Time complexity analysis:

1. CYK.java class

```java
/**
 * Class which builds a CYK table given a grammar to check whether the String belongs to that grammar
 * @author anush
 */
public class CYK {

        /**
         * A Jagged 2D array which will be used to implement the CYK algorithm dynamically
         * Each cell in the array will store a list of lhs of the rules of this grammar
         */
        ArrayList<String>[][] cykTable;

        /**
         * The string that we want to check can be formed through this grammar
         */
        String checkString;

        /**
         * Parser object used to parse and implement cnf rules
         */
        CNFParse cnf;

        /**
         * The constructor to implement this CYK
         * @param str The string to be checked
         * @param rules The rules of the grammar (in CNF)
         */
        @SuppressWarnings("unchecked")
        public CYK(String str, ArrayList<String> rules) {
                this.checkString = str;
                cnf = new CNFParse(rules);

                //Number of rows = length of the String
                this.cykTable = new ArrayList[str.length()][];
```

lhs → rhs

This method will take input string and list of rules.

call CNFParse class methods. It takes O(n^2) running time.

if input string is length n, then
cyk table will have n number of rows.

```
        //Filling in number of columns for each row
        for(int i = 0; i < str.length(); i++) {
                cykTable[i] = new ArrayList[str.length()-i];
        }

        this.fillTable();
}

/**
 * Method that fills the CYK Table
 */
public void fillTable() {

        //i = Row variable
        for(int i = 0; i < checkString.length(); i++) {
                //j = Column variable
                for(int j = 0; j < checkString.length() - i; j++) {
                        cykTable[i][j] =  fillCell(j, i);
                }
        }
}

/**
 * Returns a list of Strings that should fill a specific cell of the array
 * @param col The column number of the cell
 * @param row The row number of the cell
 * @return A list of Strings to fill this cell
 */
public ArrayList<String> fillCell(int col, int row) {
        ArrayList<String> rules = new ArrayList<String>();

        if(row == 0) {
                /*
                 * First row
                 * Fill the cell with the rule used to obtain the literal corresponding to this column
```

// decrement each row's colum by 1

Total This function will run O(n^3). This is expensive compare to other functions.

If input string length is n,
then outer for loop will run n times.
Inner for loop will run n-1 times.
therefore, fillTable() will have O(n^2) running time complexity.

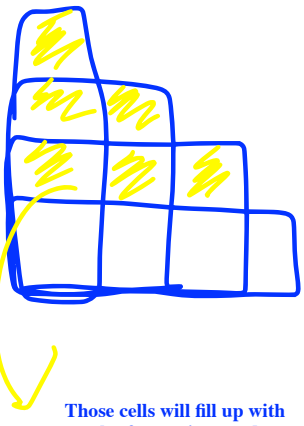
```
         * Get the character of this column => index is column number - 1
         */

        //The literal is in the same position as the column number
        //Indexing starts from 0 so the position is actually column number - 1
        String c = this.checkString.charAt(col) + "";

        //Get the list of rules that can be used to obtain the literal
        rules = cnf.belongsToRules(c);
        return rules;
}

else {

        //List of Strings returned after doing Cartesian Product of 2 cells
        ArrayList<String> prodList = new ArrayList<String>();

        //List of Strings that are can be obtained in a cell
        ArrayList<String> strList = new ArrayList<String>();

        for(int i = 0; i < row; i++) {
                //Get the Cartesian Product of the cells
                prodList = cartProd(getCell(col, i), getCell(col+1+i, row - (i+1)));

                if(prodList == null) {
                        //One or more of the cells were null.
                        //The product is also null
                        continue;
                }

                if(i == 0) {
                        //There are no duplicates from the first product
                        strList.addAll(prodList);

                } else {
                        //Check for the existing list for duplicates; only fill in unique pairs
```

O(1)

**If input string length is 5,**
**we will have 5 rows and 5 colums of CYK table**

**Therefore, if input string length is n,**
**then we will have n x n CYK table.**

**This indicate that outter for loop will run n times.**



4x4

Those cells will fill up with result of cartesian products.

