CPSC 335 Project 1

Jason Angel
jasonangel@csu.fullerton.edu

Pseudocode for Reformat Date Algorithm

```
reformat_date(S):
        test_date_vars = empty list of strings # will contain string variables parsed from S to be validated (year, month, and date)
       valid_date_vars = empty list of strings # will contain valid content that will be concatenated to a single string in the end (year, month, and
date)
        pos_map = empty list of ints # will contain mapping of index positions for  test_date_vars. Depends on the case of date format.
                                        # for pos_map, index positions 0,1,and 2 respond to year, month, and day respectively.

        WHITESPACE = " \r\n\t\v\f"
       start = S.find_first_of_any_chars(WHITESPACE)
       end = S.find_last_of_any_chars(WHITESPACE)

       if (start == end):
                A = ""
       else:
                A = S.substr(start, end - start + 1)

        # Split string into variables to be tested
       if A.find('-'):
                parseString(test_date_vars, A, '-')
       elif A.find('/'):
                parseString(test_date_vars, A, '/')
       elif A.find(' '):
                A.remove(',') # remove all occurances
               parseString(test_date_vars, A, ' ')
       else:
                throw exception

       if test_date_vars.size() != 3:
                throw exception

       if test_date_vars[0].empty() or test_date_vars[1].empty() or test_date_vars[2].empty():
                throw exception

         if first character of test_date_vars[0] is alpha:  # we are likely in "MONTH D, Y" or "MON D, Y" format:
                pos_map[0] = 2, pos_map[1] = 0, pos_map[2] = 1
               dateValidation(valid_date_vars, test_date_vars, pos_map)
        elif first character of test_date_vars[0] is int:
                if test_date_vars[0].size() == 1 or test_date_vars[0].size() == 2:  # we are likely in "M/D/Y" format
                        pos_map[0] = 2, pos_map[1] = 0, pos_map[2] = 1
                        dateValidation(valid_date_vars, test_date_vars, pos_map)
                elif test_date_vars[0].size() == 4:  # we are likely in "Y-M-D" format
                         pos_map[0] = 0, pos_map[1] = 1, pos_map[2] = 2
                        dateValidation(valid_date_vars, test_date_vars, pos_map)
                else:
                         throw exception
       else:
                throw exception

       # valid date_vars should now have Y,M, D in the respective order and should be valid

       if valid_date_vars[1].size() == 1:
                valid_date_vars[1].push_front('0')
       if valid_date_vars[2].size() == 1:
                valid_date_vars[2].push_front('0')

       date = valid_date_vars[0] + '-' + valid_date_vars[1] + '-' + valid_date_vars[2]

       return date
```

```
parseString(test_date_vars, str, delim):
        field = ""

        for c in str:
                if c == delim or c == str.size() - 1:
                        if c == str.size() - 1:
                                field = field + c
                        test_date_vars.push_back(field)
                        field = ""
                else:
                        field = field + c

convertStringToLower(str):
        for c in str:
                c = tolower(c)

fieldIntValidation(str):
        for c in str:
                if !isdigit(c):
                        throw exception

dateValidation(valid_date_vars, test_date_vars, pos_map):
        # Note pos_map index values [0],[1],[2] correspond to year, month, and day respectively and their index positions in test_date_vars

        fieldIntValidation((test_date_vars[pos_map[0]])
        int y_int = convert_to_int(test_date_vars[pos_map[0]])

        if y_int < 1900 or y_int > 2099:
                throw exception
        else:
                valid_date_vars[0] = test_date_vars[pos_map[0]]

        if first character of test_date_vars[pos_map[1]] is alpha:
                months = vector of strings consisting of all 12 months (full names)

                convertStringToLower(test_date_vars[pos_map[1]])

                month_num = 1
                for month in months:
                        convertStringToLower(month)

                        if !month.find(test_date_vars[pos_map[1]])
                                if test_date_vars[pos_map[1]].size() == 3 or month.size() == test_date_vars[pos_map[1]].size():
                                        valid_date_vars[1] = convertToString(month)
                                        break

                if valid_date_vars[1].empty():
                        throw exception
        else:
                fieldIntValidation((test_date_vars[pos_map[1]])
                m_int = convert_to_int(test_date_vars[pos_map[1]])

                if m_int < 1 > m_int > 12:
                        throw exception
                else:
                        valid_date_vars[1] = test_date_vars[pos_map[1]]

        fieldIntValidation((test_date_vars[pos_map[2]])
        int d_int = convert_to_int(test_date_vars[pos_map[2]])

        if d_int < 1 or d_int > 31:
                throw exception
        else:
                valid_date_vars[2] = test_date_vars[pos_map[2]]
```

Mathematical Analysis

Algorithm 1

Note:

append_run:

O(1)

Performing chronological step counting on run_length_encode gives:

= 1+1+1+1+1+1(n-1)+1(n-1)+1(n-1)+1(n-1)+1(n-1)+1(n-1)+1(n-1)+1+1

= 7n

Proving efficiency class by limits:

lim T(n) / f(n)

n -> ∞

lim 7n / n = 7

n -> ∞

Therefore $7n \in O(n)$

Algorithm 2

Performing chronological step counting on longest_frequent_substring gives:

