California State University, Fullerton
CPSC 335 Project 1 Report
04/06/2021

Submitted by:

Janelle Estabillo estabillojanelle@csu.fullerton.edu
Nour Alquraini nouralquraini@csu.fullerton.edu
Triet Le tle877@csu.fullerton.edu

Submitted to:
Kevin Wortman

I. Telegraph Algorithm: Pseudocode

```
        //pseudocode starts here
telegraph(string S)
        Bool flag = false;
        String return_s;
        Int size = s.size()
        if S is empty
                Add “STOP.” to return_s
                return return_s
        for (int i = 0; i < return_s.size(); i++)
                If s[i] is ‘!’ or ‘?’ or ‘;’
                        Return_s += ‘.’
                        flag = false
                Elseif s[i] is an alphabet
                        return_s+= toupper(s[i]);
                        flag = false
                Else if s[i] a digit or ‘.’
                        return_s += s[i];
                        flag = false;
                Else if s[i] = space
                        if flag == false
                                return_s+= s[i];
                                flag = true;
        check the last word if it equals to “STOP.”
                if not
                         add “STOP.” at the end
       Return return_s
```

II. Mathematical Analysis

A. Dip Search Problem

Pseudocode and Analysis using Properties of O

```
Initialize last_dip=-99  O(1)

If values < 3 O(1)

      Return values

Else O(1)

      for (int i = 0; i <= values.size() - 3; i++) O(1) {// n-2 iterator

             If (if first element is equal to the third element and 2nd
             element is less than the first element) O(1)

                    Last_dip = i O(1)

vector<int>::const_iterator result = values.begin(); O(1)

If last dip is greater or equal to 0 O(1)   //worst case

       Print last_dip O(1)

       Advance the iterator (result, last_dip) O(1)

       Return result; O(1)

Else

       Print last_dip

       Return values.end()
```

Analysis

Proof. By properties of O

$3 + 3*(n-2) + 5$	$= 3n - 1$	
	$\in O(3n - 6)$	(trivial)
	$= O(3n)$	(dominated term)
	$= O(n)$	(constant factor)

Therefore, by the properties of O, the time complexity is O(n) which is linear.

B. Longest Balanced Span Problem
Pseudocode and Analysis using Properties of O

```
If values are empty O(1)
        Return nullopt
Initialize begin, end, best, sum, and quantity O(1)
For (int start = 0; start < values.size(); start++) O(1)   // n iterator
        Sum = values.at(start);     O(1)
        If element is 0 O(1)
                Set quantity = 0 O(1)
                If (best is < 0 or quantity is >= best O(1)
                        Set best = quantity O(1)
                        Begin = start O(1)
                        End = start + 1 O(1)
        For (loop through the rest elements) O(1)  // n-1 iterator
                Sum = add values up      O(1)
                If sum is 0              O(1)
                 quantity = finish - start//get length of found span  O(1)
                 If best == 0 or quantity >= best     O(1)
                        Set best = quantity O(1)
                        Begin = start O(1)
                        End = start + 1 O(1)

vector<int>::const_iterator begin_ = values.begin(); O(1)
vector<int>::const_iterator end_ = values.begin(); O(1)
advance(begin_, begin); O(1)
advance(end_, end); //move iterator to the end O(1)
If best is greater than or equal to 0 O(1)     //worst case
        span result = span(begin_, end_); O(1)
        return result; O(1)
Else
        Return null
```

Analysis

Proof. By Properties of O.

T(n) = 2+ n (8+ (n-1)*8) + 7 = 9 + n (8 + 8n - 8)

= 9 + 8n^2

∈ O (9 + 8n^2) (trivial)

= O(8n^2) (dominated)

=O(n^2) (constant)

Therefore, by the properties of O, the time complexity is O(n^2) which is quadratic.

