Project 1: Mathematical and Empirical Analysis

Roberto Guerra robert5guerra@csu.fullerton.edu

Jonathan Aguirre jonaguirre@csu.fullerton.edu

Arifulla shaik arifullashaik@csu.fullerton.edu

1. Pseudocode describing your reformat date algorithm.

```
define global monthArr and initialize with full name of Months
define global monArr and initialize with first three letters of months

// validate range helper function
bool validateRange(int year, int, month, int day):
        check if yer between 1900 and 2099  else throw exception
        check if month between 1 and 12 else throw exception
        check if day between 1 and 31 else throw exception
  return true

// is_digits function to verify if numbers
bool is_digits(string& str):
        return true if all are digits else return false

// check pattern function
int checkPattern(input):
        if not is_digits(input) : throw exception
        check if input contains “ - “, if so return 1
        check if input contains “ / “ , if so return 2
        check if input month is full month name, if so return 3
        check if input month is first 3 characters of month, if so return 4
  throw exception otherwise

def reformat_date(string input) :
        pattern = checkPatter(input)   (valid patterns:   1, 2, 3, 4)
        check if pattern is valid : otherwise throw invalid_argument
        // needed variables
        output set to empty string
        year set to 0
        month  set to 0
        strMonth set to empty string
        strDay set to empty string
        If (pattern == 1):
            make temp index
            assign values to year, month, and day
            Increment temp index to account for ‘-’
            if (validateRange(year, month, day):
                check that year, month, day are within bounds set
                concatenate to output variable and return
        If (pattern == 2):
            make temp index
            assign values to year, month, and day
            Increment temp index to account for ‘/’
            if (validateRange(year, month, day):
                check that year, month, day are within bounds set
            concatenate to output variable and return
```

```
if (patter == 3) :
    make a temp index
    assign month to inputMonth variable
    increment  temp index
    for loop to check if inputMonth is in monthArr
    assign value to day
    increment temp index based on comma
    assign value to year
    if (validateRange(year, month, day):
        check that year, month, day are within bounds set
if (pattern == 4):
    make a temp index
    assign month to inputMonth variable
    increment  temp index
    for loop to check if inputMonth is in monArr
    assign value to day
    increment temp index based on a space character
    assign value to year
    if (validateRange(year, month, day):
        check that year, month, day are within bounds set
    concatenate to output variable and return
```

2. Mathematical analyses for each of the three algorithms.

```
run_length_encode(S):
          C = ""                1
   if S is empty: 1
          return C              1
          run_char = S[0] 1
          run_length = 1    1
          for each character c in S after the first: n
          if c == run_char:                   n
          run_length++                n
          else:
          append_run(C, run_char, run_length)n
          run_char = c                        n
          run_length = 1                      n
          append_run(C, run_char, run_length)                   1
          return C                            1

append_run(C, run_char, run_length):                  1
          if run_length > 1:                          1
      C.append(convert_to_string(run_length))         1
          C.append(run_char)                         1
```

$$T(n) = 5n + 11$$

By properties of O

$$5n + 11 \in O(5n + 11) \qquad (trivial)$$

$$= O(\max(5n, 11)) \qquad (Dominated\ terms)$$

$$= O(5n)$$

$$= O(n) \qquad\qquad (Constant\ factor)$$

```
longest_frequent_substring(S, k):
    freq = new dictionary that maps a char to an int          1
    for c in S:                    n
        if c not in freq:      n
            freq[c] = 1        n
        else:
            freq[c]++          n
    best = ""          1
    for b from 0 through n-1:  n
        for e from b+1 through n:     n*n
            cand = S[b:e]             n*n*n
            if every character c in cand has freq[c] >= k:   n*n*n
                if cand.size() > best.size():                n*n*n
                    best = cand               n*n*n
    return best        1
```

