Mehmet Barış Yaman
150130136

ANALYSIS OF ALGORITHMS 2
PROJECT 2 REPORT

Mehmet Barış Yaman
150130136

1) If the advantages of divide and conquer methodology were not to be used in your implementation, how would you formulate the problem?

If that methodology were not to be used, then iterative process should be used instead of recursion. Then I would use a big for loop using the argument counter that access node vector's elements each time. When a transformation is made correcty then the counter's value changes and for loop gets the next iteration. When there is no transformation rule, the for loop also gets the next iteration, but after a correct transformation is done it will come back to rightmost index, since the counters value should always hold the last element of the vector in order to do the operations with rightmost tag set pairs. Additionally when S is obtained and no other tag is left this function will return correctly, since all loop process in a function will terminate with “return” statement in C++.

2) What parameters does the complexity of your implementation depend on? How would you represent the worst case complexity in big O notation?

The complexity of my algorithm depends on the number of words in the sentence and the transformations with rightmost pairs. If the number of words increases the number of recursions will increase, since the input size gets bigger. Additionally, if there is no rule for the transformations of rightmost pairs, the counters value shiftes left and checks again whether there is a rule or not. The complexity of the algorithm increases with that, since more recursions come. In the worst case scenario, there is always a transformation rule with the leftmost pairs and the function always goes leftmost index before any transformation. With that process the algorithms complexity will be O(n^2), since all elements are accessed in both coming leftmost indexes and the transformations.

3) If the rule VP → NP VP were to be added to current set of grammar rules, this would cause ambiguity because it would overlap with the rule S → NP VP. Briefly explain, how you would change your implementation to overcome this problem. Would it change the worst case complexity? If so, what would the complexity be in big O notation?

With that addition of grammer rule, the algorithm will have 3 options to overcome to the problem of when the transformation operands are NP and VP sequentially:

1) The algorithm will select S → NP VP
2) The algorithm will select VP → NP VP
3) The algorithm will shift left and looks the next operands

Therefore, the algorithm should look at first the option 1 and print out the result if there is no other elements left. If there is an element left, the algorithm gets option 2 and checks whether S is accessed or not. If not the algorithm traces back to the point where it has 3 options and chooses option 3 and tries to access S again.

Considering the options the worst case will be accessing S with option 3. Since option 1 and option 2 will be tried, the complexity will increase. Because all elements are accessed in the two options, the complexity will increase O(n^3).