Project 1: Substrings and Dates

Members:

Shaleen Mathur - shaleenmathur26@csu.fullerton.edu

Dania Nasereddin - dnasereddin@csu.fullerton.edu

<u>Hypothesis for Reformat Date:</u> For large values of n, the mathematically-derived efficiency class of an algorithm accurately predicts the observed running time of an implementation of that algorithm.

Pseudocode for Re Format Date Algorithm:

```
     def reformat_date(input): #reformat date function
          Output = " "
          Months = [all the months Janurary-December]

          Type = 0
          Month = " "
          Year = " "
          Day = " "

          For every character in input:
               If character at current index is a number:
                    Throw invalid input

               If character at current index is '-':
                    Type = 1

               Else if character at current index is '/':
                    Type = 2

               Else if there is a month with a 3-letter
abbreviation:
          Type = 3

               Else if there is a month with full abbreviation:
                    Type = 4


          If Type == 1: #date type with the
               Break up the day, month, and year into their own
part
               Get rid of all empty space from year, day, and
month
```

```
                Convert day, month, and year into the
     digit-format of it (ex. June = 06)

                If all 3 parts are valid within the range, set
           that to output


           If Type == 2: #date with the '/' format
                Break up the day, month, and year into their own
                part

                Break up the day, month, and year into their own
                part

                Get rid of all empty spaces from the date

                Convert day, month, and year into the digit
                format of it

                If all 3 parts are valid, set that to output


           If Type == 3:
                Break up the day, month, and year into their own
           part
                Remove white space

                Convert day, month, and year into their digit
           format

                If 3-letter abbreviation of the month is in the
           Months array, set a temp int variable to the next
           index

                If the date is valid, make that the output


           If Type == 4:
                Break up the day, month, and year into their own
                thing

                Remove white space
```

```
          Convert day, month, and year into their digit
     format

          If Month is in the list of months, set the temp
     int variable to the next index

          If the date is valid, make that the output


Return output
```

<u>Run-Length Encode</u>

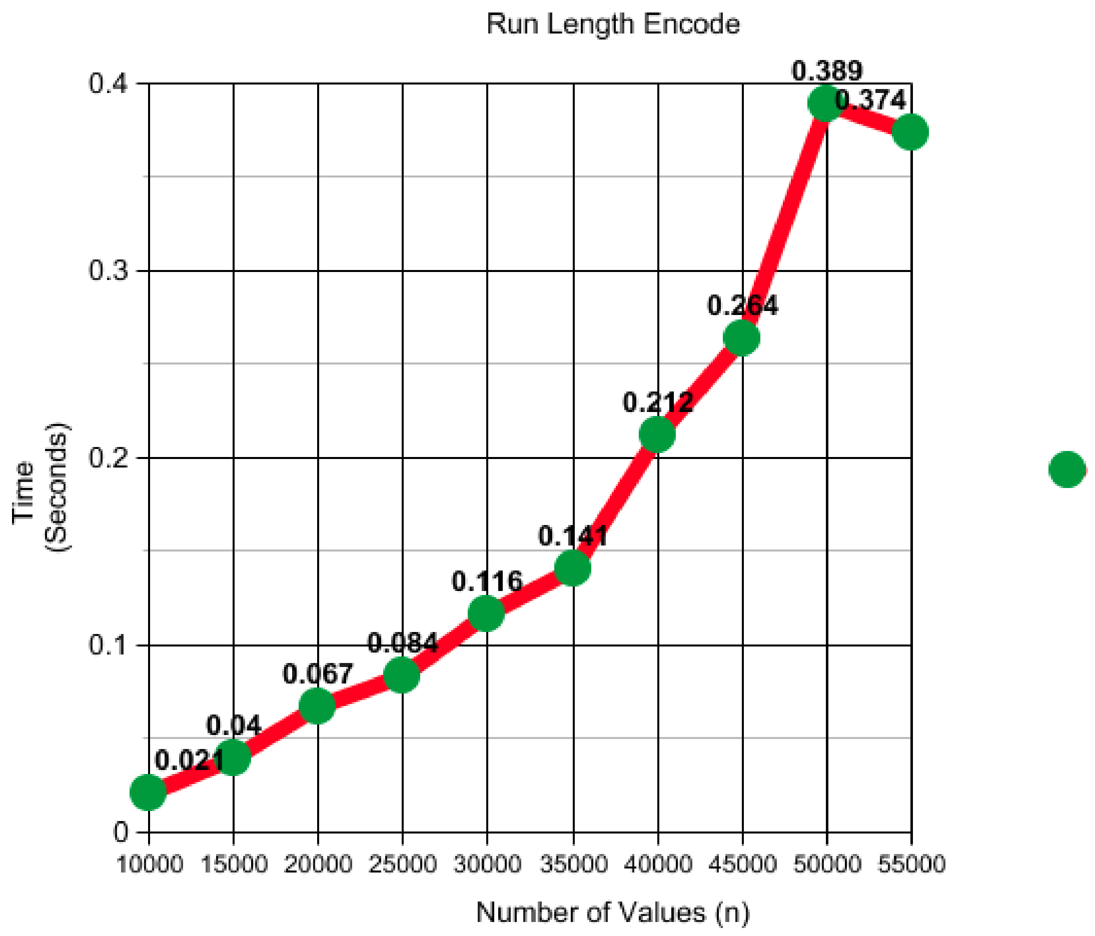
Input: A string *S* of *n* characters, where each character is a lower-case letter or space