$T(n) = 4n^3 + n^2 + 5n + 3$

By properties of O

$4n^3 + n^2 + 5n + 3 \in O(4n^3 + n^2 + 5n + 3) \qquad (trivial)$

$= O(\max(4n^3, 5n^2, 5n, 3)) \qquad (Dominated\ terms)$

$= O(4n^3)$

$= O(n^3) \qquad\qquad (Constant\ factor)$

Reformat Date Algorithm

$$T(n) = 2n^2 + 9n + 8$$

By properties of O

$$2n^2 + 9n + 8 \in O(2n^2 + 9n + 8) \qquad (trivial)$$

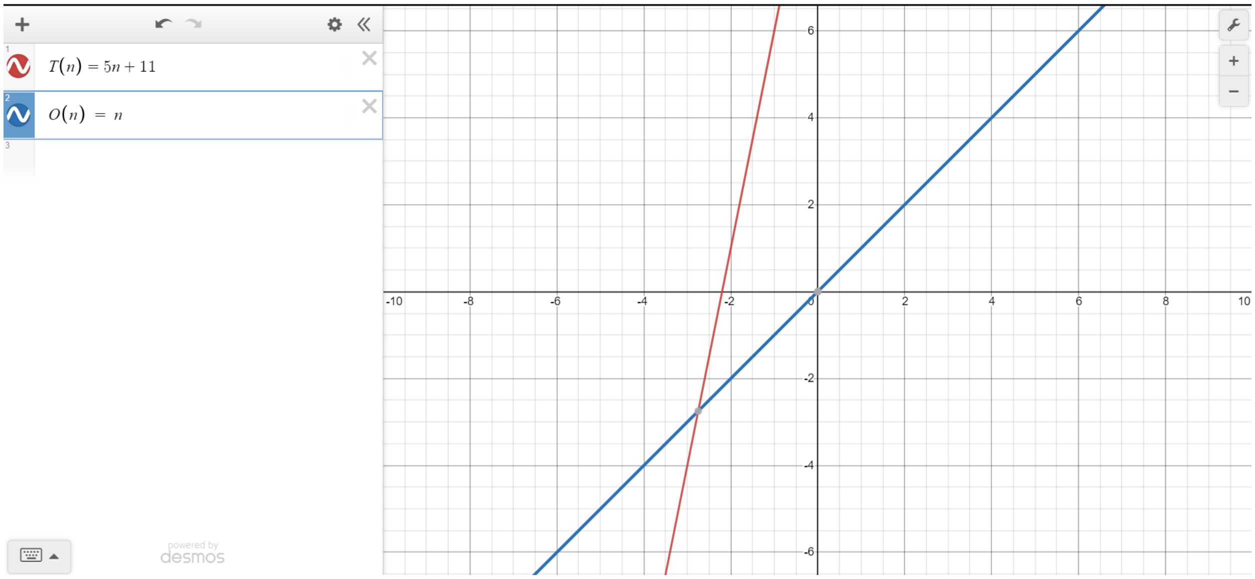
$$= O(\max(2n^2, 9n, 8)) \qquad (Dominated\ terms)$$

