This document contains test reports of all fellow classmates and explanations of motivations behind each test. Tests will be for correctness, memory usage and timing when applicable.

1 Anagrams

1.1 Correctness:

To test for correctness we had seven test cases:

Case	String
1	‘‘a’’
2	‘‘abc’’
3	‘‘aaa’’
4	‘‘cabcab’’
5	‘‘annamalis’’
6	‘‘aaaaaaaabb’’
7	‘‘qwertyuiop’’

The reasoning behind each case is as follows:
Note: For the last 3 cases, we know that the java HeapSpace = 2,097,152 bytes and that $10!$ *= 3,628,800 where as* $9!$ *= 362,880.*

1. *Base case to see if the algorithm will print out just a single string.*
2. *Just a simple input meant to easily check for correctness and to see if the algorithm is overkill when it comes to time complexity and/or space usage.*
3. *This is meant to test and see if the student's code can handle a string of all the same character, yet still be an easy case.*
4. *Tests students ability to handle unsorted and longer strings with repeating letters.*
5. *We use "annamalis" so as to test the 9 character correctness aspect. If the students were to use all of the space and store repeats, it would not time out in the system, but it would show up in our space analysis section where as if they did not store the extra strings, it would also show in the space analysis.*
6. *Expect any algorithm which requires storage of all of the numbers to fail this case*
7. *If students ignore all of the repetitions, they would pass this case since it would only be* $\frac{10!}{8!\cdot 2!}$ *= 45 strings being stored.*

The results were:

Group name	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
Lyle & Derek	Passed	Passed	Passed	Passed	Passed	Passed	Passed?
Viet & Christ	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Jon & Trung	Passed	Passed	Passed	Passed	Passed	Passed	Passed?
Colin & Rafael	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Jiri & Ronak	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Josh & Parker	Passed	Passed	Passed	Passed	Passed	Passed	Passed?
Brenda & Brendan	**Failed**	Passed	Passed	Passed	Passed	Passed	**Failed**
Lukas & Sharp	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**

The reason why we put "Passed?" for case 7 is because our program was unable to generate a complete list to check for correctness, however when compared to eachouther's output, we found them all to be the same. For everyone else who failed this case, it was because they had an Out of Memory error caused by exceeding the heap space allotted for the program.

1.2 Timing:

The results were (seconds):

Group name	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
Lyle & Derek	**0.002**	**0.008**	**0.007**	**0.016**	**0.956**	**0.014**	**87.732**
Viet & Christ	0.308	0.196	0.088	0.133	1.384	0.080	—-
Jon & Trung	0.164	0.063	0.089	0.106	225.2	5.987	119.6
Colin & Rafael	0.128	0.074	0.098	0.085	1.136	0.081	—-
Jiri & Ronak	0.189	0.097	0.091	0.119	1.920	0.124	—-
Josh & Parker	0.060	0.057	0.055	0.075	1.225	0.088	108.1
Brenda & Brendan	0.111	0.084	0.083	0.109	1.171	0.118	—-
Lukas & Sharp	0.076	0.080	0.080	0.124	2.152	0.091	—-

For this Section, Lyle and Derek had the most optimized code for Timing due to the fact that they printed straight out onto the command line. Apart from this everyone else had very similar run time with the exception of Jon and Trung's code. This was mostly due to the fact that they consistently did checks for different cases whenever they encountered a new enumeration of the word i.e. if every character is the same.

1.3 Memory:

The results were (percent used of total memory):

Group name	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
Lyle & Derek	**3.4%**	**3.5%**	**3.4%**	**4.0%**	**16.9%**	**4.0%**	**19.2%**
Viet & Christ	5.0%	5.1%	5.0%	5.6%	39.0%	5.6%	—-
Jon & Trung	5.1%	5.1%	5.1%	6.2%	23.2%	24.2%	25.3%
Colin & Rafael	5.1%	5.0%	5.1%	5.6%	40.6%	5.6%	—-
Jiri & Ronak	5.9%	5.9%	5.9%	7.0%	78.8%	7.0%	—-
Josh & Parker	5.1%	5.1%	5.1%	5.6%	24.2%	5.6%	25.3%
Brenda & Brendan	5.5%	5.5%	5.5%	6.1%	38.9%	6.1%	—-
Lukas & Sharp	5.2%	5.2%	5.2%	5.6%	40.5%	5.8%	—-

For this Section, Lyle and Derek had the most optimized code again for Memory due to the fact that they printed straight out onto the command line. Apart from this everyone else had very similar space usage with the exception of Jon and Trung's code when it came to case 6. We had expected to see a similar drop in memory usage for case 6, but because their code stored extra information for each word Jon and Trung's algorithm did significantly worse than all other groups.

1.4 Summary:

Group name	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
Lyle & Derek	Passed	Passed	Passed	Passed	Passed	Passed	Passed
Viet & Christ	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Jon & Trung	Passed	Passed	Passed	Passed	**Failed**	**Failed**	Passed
Colin & Rafael	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Jiri & Ronak	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Josh & Parker	Passed	Passed	Passed	Passed	Passed	Passed	Passed
Brenda & Brendan	**Failed**	Passed	Passed	Passed	Passed	Passed	**Failed**
Lukas & Sharp	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**

The only groups to pass all cases were Lyle & Derek and Josh & Parker.

2 Turtle Tokenizer

To test for correctness we had seven test cases:

Number	String
1	‘‘f10 R120’’
2	‘‘g10’’
3	‘‘x2{f60x30{10R60L10f}g16}’’
4	‘‘f60x30{10R60L10f}g16’’
5	‘‘10R60L10f’’
6	‘‘yolo’’
7	‘‘x2{f60x30{10R60L10f}g16’’

The reasoning behind each case is as follows:

1. *Base case to see if the algorithm will correctly parse two basic commands with camel case*
2. *An invalid entry which has numbers still to see if the program will output anything*
3. *Tests if the student's code can correctly return a multiplyer command with its corresponding braces, with nested braces.*
4. *Tests the student's code ability to return multiple commands with a multiplyer and an invalid entry*
5. *Tests if the code identifies a missing letter at the beginning and the precense of a stand alone valid entry*
6. *Tests to see if students ignore invalid single letters but can pick up on valid single letter entries*
7. *Tests to see if the students code is able to catch an illegal entry (missmatched brackets) and throw the correct exception*

The results were:

Group name	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
: Lyle & Derek	**Failed**	**Failed**	**Failed**	**Failed**	**Failed**	**Failed**	**Failed**
Viet & Christ	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Jon & Trung	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Colin & Rafael	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Jiri & Ronak	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Josh & Parker	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Brenda & Brendan	Passed	Passed	Passed	Passed	Passed	Passed	**Failed**
Lukas & Sharp	Passed	Passed	Passed	Passed	Passed	Passed	Passed

Lyle & Derek print everything out two times due to a print statement within their code; if commented out they would pass cases 1 and 3. They fail other cases because they return invalid tokens and do not throw the correct exception when mismatched brackets are inputed.

2.1 Timing & Memory:

For this file, there was little to no difference in run time or memory usage as all had around 0.001 seconds to run each case and took 2.6% memory

2.2 Summary:

The only groups to pass all cases were Lyle & Derek and Josh & Parker.

3 Turtle Graphics