Project 1: Substrings and Dates

Members:
Karnikaa Velumani - karnikaavelumani@csu.fullerton.edu
Joel Anil John - joel.aniljohn@csu.fullerton.edu
Khang Tran - ktran1405@csu.fullerton.edu

Hypothesis: *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 (Reformat Date Algorithm):

```
valid():
    y = stoi(year);
    m = stoi(month);
    d = stoi(day);
    if y>=1900 and y<=2099 and m>=1 and m<=12 and d>=1 and d<=31):
        return true;
     return false;

reformat_date():
    month, year, day, result;
    start = 0, end = 0;
    for x from 0 through the length of x:
        if input[x] == ' ':
            start++;
            end++;
        else:
            break;
    while end<input.length():
        if input[end] == '-':
            if year == " ":
                year = portion of string indicating year;
                start = end+1;
                end++;
            else if month == " ":
                month = portion of string indicating month;
                start = end+1;
                end++;
                break;
            else
                day = portion of string indicating day;
                start = end+1;
                end++;
        else if input[end] == '/':
            Repeat same code as input[end] == '-'
        else if input[end] == ' ' or input[end] == ',':
            if input[end] == ' ':
                data = portion of string indicating month;
                for i from 0 to length of data:
```

```
                data[i] = make all characters lowercase
            month = convert[data];
            start = end+1;
            end++;
        else:
            day = portion of string indicating day;
            if end+2+4<=input.length():
                year = portion of string indicating year;
            else:
                print("Invalid input")
        break:
    else:
        end++;
if day == " ":
    while end<input.length and input[end] != ' ':
        end++;
    if end-start>2:
        print("Invalid input")
    day = portion of string indicating day;
    if day[1] == ' ':
        day = portion of string indicating day;
    else if year == " ":
        while end<input.length() and input[end]!=' ':
            end++;
    if end-start>4:
        print("Invalid input")
    year = portion of string indicating year;
if valid(year, month, day):
    if day.length()<2:
        day = "0" + month;
    result = year + "-" + month + "-" + day;
    else:
        print("Invalid input");
if result == " ":
    print("Invalid input")
return result;
```