C. Telegraph Style String Problem
Pseudocode and Analysis using Properties of O

```
//pseudocode starts here
telegraph(string S)
    Bool flag = false;  O(1)
    String return_s;    O(1)
    Int size = s.size() O(1)
    if S is empty       O(1)
        Add "STOP." to return_s
        return return_s
    for (int i = 0; i < return_s.size(); i++)  O(1)     // n iteration.
        If s[i] is '!' or '?' or ';' O(1)
            return_s+= '.'
            flag = false
        Elseif s[i] is an alphabet O(1)
            return_s+= toupper(s[i]);
            flag = false
        Else if s[i] a digit or '.' O(1)
```

```
                return_s += s[i];
                flag = false;
            Else if s[i] = space O(1)          //Space is  the worst-case
                if flag == false  O(1)
                    return_s+= s[i];  O(1)
                    flag = true;  O(1)
        check the last word if it equals to "STOP."    O(1)
            If not then add "STOP." at the end      O(1)
        Return return_s O(1)
```

Analysis

Proof. By Properties of O

$T(n) = 4 + 8n + 3 \quad = O(8n + 7)$ (trivial)

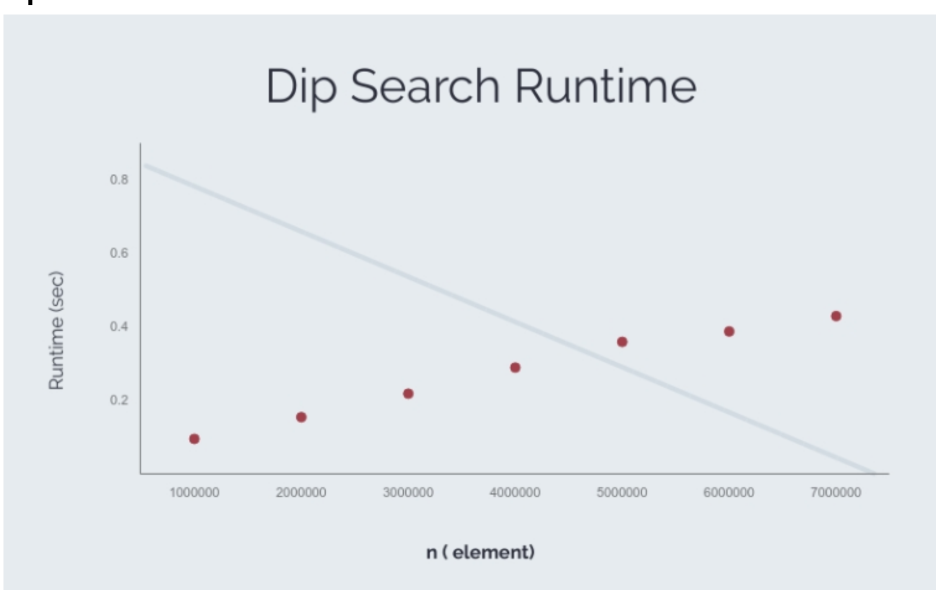
$= O(8n)$ (dominated)

$= O(n)$ (constant)

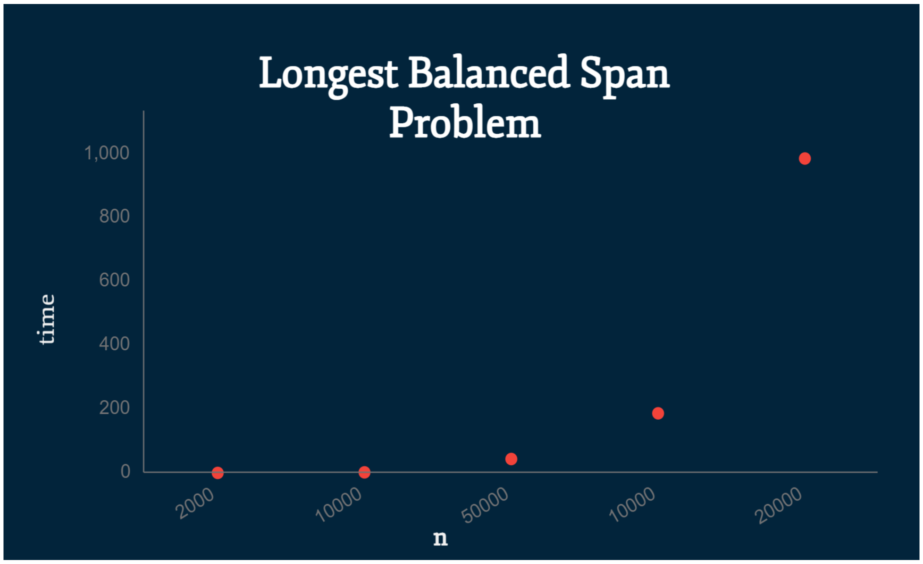
Therefore, by the properties of O, the time complexity is O(n) which is linear.