= 1 + 1(n) + 1(n) + 1(n) + 1 + 1(n-1) + 1(n^2) + n(n^2) + n(n^2 + 1) + 1(n^2) + 1(n^2) + 1

= 2n^3 + 3n^2 + 5n + 2

Proving efficiency class by limits:

lim 2n^3 + 3n^2 + 5n + 2 / n^3 = 2

n -> ∞

Therefore $2n^3 + 3n^2 + 5n + 2 \in O(n^3)$

Algorithm 3

Note:

parseString:

O(n)

convertToString:

O(n)

date_int_validation:

O(n)

date_validation:

O(n)

Performing chronological step counting on reformat_date gives:

= 1 + 1 + 1 + 1 + n + n + 1 + 1 + 1 + 1(n) + 1(n) + 1(n) + 1 + n + 1 + 1 + 1

+ 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + n + 1 + 1 + 1 + 1

= 7n + 24

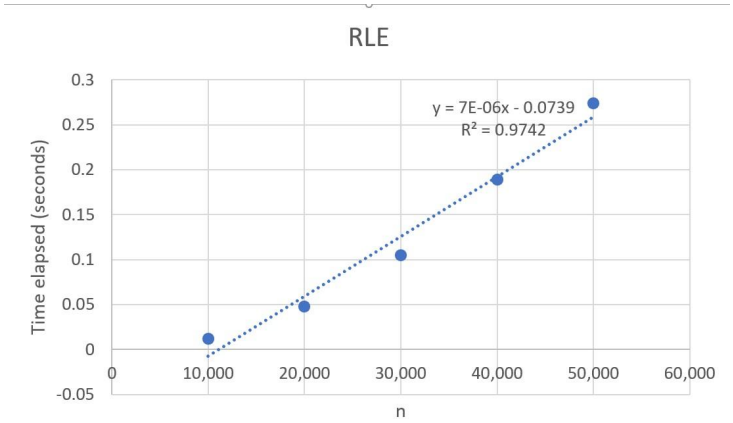
Proving efficiency class by limits:

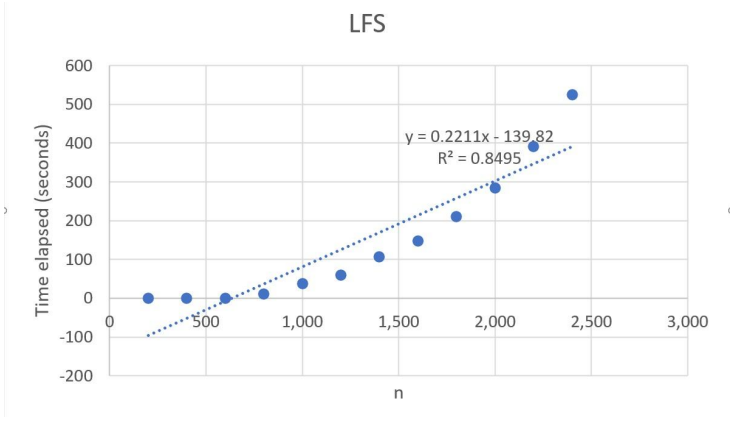
lim 7n + 24 / n = 7

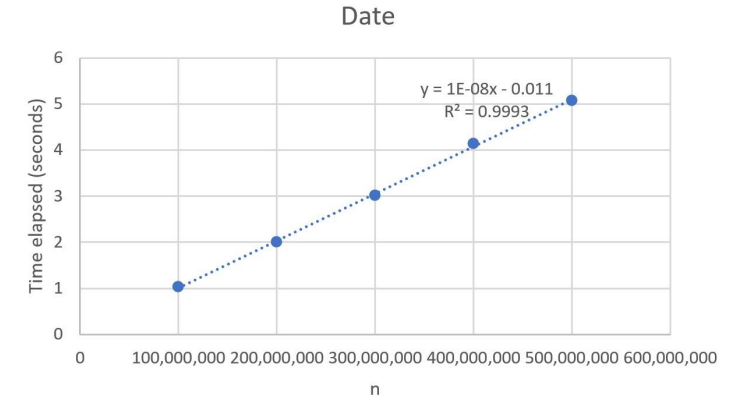
n -> ∞

Therefore $7n + 24 \in O(n)$

Scatter Plots







Questions

1. The efficiency class for RLE, LFS, and the date algorithm is O(n), O(n^3), and O(n) respectively.
2. Yes there is a noticeable difference. The RLE algorithm, which has an efficiency class of O(n) is much faster than LFS, which has an efficiency class of O(n^3). This is not surprising because LFS involves nested for loops whereas RLE has a single for loop at most.
3. Yes the fit lines are consistent with the efficiency classes because each line has a high r^2 value.
4. Yes, all the evidence is consistent with the hypothesis. The data gathered for each algorithm matches their mathematically derived efficiency class. Time increases as input increases and is also dependent on the efficiency class .