Run-Length Encode
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

$5 + n(6 + 5 + 2) + 1 + 1$

$= 5 + 6n + 5n + 2n + 2$

$= 13n + 7$ → **chronological step count**

Prove by properties of O

$T(n) = 13n + 7 \epsilon O(13n + 7)$

$T(n) = 13n + 7 \epsilon O(13n, 7)$

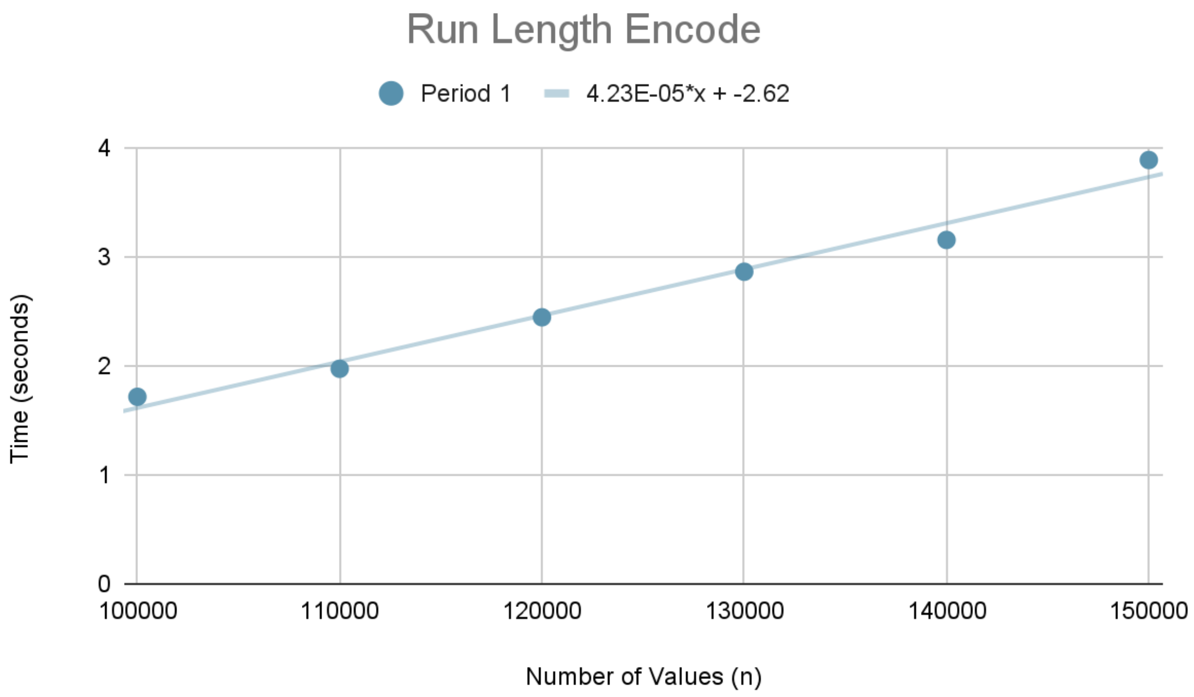
$T(n) = 13n + 7 \epsilon O(13n)$

$T(n) = 13n + 7 \epsilon O(n)$

$T(n) = O(n)$ → **efficiency class**

Scatter Plot

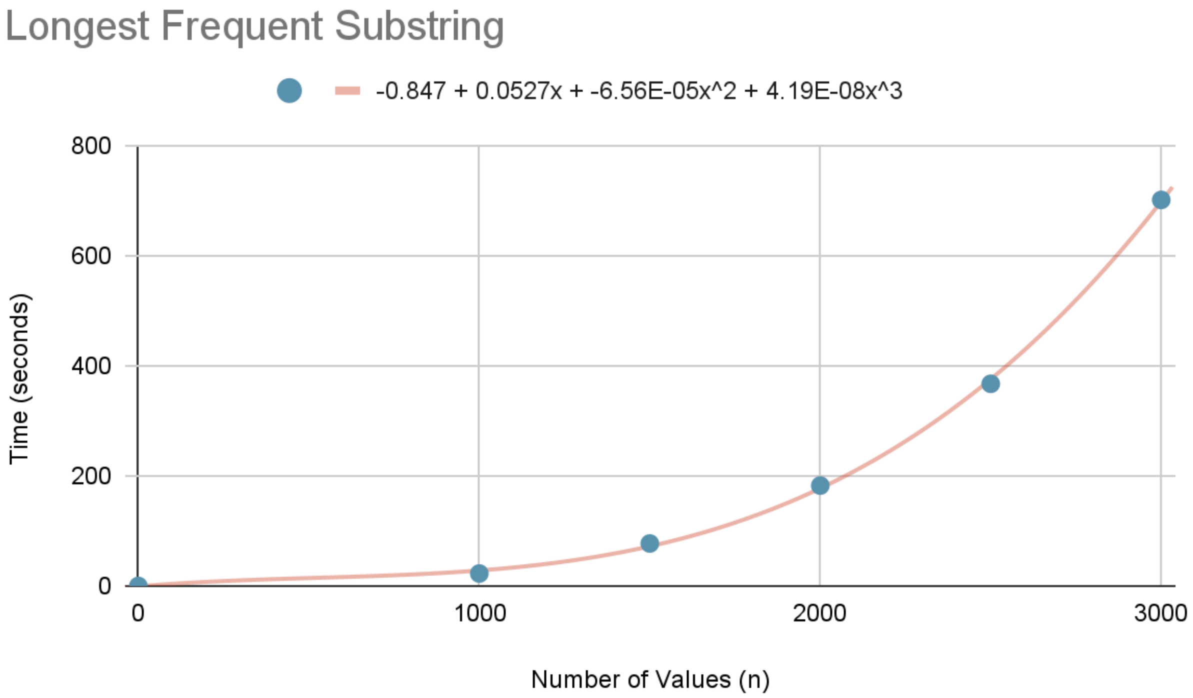
Run Length Encode	
Number of Values (*n*)	**Time (*seconds*)**
100000	1.715
110000	1.972
120000	2.444
130000	2.8615
140000	3.1529
150000	3.8856



