java.base/java.lang.Thread.run(Thread.java:834)
	at DNABenchmark.strandSpliceBenchmark(DNABenchmark.java:79)
	at DNABenchmark.main(DNABenchmark.java:112)

Process finished with exit code 1
```

Question 2: are the benchmark timings for StringBuilderStrand
consistent with the explanation below that the time to execute
cutAndSplice is O(bS)?

Answer 2 Part 1:
We can see below the output of StringBuilderStrand which we expect to have the same runtimes as in question 2 with StringStrand since b is held constant as can be seen by each append remaining at 1290. The reason why this class is more efficient than StringStrand is because it makes use of StringBuilder which uses a char array to populate the the new strand which since it is primitive does not require as much memory as a String because primitives are mutable.

Answer 2 Part 2:
Below are the first few runs of the Benchmark with b tripled. We can see a clear difference between these runs and the runs when b was 1290. As was expected from theory, the majority of the runs grew larger because the runtime O(bs) would have changed to O(3bs). However as can be seen from closer analysis of the runtimes, the time did not triple exactly which can be assumed to be caused by the state computer system itself during the trials.

Results from DNABenchmark using mySource = mySource+mySource+mySource;

```
-----
Class                   splicee         recomb      time appends
-----
StringBuilderStrand:               256          14,401,413 0.038        3870
StringBuilderStrand:               512          14,896,773 0.039        3870
StringBuilderStrand:             1,024          15,887,493 0.041        3870
```

```
StringBuilderStrand:              2,048         17,868,933 0.043       3870
StringBuilderStrand:              4,096         21,831,813 0.047       3870
StringBuilderStrand:              8,192         29,757,573 0.041       3870
StringBuilderStrand:             16,384         45,609,093 0.056       3870
StringBuilderStrand:             32,768         77,312,133 0.107       3870
StringBuilderStrand:             65,536        140,718,213        0.099       3870
StringBuilderStrand:            131,072        267,530,373        0.275       3870
StringBuilderStrand:            262,144        521,154,693        0.442       3870
StringBuilderStrand:            524,288      1,028,403,333        0.738       3870
```

Results from DNABenchmark using StringBuilderStrand on ecoli.txt

```
dna length = 4,639,221
cutting at enzyme gaattc
-----
Class                    splicee          recomb      time appends
-----
StringBuilderStrand:                256          4,800,471  0.023       1290
StringBuilderStrand:                512          4,965,591  0.018       1290
StringBuilderStrand:              1,024          5,295,831  0.007       1290
StringBuilderStrand:              2,048          5,956,311  0.007       1290
StringBuilderStrand:              4,096          7,277,271  0.008       1290
StringBuilderStrand:              8,192          9,919,191  0.009       1290
StringBuilderStrand:             16,384         15,203,031 0.011       1290
StringBuilderStrand:             32,768         25,770,711 0.022       1290
StringBuilderStrand:             65,536         46,906,071 0.029       1290
StringBuilderStrand:            131,072         89,176,791 0.077       1290
StringBuilderStrand:            262,144        173,718,231        0.101       1290
StringBuilderStrand:            524,288        342,801,111        0.255       1290
StringBuilderStrand:          1,048,576        680,966,871        0.415       1290
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
	at java.base/java.util.Arrays.copyOf(Arrays.java:3745)
	at java.base/java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.java:172)
	at java.base/java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:538)
	at java.base/java.lang.StringBuilder.append(StringBuilder.java:174)
	at StringBuilderStrand.append(StringBuilderStrand.java:70)
	at IDnaStrand.cutAndSplice(IDnaStrand.java:41)
	at DNABenchmark.strandSpliceBenchmark(DNABenchmark.java:67)
	at DNABenchmark.main(DNABenchmark.java:112)

Process finished with exit code 1
```

Question 3: Explain why the time for LinkStrand does not change much at all over all the runs in the benchmark program. Explain why you think memory is exhausted at the specific strand size you see in your timings -- as compared to exhaustion for String and StringBuilder.

Answer 3

The reason that LinkStrand does not change much during all the runs is because we are taking advantage of how nodes work. With each node having info of a sequence of chars, the only work the computer has to do is to change the pointer for the info of the node

which runs at O(1), constant time. Another benefit is that the splicee does not have
to be stored again in memory. Instead the pointers of the nodes can point to
the same splicee.

Results from DNABenchmark using LinkStrand on ecoli.txt

```
dna length = 4,639,221
cutting at enzyme gaattc
-----
Class                   splicee         recomb      time appends
-----
LinkStrand:                 256          4,800,471  0.044      1290
LinkStrand:                 512          4,965,591  0.029      1290
LinkStrand:               1,024          5,295,831  0.011      1290
LinkStrand:               2,048          5,956,311  0.006      1290
LinkStrand:               4,096          7,277,271  0.006      1290
LinkStrand:               8,192          9,919,191  0.007      1290
LinkStrand:              16,384         15,203,031  0.006      1290
LinkStrand:              32,768         25,770,711  0.013      1290
LinkStrand:              65,536         46,906,071  0.006      1290
LinkStrand:             131,072         89,176,791  0.006      1290
LinkStrand:             262,144        173,718,231       0.013      1290
LinkStrand:             524,288        342,801,111       0.006      1290
LinkStrand:           1,048,576        680,966,871       0.007      1290
LinkStrand:           2,097,152      1,357,298,391       0.013      1290
LinkStrand:           4,194,304      2,709,961,431       0.007      1290
LinkStrand:           8,388,608      5,415,287,511       0.010      1290
LinkStrand:          16,777,216     10,825,939,671       0.012      1290
LinkStrand:          33,554,432     21,647,243,991       0.004      1290
LinkStrand:          67,108,864     43,289,852,631       0.015      1290
LinkStrand:         134,217,728        86,575,069,911       0.013      1290
LinkStrand:         268,435,456       173,145,504,471       0.010      1290
LinkStrand:         536,870,912       346,286,373,591       0.010      1290
LinkStrand:       1,073,741,824       692,568,111,831       0.010      1290
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
	at java.base/java.util.Arrays.copyOf(Arrays.java:3745)
	at java.base/java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.java:172)
	at java.base/java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:538)
	at java.base/java.lang.StringBuilder.append(StringBuilder.java:174)
	at DNABenchmark.main(DNABenchmark.java:109)

Process finished with exit code 1
```