Output: a string *C* where each run of k repetitions of the character *x* is replaced with the string "*kx*"

Mathematical Analysis of Run-Length Encode
Using chronological step count: $5 + n + 12 + 5 + 1 + 1$ $= n + 24$ $= n + 24$ **Use Properties of O to prove:** $T(n) = n + 24 \in O(n + 24)$ **(trivial)** $= O(n)$ **(dominated terms)** $= O(n)$ **(constant factor)**

Scatter Plot for Run Length Encode:

Number of Values (*n*)	**Time (*seconds*)**
10000	**0.021**
15000	**0.040**
20000	**0.067**
25000	**0.084**
30000	**0.116**
35000	**0.141**
40000	**0.212**
45000	**0.264**
50000	**0.389**
55000	**0.374**



]

Input: a string *S* of *n* characters, and integers $k \geq 0$

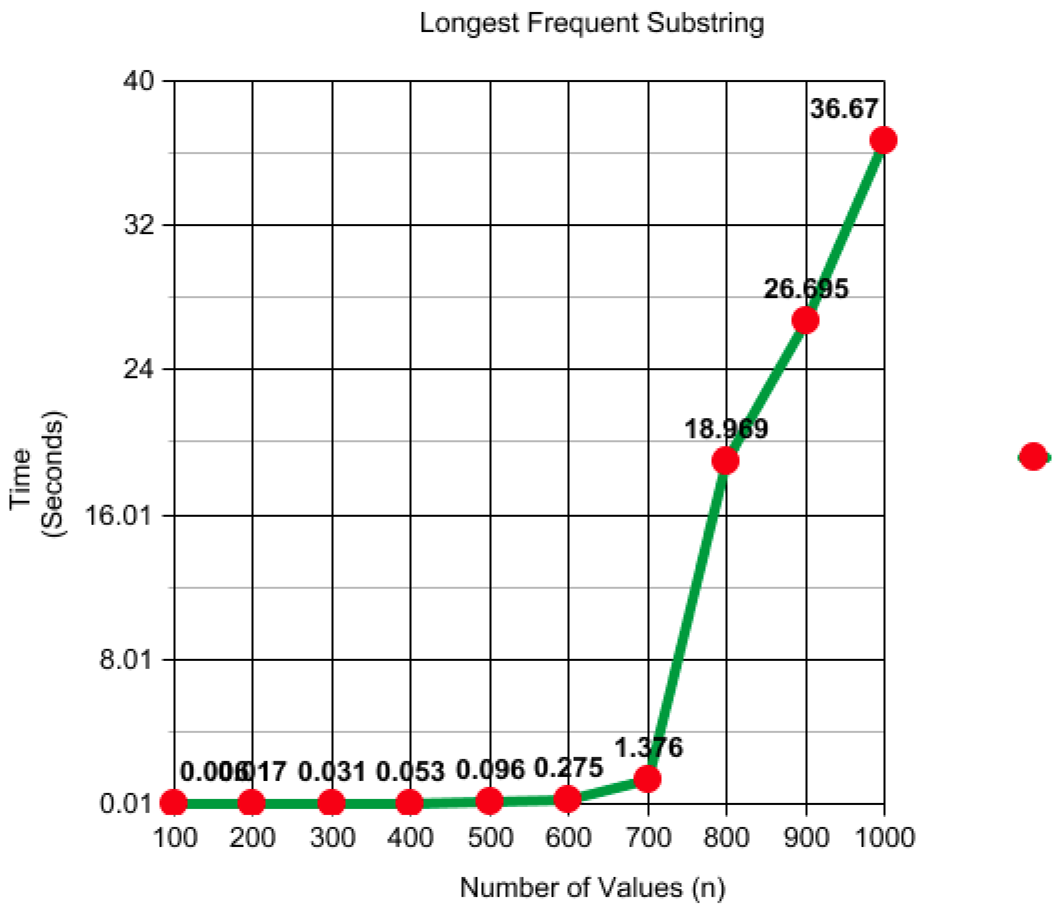
Output: the longest string *B* such that *B* is a substring of *S*, and every character in *B* appears at least k times in *S*