Based on my drawing, we will fill up all cells except last row that contains n cells, we will fill up rest of cells with result of catesiant product.
so, this for loop will run approximately O(n).

```
                for(int j = 0; j < prodList.size(); j++) {
                        if(!(strList.contains(prodList.get(j)))) {
                                strList.add(prodList.get(j));
                        }
                }
        }
}

//List of rules that a String belongs to
ArrayList<String> strRules;

for(int i = 0; i < strList.size(); i++) {
        //Get the list of rules that each String in the produced list can be produced through
        strRules = cnf.belongsToRules(strList.get(i));

        if(strRules == null) {
                //If there are no rules producing this String, it will be null.
                //Don't fill in anything in the rule list
                continue;
        }

        if(i == 0) {
                //There are no duplicates for the rules producing the first String
                rules.addAll(strRules);

        } else {
                for(int j = 0; j < strRules.size(); j++) {
                        //Check for duplicates
                        if(!(rules.contains(strRules.get(j)))) {
                                rules.add(strRules.get(j));
                        }
                }
        }

}
```

total : Outter for loop and innter for loop time compelxity = O(n^2)

O(n)

$\sum_{i=1}^{n-1} n - i$

```
                //Return the list of rules that can produce Strings of this cell
                return rules;
        }
}
```

T(n) = 1 + n^2 + n , so this function takes O(n^2) time complexity

```
/**
 * Returns the list of Strings in the given cell of the table
 * @param col The column number of the cell
 * @param row The row number of the cell
 * @return The list of Strings in the given cell of the table
 */
public ArrayList<String> getCell(int col, int row) {
        ArrayList<String> str = this.cykTable[row][col];
        if(str == null || str.isEmpty()) {
                return null;
        }
        return str;
}

/**
 * Returns a list of ordered pairs formed as a result of a Cartesian product between 2 sets of strings
 * @param str1 The first set of Strings
 * @param str2 The second set of Strings
 * @return The resulting Cartesian product of the 2 strings
 */
public ArrayList<String> cartProd(ArrayList<String> str1, ArrayList<String> str2) {
        //The "product" of 2 Strings
        String prod;
        //The list resulting from the Cartesian Product of the given list of Strings
        ArrayList<String> prodList = new ArrayList<String>();

        //If either of the Strings are null, the resulting product is null
        if(str1 == null || str2 == null) {
                return null;
        }
```

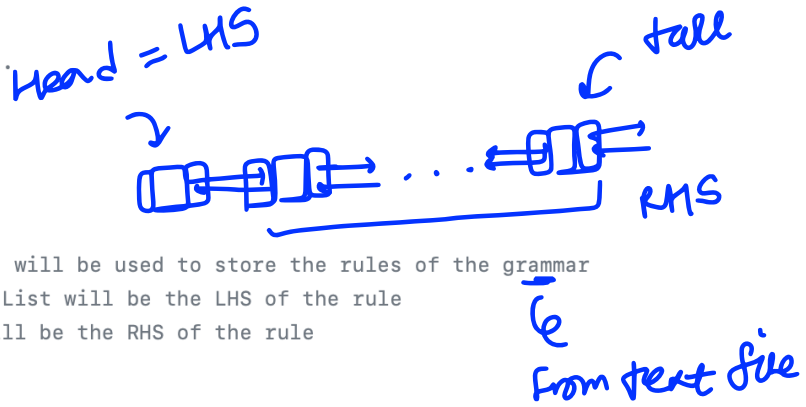
total running time for inner and outter for loop => O(n^2)

```
                //Perform Cartesian product of the Strings in the list
                for(int i = 0; i < str1.size(); i++) {
                        for(int j = 0; j < str2.size(); j++) {
                                //The Cartesian product is the concatenation of the Strings
                                prod = str1.get(i) + str2.get(j);

                                //Check if the pair is already in the list. If not, add it to the list
                                if(!(prodList.contains(prod))) {
                                        prodList.add(prod);
                                }
                        }
                }

                //return list of Cartesian products
                return prodList;
        }

        /**
         * Check whether given String can be obtained through this grammar
         * @return True, if the String can be obtained from this grammar. Else, false
         */
        public boolean checkInGrammar() {
                //Get the topmost (0, n-1) cell of the table
                ArrayList<String> topCell = this.getCell(0, checkString.length() - 1);

                //The String can only be obtained through this grammar if the cell contains the start state.
                //Check whether the cell is null, or contains the start state. Returns true or false accordingly

                if(topCell == null) {
                        return false;
                }
                return topCell.contains(cnf.getStart());
        }
}
```

**Conclusion for CYK class,**
**This class will have approximately O(n^3) running time complexity,**
**because of CYK() function**