$$= O(2n^2)$$

$$= O(n^2) \qquad\qquad (Constant\ factor)$$

3. Scatter plots for each of the three algorithms.

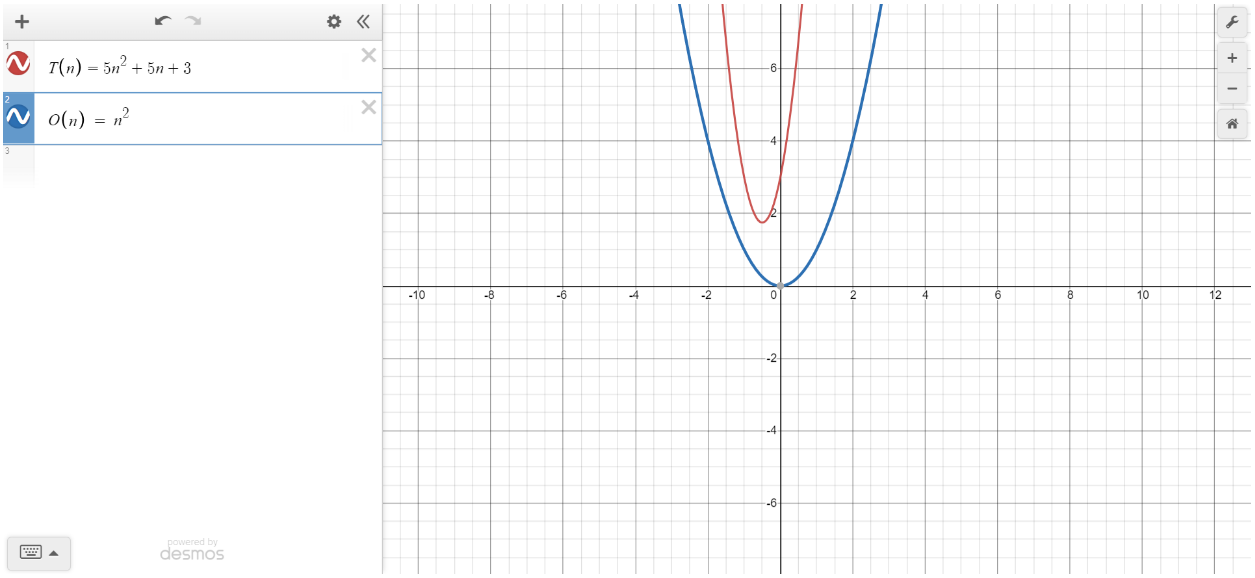
Algorithm 1



1 $T(n) = 5n + 11$

2 $O(n) = n$

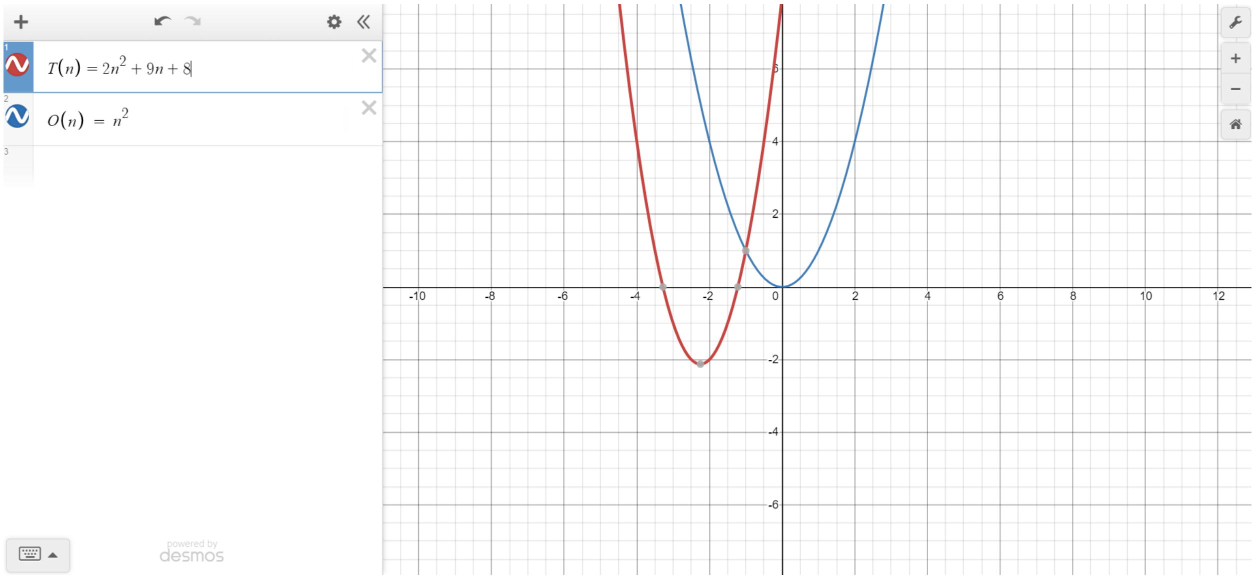
Algorithm 2



1 $T(n) = 5n^2 + 5n + 3$

2 $O(n) = n^2$

Algorithm 3



1 $T(n) = 2n^2 + 9n + 8$

2 $O(n) = n^2$

4. Answers to the following questions. (Each answer should be at least one complete sentence.)
    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.)

The efficiency class for each of the algorithms according to our mathematical analysis and by properties of O came out to be for run-length encode $O(n)$, longest frequent substring as $O(n^2)$, and for the third algorithm $O(n^2)$.

    2. Between the run-length encode and longest frequent substring algorithms, is there a noticeable difference in the running speed? Which is faster, and by how much? Does this surprise you?

After running the make test command a few times it looks like the Longest frequent substring is much faster. Last run for run-length encode was 1098ms vs 31ms for Lfs. This is surprising since Lfs has a nested for loop.

    3. Are the fit lines on your scatter plots consistent with the efficiency classes predicted by your math analyses? Justify your answer.

The fit lines on the scatter plot are consistent with efficiency classes that were predicted by the math analysis because the shape of T(n) and Big O are generally the same shape. However, the graphs generated from T(n) are usually displaced on the x-axis or y-axis and can sometimes be narrow or wider compared to the graph generated from the Big O.

4. Is all this evidence consistent or inconsistent with the hypothesis stated on the first page? Justify your answer.

From the looks of all the evidence, for large values of n the derived efficiency classes were accurately predicted for the observed running time of the implementation.