Longest Frequent Substring
Input: a string *S* of *n* characters, and integer $k \geq 0$ **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*

<table>
<tr><th>Mathematical Analysis</th><th colspan="2">Scatter Plot</th></tr>
<tr><td rowspan="7">$1 + n(4) + 1 + n(n) + n(n + 4) + 1$
$= 1 + 4n + 1 + n^2 + n(n + 4) + 1$
$= 1 + 4n + 1 + n^3 + n^2 + 4n^2 + 4n + 1$
$= n^3 + 5n^2 + 8n + 3$ → <b>chronological step count</b>

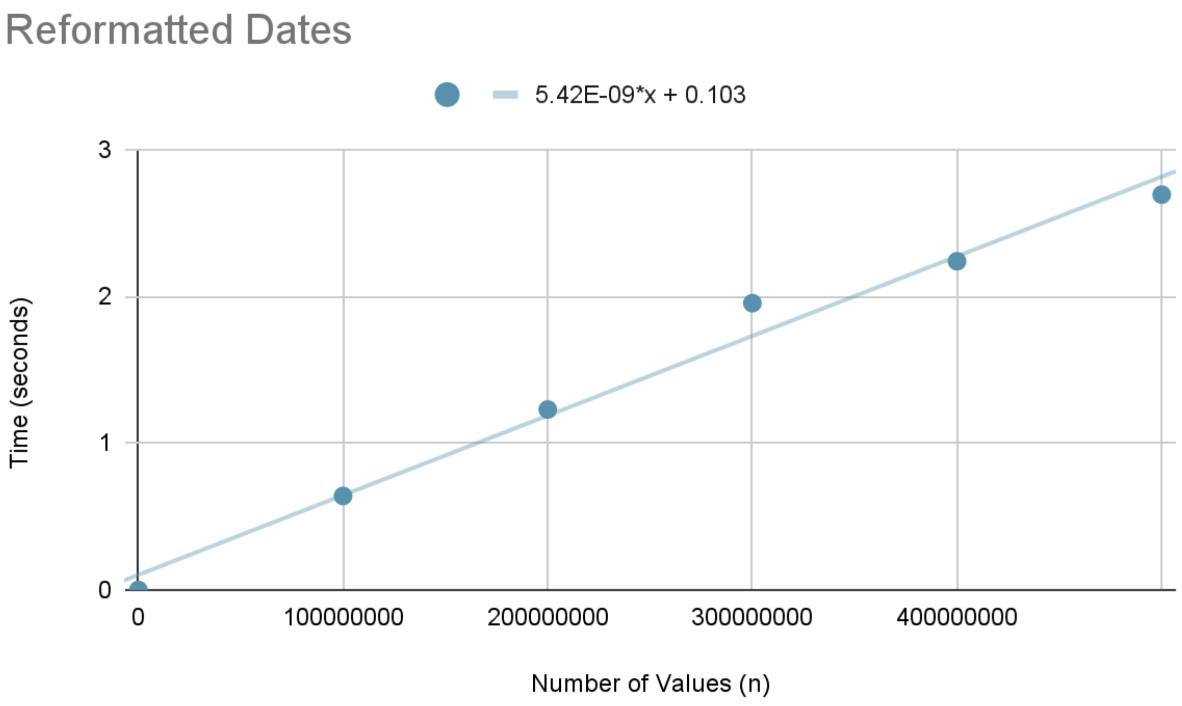
<i>Prove by properties of O</i>
$T(n) = n^3 + 5n^2 + 8n + 3 \in O(n^3 + 5n^2 + 8n + 3)$
$T(n) = n^3 + 5n^2 + 8n + 3 \in O(n^3, 5n^2, 8n, 3)$
$T(n) = 13n + 7 \in O(n^3)$
$T(n) = O(n^3)$ → <b>efficiency class</b></td><th colspan="2">Longest Frequent Substring</th></tr>
<tr><th>Number of Values (<i>n</i>)</th><th>Time (<i>seconds</i>)</th></tr>
<tr><td>1000</td><td>22.62</td></tr>
<tr><td>1500</td><td>77.12</td></tr>
<tr><td>2000</td><td>182.299</td></tr>
<tr><td>2500</td><td>367.059</td></tr>
<tr><td>3000</td><td>700.93</td></tr>
</table>