III. Scatter Plots

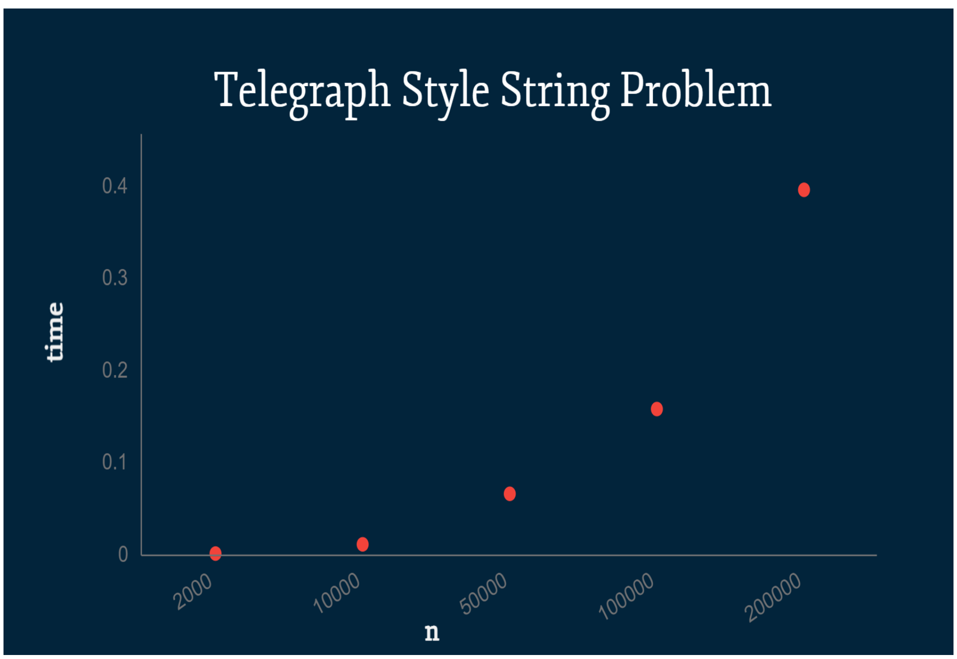
A. Dip Search Problem



B. Longest Balanced Span Problem



C. Telegraph Style String Problem



IV. Additional Questions

A. **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.)**

   1. **Dip Search Problem**
      - ➔ The efficiency class for the Dip Search Problem using the Properties of O is O(n) which is Linear.
   2. **Longest Balanced Span Problem**
      - ➔ The efficiency class for the Longest Balanced Span Problem using the Properties of O(n^2) which is quadratic.
   3. **Telegraph Style String Problem**
      - ➔ The efficiency class for the Telegraph Style String Problem using the Properties of O is O(n) which is Linear.

B. **Between the dip search and longest balanced span algorithms, is there a noticeable difference in the running speed? Which is faster, and by how much? Does this surprise you?**
   - ➔ There was a very noticeable difference between the two. The longest balance span is noticeably slower than the dip search. This doesn't surprise me as per our empirical analysis and the mathematical analysis data, the evidence points out that the longest balance span will run noticeably slower. Therefore, dip search is considerably faster than the longest balance span algorithm.

C. **Are the fit lines on your scatter plots consistent with the efficiency classes predicted by your math analysis? Justify your answer.**
   - ➔ For the dip search problem, the scatter plot shows a linear line which is consistent with the efficiency classes that were predicted by our mathematical analysis which was O(n) time complexity.
   - ➔ For the longest balanced span problem, the scatter plot is consistent with the efficiency class since we got O(n^2) from the mathematical analysis which can be seen as evidence on the scatter plot.
   - ➔ For the telegraph style string problem, the scatter plot shows a linear line which is consistent with the efficiency classes that were predicted by our mathematical analysis which was O(n) time complexity.

D. **Is all this evidence consistent or inconsistent with the hypothesis stated on the first page? Justify your answer.**
   - ➔ The hypothesis stated that "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." Three algorithms have been analyzed empirically using at least 5 data points and mathematically using the Properties of O. Therefore, it has been concluded that all evidence is consistent with the stated hypothesis.