Mathematical Analysis for Longest Frequent Substring
Using chronological step count: $9 + n(4) + 4 + n(n) + n(n+7) + 1$ $= 9 + 4n + 4 + n^2 + n(n+7) + 1$ $= 9 + 4n + 4 + n^3 + n^2 + 7n^2 + 1$ $= n^3 + 8n^2 + 4n + 14$ **Use Properties of O to prove:** $T(n) = n^3 + 8n^2 + 4n + 14 \in O(n^3 + 58 + 4n + 14)$ **(trivial)** $= O(max(n^3, 8n^2, 4n, 14)$ **(dominated terms)** $= O(n^3)$ **(dominated terms)** $= O(n^3)$ **(constant factor)**

Scatter Plot for Longest Frequent Substring:

Number of Values (n)	**Time (seconds)**
100	**0.006**
200	**0.017**
300	**0.031**
400	**0.053**
500	**0.096**
600	**0.275**
700	**1.376**
800	**18.969**
900	**26.695**

1000	**36.670**


Longest Frequent Substring
Time (Seconds)
40
32
24
16.01
8.01
0.01
100
200
300
400
500
600
700
800
900
1000
Number of Values (n)
0.006
0.017
0.031
0.053
0.096
0.275
1.376
18.969
26.695
36.67

Re-Format Date:

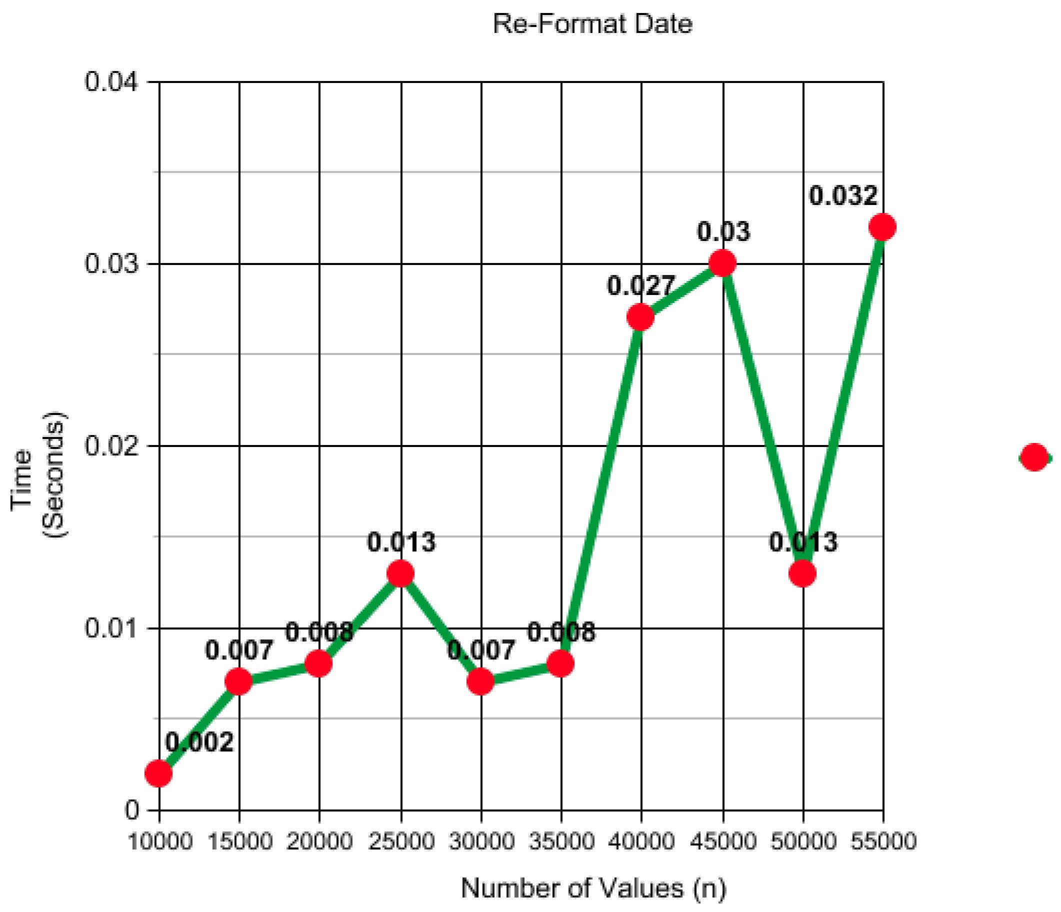
Input: a string *S* that may fit in pattern 1,2,3,4