Reformat 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

<table>
<tr><th>Mathematical Analysis</th><th colspan="2">Scatter Plot</th></tr>
<tr><td rowspan="8">$4 + n(5) + n(15) + 11$
$= 4 + 5n + 15n + 11$
$= 25n + 15$ → <b>chronological step count</b>

<i>Prove by properties of O</i>
$\mathrm{T}(n) = 25n + 15 \in O(25n + 15)$
$\mathrm{T}(n) = 25n + 15 \in O(25n , 15)$
$\mathrm{T}(n) = 25n + 15 \in O(25n)$
$\mathrm{T}(n) = 25n + 15 \in O(n)$
$\mathrm{T}(n) = O(n)$ → <b>efficiency class</b></td><th colspan="2">Run Length Encode</th></tr>
<tr><th>Number of Values (<i>n</i>)</th><th>Time (<i>seconds</i>)</th></tr>
<tr><td>100000</td><td>1.715</td></tr>
<tr><td>110000</td><td>1.972</td></tr>
<tr><td>120000</td><td>2.444</td></tr>
<tr><td>130000</td><td>2.8615</td></tr>
<tr><td>140000</td><td>3.1529</td></tr>
<tr><td>150000</td><td>3.8856</td></tr>
</table>



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 run_length_encode algorithm produced a run time that was within the efficiency class of O(n) linear time. The longest_frequent_substring algorithm produced a run time of O(n^3) cubic time. The reformat_date algorithm produced a run time of O(n) linear time.

2. **Between the first two algorithms, is there a noticeable difference in the running speed? Which is faster, and by how much? Does this surprise you?**
    a. Between the first two algorithms, the first one is considerably faster. The first algorithm has a slope of $4.23*10^{-5}$ while the second algorithm started off with a slope of 0.109 and exponentially increased to a slope of 0.667742, meaning that it had a much sharper rate of change. This was not surprising because once the mathematical analysis was performed, it became clear that the function would produce a run time of O(n^3). Additionally, by looking at the nested for-loops inside of the pseudocode, we were already able to deduce that the run-time would be around cubic time.

3. **Are the fit lines on your scatter plots consistent with the efficiency classes predicted by your math analyses? Justify your answer.**
    a. Yes, the first and third graphs produced a best fit line that was linear, which is consistent with the O(n) efficiency class we had mathematically established for those two algorithms. Similarly, the second graph produced a best fit line that was cubic, which was consistent with the O(n^3) efficiency class we had mathematically established for that algorithm.

4. **Is all this evidence consistent or inconsistent with the hypothesis stated on the first page. Justify your answer.**
    a. Yes, all the evidence we have gathered so far is very much consistent with the hypothesis stated in the first page. Our mathematical analysis predicted an O(n) run time for the first and third algorithms, which was then confirmed by the linear line of best fit that is produced in the scatter plots for both of those algorithms. The mathematical analysis predicted an O(n^3) run time for the third algorithm, which was also confirmed by the cubic line of best fit that the second scatter plot was able to produce.