**Question 1**

The big O asympototic runtime complexity of cutAndSplice when using StringStrand is

. This is because:

The length of fragments is and the length of splicee is , so the total length would be these two combined, which is . Then looking at the append() method in StringStrand, the runtime complexity can be calculated as in general terms, when is the length of the input string and is the number of times the append is called. Since is the number of fragments, we can substitute this into . Since is the total length of the string, we can substitute this into . Hence, the final big O asymptotic runtime complexity is

.

The empirical data from running this code is as follows:

dna length = 320,160

cutting at enzyme gaattc

----------------------------------------------------------------------

Class dna,N splicee,S recomb time(ms) breaks,b

----------------------------------------------------------------------

StringStra: 320,160 10,000 769,890 29 45

StringStra: 320,160 20,000 1,219,890 32 45

StringStra: 320,160 40,000 2,119,890 30 45

StringStra: 320,160 80,000 3,919,890 47 45

StringStra: 320,160 160,000 7,519,890 81 45

StringStra: 320,160 320,000 14,719,890 156 45

StringStra: 320,160 640,000 29,119,890 308 45

StringStra: 320,160 1,280,000 57,919,890 650 45

StringStra: 320,160 10,000 769,890 28 45

StringStra: 640,320 10,000 1,539,780 37 90

StringStra: 1,280,640 10,000 3,079,560 137 180

StringStra: 2,561,280 10,000 6,159,120 544 360

StringStra: 5,122,560 10,000 12,318,240 2,137 720

The empirical data reasonably conforms to our theoretical calculation because the quadratic effect of is reflected in the actual runtime, and the linear effect of is also reflected in the actual runtime. Minor discrepancies may be due to our small sample size or the lack of data points.

**Question 2**

The big O asymptotic runtime complexity of cutAndSplice when using StringBuilderStrand is . This is because:

The length of fragments is and the length of splicee is , so the total length would be these two combined, which is . Then looking at the append() method in StringBuilderStrand, the runtime complexity is linear this time because the for loop simply runs for times in general terms, when is the number of fragments. This runtime complexity should then be multiplied by , when is the length of the input string. Since is the total length of the string, we can substitute this into . Hence, multiplying these two, the final big O asymptotic runtime complexity is , which can be written as .

The empirical data from running this code is as follows:

dna length = 320,160

cutting at enzyme gaattc

----------------------------------------------------------------------

Class dna,N splicee,S recomb time(ms) breaks,b

----------------------------------------------------------------------

StringStra: 320,160 10,000 769,890 24 45

StringStra: 320,160 20,000 1,219,890 29 45

StringStra: 320,160 40,000 2,119,890 28 45

StringStra: 320,160 80,000 3,919,890 43 45

StringStra: 320,160 160,000 7,519,890 86 45

StringStra: 320,160 320,000 14,719,890 160 45

StringStra: 320,160 640,000 29,119,890 297 45

StringStra: 320,160 1,280,000 57,919,890 610 45

StringStra: 320,160 10,000 769,890 12 45

StringStra: 640,320 10,000 1,539,780 39 90

StringStra: 1,280,640 10,000 3,079,560 146 180

StringStra: 2,561,280 10,000 6,159,120 551 360

StringStra: 5,122,560 10,000 12,318,240 2,089 720

The empirical data, however, only matches our theoretical calculation to a limited extent. While the linear effect of is reflected in the actual runtime, the linear effect of is not reflected as runtime seems to be closer to a quadratic increase. This discrepancy could be due to our small sample size or the lack of data points to display a more accurate trend.

**Question 3**

Since each character of the String takes 1 byte of memory to store, the total memory required to store the result of the cutAndSplice operation on the StringStrand object would be equal to the number of characters. The number of characters in StringStrand object is because the for loop in cutAndSplice is iterated times, and the splicee length is added each time, leading to . is added to this because it is the total number of bytes from search.split. Therefore, bytes of memory would be needed to store the result on the StringStrand object.

When we Using a StringBuilderStrand, however, more memory will be needed. This is because StringBuilderStrand doubles the internal buffer when it exceeds the initial array capacity, just like an ArrayList. Hence, more memory will be required.

**Question 4**

The big O asympototic runtime complexity of cutAndSplice when using LinkStrand is . This is because:

The to.String() method only iterates once, running through the length of the string which is , so it has a runtime of . The getInstance() method has constant runtime, and split also has a runtime of . The append() method in LinkStrand has constant runtime because it simply creates a new node and points the last node to it. The append() method will iterated for times due to the for loop, leading to a runtime complexity of . Adding these all together, the runtime complexity would be , which can be abbreviated as . Hence, the final big O asymptotic runtime complexity is . From this runtime, we can see that LinkStrand is overall a much more efficient method than StringStrand or StringBuilderStrand in terms of runtime.

The empirical data from running this code is as follows:

dna length = 320,160

cutting at enzyme gaattc

----------------------------------------------------------------------

Class dna,N splicee,S recomb time(ms) breaks,b

----------------------------------------------------------------------

StringStra: 320,160 10,000 769,890 32 45

StringStra: 320,160 20,000 1,219,890 26 45

StringStra: 320,160 40,000 2,119,890 26 45

StringStra: 320,160 80,000 3,919,890 41 45

StringStra: 320,160 160,000 7,519,890 71 45

StringStra: 320,160 320,000 14,719,890 131 45

StringStra: 320,160 640,000 29,119,890 271 45

StringStra: 320,160 1,280,000 57,919,890 578 45

StringStra: 320,160 10,000 769,890 25 45

StringStra: 640,320 10,000 1,539,780 33 90

StringStra: 1,280,640 10,000 3,079,560 115 180

StringStra: 2,561,280 10,000 6,159,120 460 360

StringStra: 5,122,560 10,000 12,318,240 1,864 720

The empirical data reasonably conforms to our theoretical calculation because the linear effect of both and are reflected in the actual runtime. For instance, when doubles and doubles, runtime is shown to quadruple as it is linearly affected by both variables. Minor discrepancies may be due to our small sample size or the lack of data points.

The empirical data, however, doesn’t match our theoretical calculation accurately, as the linear effect of is not clearly reflected in the actual runtime. For instance, when is doubled, runtime seems to almost quadruple, emulating a quadratic effect. It also seems to be slightly affected by other variables. Such discrepancies could be due to our small sample size or the lack of data points to display a more accurate trend.

**Question 5**

Since the number of breaks in the enzyme is , we would split the string times, creating a total number of nodes, where 1 is added due to the pointer to the null at the end. As 8 bytes of memory are required to store each node, bytes of memory are needed to store all these nodes. The total memory needed is equivalent to the total number of characters since each character takes 1 byte of memory to store, so we should add (length of DNA) and (length of splicee) to this value. Therefore, we would need a total of bytes of memory.