Project 1: Mathematical Analysis

Algorithm 1: Character Mode

char character\_mode(const string\_vector& strings) {

int charArray[256] = { 0 }; // O(1) constant time

string word; // O(1)

char character; // O(1)

int num\_words = strings.size(); // O(1)

int num\_chars = 0; // O(1)

int index\_biggest = 0; // O(1)

for (int i = 0; i < num\_words; i++) //n

{

word = strings[i]; // O(1)

num\_chars = strings[i].size(); // O(1)

for (int j = 0; j < num\_chars; j++) //45 worst case

{

character = word[j]; // O(1)

charArray[int(character)]++; // O(1)

}

}

for (int i = 0; i < 256; i++) //256 constant time

{

if (charArray[index\_biggest] < charArray[i]) // O(1)

index\_biggest = i; // O(1)

}

return char(index\_biggest); // O(1)

}

lemma: 7 + n(2 + 45(2)) + 256(2)

∴

Algorithm 2: Longest Mirrored String

def longest\_mirrored\_string(V):

best = “” //O(1)

for a in V: //O(n)

for b in V: //O(n - 1)

if a is the mirror of b and len(a) > len(best):

best = a //45 worst case

return best //O(1)

lemma: 2 + n(n(45))

∴

Algorithm 3: Longest Substring Trio

def longest\_subset\_trio(V):

best\_length = 0 //O(1)

trio = Vector(3, “”) //O(3)

for a in V: //O(n)

for b in V: //O(n)

for c in V: //O(n)

abc\_length = |a|+|b|+|c| //O(4)

if <a, b, c are a subset trio> and

  abc\_length > best\_length:

best\_length = abc\_length //Constant time

trio[0] = a //O(1)

trio[1] = b //O(1)

trio[2] = c //O(1)

return trio //O(1)

lemma: 5 + n(n(n(Constant C + 3 + 4)))

∴

**Project Questions and Answers:**

**◦Is there a noticeable difference in the performance of the three algorithms? Which is faster, and by how much? Does this surprise you?**

Absolutely, the differences were significant between the performance of the various algorithms. The fastest one was the character mode function, the second fastest was the longest mirror, and the slowest was the longest subset trio algorithm. It was very surprising to see how long it would take for the runtime of the longest subset trio algorithm, even after already knowing that its time complexity was higher.

**◦What is the efficiency class of each of your 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 classes:

character\_mode – o(n)

longest\_mirrored\_string - O(n2)

longest\_substring\_trio – O(n3)

**◦Are the fit lines on your scatter plots consistent with these efficiency classes? Justify your answer.**

Yes they are because when you analysis the charts, specically, the character mode chart the scatterplot line is very linear in behavior. While when we analyze the scatter plot data of the longest mirrored string we see that the line behaves more exponentially, and even more exponentially when we analyze the longest substring trio chart. In the end, the scatter plot data definitely did match our findings of the various efficiency classes.

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

Our findings were definitely consistent with the hypothesis previously stated because when we ran the program executing the character count, from small vectors to very large vectors the program ran very quicky. Which was consistent with the efficiency class of O(n). There

was even more confluence when we analyzed the charts for the character mode algorithm, seeing that it acted linear. However, the big O runtime of the longest mirrored string was O(n2), and it took quite a bit longer to run than the previous algorithm. The last algorithm, longest subset trio was what opened our eyes to really understanding how important the efficiency classes are because the runtime would have even taken up to aleast 45 hours we calculated for it to run if the whole input file was used. Which was very interesting to comprehend considering how much quicker the two previous algorithms ran. In conclusion, our evidence and analysis was consistent with the originally given hypothesis.

**Empirical Analysis Data**

|  |  |
| --- | --- |
| Character Mode | |
| N | Time |
| 10000 | 0.011998 |
| 20000 | 0.0347496 |
| 30000 | 0.058334 |
| 40000 | 0.0736592 |
| 50000 | 0.0950064 |
| 60000 | 0.122432 |
| 70000 | 0.130214 |
| 80000 | 0.146285 |
| 90000 | 0.170978 |
| 99171 | 0.197834 |

|  |  |
| --- | --- |
| Longest Mirror | |
| N | Time |
| 1000 | 0.3163 |
| 3000 | 2.82366 |
| 5000 | 7.695 |
| 7000 | 14.9801 |
| 9000 | 24.6612 |
| 11000 | 39.2653 |
| 13000 | 51.39 |
| 15000 | 69.1609 |
| 17000 | 89.2903 |
| 19000 | 109.774 |

|  |  |
| --- | --- |
| Substring Trio | |
| N | Time |
| 50 | 0.221378 |
| 100 | 1.93244 |
| 150 | 6.4358 |
| 200 | 15.825 |
| 250 | 29.8595 |
| 300 | 56.6512 |
| 350 | 84.6026 |
| 400 | 127.08 |
| 450 | 176.518 |
| 500 | 251.94 |

Algorithm 1: Character Mode

Algorithm 2: Longest Mirrored String

Algorithm 3: Longest Substring Trio