Output: a string *D* that contains the same date in YYYY-MM-DD format; or throws an exception if *S* is invalid

Mathematical Analysis for Re-Format Date
Using chronological step count: $5 + 6 + n(5) + n(6) + 12$ $= 5 + 6 + 5n + 6n + 12$ $= 11n + 23$ $= 11n + 23$ **Use Properties of O to prove:** $T(n) = 11n + 23 \in O(11n + 23)$ **(trivial)** $= O(11n)$ **(dominated terms)** $= O(n)$ **(constant factor)**

Scatter Plot for Re-Format Date:

Number of Values (n)	**Time (seconds)**
10000	**0.002**
15000	**0.007**
20000	**0.008**
25000	**0.013**
30000	**0.007**
35000	**0.008**
40000	**0.027**
45000	**0.030**
50000	**0.013**
55000	**0.032**



Questions:

1. What is the efficiency class of each of the algorithms, according to your own mathematical analysis? (You are not required to include all your math work, just state the classes you derived and proved.)
    a. **The efficiency class of the Run Length Encode algorithm is O(n), the efficiency class of the Longest Frequent Substring algorithm is O(n^3), and the efficiency class of the Re-Format Date algorithm is O(n).**

2. Between the run-length encode and longest substring algorithms, is there a noticeable difference in the running speed? Which is faster, and by how much? Does this surprise you?
    a. **Between the run-length encode and longest substring algorithms, there is a noticeable difference in the running speed because the efficiency class of the run-length encode is O(n), while the efficiency class for longest frequent substring is O(n^3). Additionally, by looking at the given plots for the scatterplot, you can see that some values of n take much longer on the longest-frequent substring compared to the run-length encode. So, we can conclude that the run-length encode algorithm is much faster than the longest-frequent substring algorithm. This does not surprising because once the mathematical analysis was concluded, we can determine that the efficiency class would be O(n^3).**
3. Are the fit lines on your scatter plots consistent with the efficiency classes predicted by your math analyses? Justify your answer.
    a. **Yes, they are consistent with the efficiency classes predicted by our math analysis because the run-length encode and reformat date have a linear line, and that is consistent with their O(n) efficiency class. The Longest Frequent substring graph was the best fit line but was gradually going up cubically, which is part of the O(n^3) efficiency class.**
4. Is all this evidence consistent or inconsistent with the hypothesis stated on the first page? Justify your answer.
    a. **Yes, all the evidence collected is consistent with the original hypothesis because of the mathematical analysis conducted for all three algorithms. We**

predicted that the time efficiency for the run-length encode and the reformat date algorithms to be O(n) and the time efficiency for the longest frequent substring algorithm would be O(n^3). The scatterplots of all three algorithms also prove that the original hypothesis is consistent because of the best fit lines and the cubic line.