James Truong

CSS501A with Professor Kim

Program2 – Time Complexity Analysis

**Part 1 using default input.txt**

X is ACCGTCTTAGCGATCAACACATTTAACAACGCGCCGCACCCCCCGTCAAACGAGCTTTTGGGCTCTTGTCCTTTTACAAGCTTCACGACGCATACAGCCTTGATCAACGGTTTGATCTGTCTCCCTTCAGCTGGCTTTAAAGGACATACATATGAAGGCCTTAATAAGGTCCGGGAACTCCACATATTCGGTACTGGGCAAACCCCATGAACCACCTCAACATGAAGAGTCCGAGGACTCTCACGATCCACCAATGCAGATCGGAACTGTGCGATCGCGTAATGAGCCGAGTACTTGGTTTGTGTTTAGGTTATGGGGGCCGGGAGCCGGTTCAATATAAGGAAGTAGTTGCAGATTAGTTGTTGCGAACGGTCATAAATTTGATGGGTAAACGTGAACTTAACAAACCGTGATAGCTAATCCTATGCATCCCTTACGTGGATCGACTCGAGTACCCAGGTGAACCGACTACTTGATAACCGGAAATCGCGGTATAAAAGCGCTCACGGTCAGGAGATATACCTCCAAGCAGTAGTCTTTCTGAGCCTAGAGTAGTAAATTACAGGGACGATGTCTTTTACCGAGGCAACATTTTATTGAGAATCACATGAGGCACAGGTAAAGGCGACATCACGATCGAGATCAACCCCTACTTGTTCAAAACATTGAGAACCAGCTCTGTTTTGGAACCTAGAAAGATAACGCATCCGCTTGATATTCCACGGCTTGTCCCTCTTGTGCGGTCCATCTATCGGAGTTTCCTCCGATACGACCCGCAATGTTTCCAGGCGTACGGTACTTTATGAATACACTCGCGCTGTAACCTGTTATGTGAAACACACACGACAGAGCTTCGCGTGGGCCCAGCGACCCGGTAATACTACATCACCGCACACGACCTCGAGCAGTCTTTGCCGGCGTCCGTAAGTAGTCTAAAGTTGTGTTGATGCTTGGGGTTAAAGCTAAATCGTCCGCAGAATACGACTCTCATCCCAAT

Y is ACCCGCACGCGCTTTGGTCTAGATTCTAGCTCCAACTTGCCTGCTAGATACTCTGTTAAAAGATGGTTTTACAACCCCCTCCTCTGTCCCTGGGGTATTATATAATACGTCGGATAGTCAGGTACAAATACAAGTGGGTGGGAATACTTTTCCTCGGATCCTAGACCACGGATTACTGCGTGGTTGACAAGAGTCGGCCCGGAGGGAAACGTGAAGGTTAGTGCAATTAAAGTCTCTAATGTGAAGCCTCCGCGAAGCGAGGAGTTTCTGAGATCGAGTACTATTTAGAGTTCGAAATCACGGCTTAACCTCACTGCCACGCATAACTTGCCGGCAATCCAGTTTTGCAACGATACTTAATTTGTGCAGCTCATCTTTGCTGTCCAGAAATAGAGCTAGTCGATCTCATCTTGCGGGTAGCCAGAAGTCCTACCGTCTCCTCCATGTAGCTTAAAAATTTCGGTGAGGATCAAAAATGATAAACGTGACAGGTAAGCTCCTACGTCTATCCTATGACCCCCGCGGCAGAATAGGTTGGTAGTGTTAGTGCGTGAGCTGGTAGAATAGAGCACACTTAGGGAAACGGGAACCGTTATGTAGGGCTGCGACACACAAAAAAGTGTTCGTTGGTAAGCTGCCTCTCCACTAAACAGGATTTCTCTGGATGATCCCATCGAAGCAAGTTACGCACCACGCCGAGGCGGACCCTGGTACTAGCTGCCCCCCCCTTTATGGGGCGCTCGTACATCAAGATGATCGCGGACTCAACCTGATTACGAGTTGTCCAAGTAGTCCAGGGTAAGAGAAACTGGAGAGA

LCS length is 573

LCS is ACCGTTTCGATCAAACGCGCCCTAAAGATGGTTTTACAACCCCCCTTCGGTTTATCTGTTCAGAAATACATGGGAATAGGTCCACCACATATTGGTTGCAAACCCCAAACAATGAAAGTCCGGACCTCCGAAGCGAGGACTGGATCGGTAATGAGCATCAGGTTATGCCGCTAATTGCAATAGTTTTGCAACGCTAATTTGTGGAGTGCAAAAAGAGCTATCATCTCCTTCGGGACCGAGTCCTACCGCTCTTGATAAAAATCGGGTATAAAAGCAGGTAAGCTCCACACTTCAGTAGGTAGTTTACGTGCAGAATAGAGAATAGGGAAGGGAACCGATAGCAACACAAAAAAGTGTTTTGGAACTGCCTCCTAACAGGTTTCCTGTCCCATCAAGTTCCCACGCCGCCTGGTACTTGCCCCCCCTTTATGTGAACAAGAGATCCGCCAACCTAGAGGTCCAAGTAGTCCGGGTAAAGAAATGGAGAGA

**Part 2 – Time Analysis of F\_it(n) and F\_re(n)**

Page 3 - F\_it(n) graph, F\_re(n) graph

Page 4-6 - F\_it(n) vs [logn, n, nlogn, n^2, n^3, 2^n] graphs

Page 7-9 - F\_re(n) vs [logn, n, nlogn, n^2, n^3, 2^n] graphs

Page 10-13 - F\_it(n) growth rate ratio graphs analysis

Page 14-15 - F\_re(n) growth rate ratio graphs analysis

**Analysis:** Based on the graphs showing F\_it(n) and F\_re(n), we can determine that F\_re(n) has an asymptotically faster growth rate than F\_it(n) as the input size n gets larger and larger. However, to determine time complexity of each algorithm, we need to do further analysis.

We can see on page 3, O(logn) is a lower bound and O(n) is an upper bound to F\_it(n) so it is likely that the function does not have a tight upper bound greater than O(n) so we can omit O(nlogn), O(n^2), O(n^3), and O(2^n) for now. Looking at the growth rate ratio on page 10, it appears that O(logn) is unlikely to be correct seeing how the ratio is increasing as n gets larger suggests that one of the growth rates dominates the other by some factor n. However, looking at the graph for O(n)/F\_it(n) on page 11, we can see that as n gets larger, the growth rate ratio looks almost constant, but we need a closer look because we don’t really care about small n for time complexity and seeing that spike in the beginning suggests that one function is significantly larger than the other resulting in the huge growth ratio in the beginning. Even the close up for n greater than or equal to 200 shows that there is a slight curve at the beginning so we should take a look at O(nlogn)/F\_it(n) and O(n^2)/F\_it(n) to check. On page 12, the growth rate ratio looks constant in the first graph, but when we take a closer look as n gets larger, we can see there is a similar downward slope to the O(n)/F\_it(n) growth ratio. On page 13, the ratio is almost constant except for smaller n for O(n^2)/F\_it(n) so we take a closer look when n > 200 and we can see that the line is actually constant as n gets larger. Therefore, I can conclude like F\_it(n) is some function of n^2 so it has a time complexity of O(n^2).

Taking a look at the F\_re(n) vs other O(n) functions pages, we can see on page 9, that F\_re(n) has a similar growth rate to O(2^n) and O(n^3) so both can serve as upper bounds to F\_re(n). As for the lower bound, we can check O(nlogn) first since the graphs are very similar for O(logn) and O(nlogn) meaning we can omit those seeing that they have smaller growth rates compared to O(nlogn). On page 14, we can see that it is unlikely that F\_re(n) has a similar growth rate to O(nlogn) since their growth rate ratio is increasing. However, on page 15, estimating F\_re(n) growth rate is much more difficult seeing how similar the graphs are. However, seeing that as n increases, we get a smaller peak for O(2^n)/F\_re(n) versus O(n^3)/F\_re(n) so that would mean the growth rates of O(2^n) and F\_re(n) are closer and similar so I estimate that F\_re(n) has a time complexity of O(2^n). The range of the growth rate values for O(2^n)/F\_re(n) is much smaller than O(n^3)/F\_re(n) as well which suggests the growth rates are similar. Another thing to note is that as n increases, the graph F\_re(n) vs O(n^3) looks like F\_re(n) would cross O(n^3) when n is greater than 3 in growth rate if we were to continue to plot data points so it would suggest that O(n^3) is not an upper bound to F\_re(n), but a lower bound since O(n^3) does not encompass F\_re(n). With that in mind, we can look at F\_re(n) vs O(2^n) graph on the same page and see that as n increases, the lines appear to be parallel so it is likely that O(2^n) is an upper bound to F\_re(n) so that would mean F\_re(n) would have a time complexity of O(2^n). The plotted lines look very similar as well.

In conclusion, I estimate that F\_it(n) has a time complexity of O(n^2) and F\_re(n) has a time complexity of O(2^n) based on the graphs and my analysis. However, the functions F\_it(n) and F\_re(n) both ran in real time on the repl.it website and their recorded outcomes and runtimes are factored on repl.it’s system resources at the time so keep in mind that running F\_it(n) and F\_re(n) may have different numbers and runtimes on repl.it or on your system.





