2. CNFParse.java

```java
package srcCode;

import java.util.ArrayList;
import java.util.Arrays;

/**
 * Class which parses the CNF rules.
 * @author anush
 */
public class CNFParse {

	/*	*
	 * An array of Linked Lists will be used to store the rules of the grammar
	 * The head of each Linked List will be the LHS of the rule
	 * The rest of the nodes will be the RHS of the rule
	 */
	Rule[] cnfRules;

	/*
	 * The rules of the grammar
	 */
	ArrayList<String> ruleset = null;

	/*
	 * The start symbol of this grammar
	 */
	String start;

	/**
	 * Constructor of the parser class
	 * @param rules An array containing the rules of the grammar.
	 */
	public CNFParse(ArrayList<String> rules) {
		this.ruleset = rules;
```



```
//The start symbol is the first lhs in the ruleset
this.start = rules.get(0).split("->")[0].trim();

//Number of linked lists = number of rules because each rule will be a linked list
this.cnfRules = new Rule[rules.size()];

//Loop to go through all the rules and store them as individual pairs
for(int i = 0; i < rules.size(); i++) {
        /*
         * Each line will be a rule in the form lhs -> rhs1 | rhs2 ...
         * The lhs and rhs are separated by "->"
         * The strings in the rhs are separated by "|"
         * Break down each rule to get the lhs of the rule,
         * and an array of all the possible rhs using this rule
         */
        String lhs = ruleset.get(i).split("->")[0].trim();
        String[] rhs = ruleset.get(i).split("->")[1].split("[|]");


        for(int j = 0; j < rhs.length; j++) {
                //Get rid of any leading and trailing spaces in the strings
                //This ensures that only the exact strings of the grammar are added to the list
                rhs[j] = rhs[j].trim();
        }


        this.cnfRules[i] = new Rule(lhs, new ArrayList<String>(Arrays.asList(rhs)));
    }
}

/**
 * Returns the start symbol of this grammar
 * @return The start symbol of this grammar
 */
```

take O(1) since, we just have to seperate lhs and rhs based on" ->" notation

O(n)

if textfile contains n row of rules than we have n number of linked list

2

For exmaple)
we have S -> ab | bc | cc | E
then we have rule size 5
lhs array length will be always 1 and
rhs array length will be total rule size - lhs array length.

If we let total rule size as n,
lhs.length will be 1
rhs.length will be n-1

outter for loop runs n times
inner for loop runs n-1 times
so total run time for this wil be O(n^2)

This will learn n-1, so O(n-1)

```java
public String getStart() {
        return this.start;
}

/**
 * Returns the rhs of the rules for a given lhs
 * @param left The given lhs that we want the rules of
 * @return The rhs of the rules
 */
public ArrayList<String> getRule(String left) {
        for(int i = 0; i < this.cnfRules.length; i++) {
                //Find the lhs in the set of rules we have
                if(this.cnfRules[i].getLeft().equals(left)) {
                        //if specific lhs found, return the rhs
                        return this.cnfRules[i].getRight();
                }
        }

        //if lhs not found, return null
        return null;
}


/**
 * Return a list of lhs of rules that a given rhs can be obtained from
 * @param right The rhs that we want to know how it can be obtained
 * @return A list of lhs of rules from which this rhs can be obtained
 */
public ArrayList<String> belongsToRules(String right) {
        ArrayList<String> lefts = new ArrayList<String>();

        for(int i = 0; i < this.cnfRules.length; i++) {
                //check whether the given string is an rhs of the rule
                if (this.cnfRules[i].inRule(right)) {
                        //add the lhs of the rule to the list
                        lefts.add(this.cnfRules[i].getLeft());
                }
        }

        //if the list is empty, rhs cannot be obtained through any rules
        //return null
        if(lefts.isEmpty()) {
                return null;
        }

        return lefts;
}

}
```

O(1)

if lhs rule matches this for loop will run all the down to rhs, so it will take O(n)

O(n)

run n times.

For CNFParse class, each for loops will run O(n^2) , O(n), and O(n) since highest degree of time complexity is dominant, this class will take approximatley O(n^2) running time.

3. Rule.java

```java
import java.util.ArrayList;

/**
 * Class which holds a Rule
 * @author anush
 */
public class Rule {

	/**
	 * The lhs of the rule
	 */
	private String lhs;

	/**
	 * A list of the rhs that can be obtained through this rule
	 */
	private ArrayList<String> rhs;

	/**
	 * Default constructor initializing an "empty" rule
	 */
	public Rule() {
		this.lhs = "";
		this.rhs = new ArrayList<String>();
	}

	/**
	 * Constructor that initializes a rule
	 * @param left The lhs of the rule
	 * @param right A list of the rhs that can be obtained through this rule
	 */
	public Rule(String left, ArrayList<String> right) {
		this.lhs = left;
		this.rhs = right;
	}
```

```java
/**
 * Returns the lhs of the rule
 * @return The lhs of the rule
 */
public String getLeft() {
        return this.lhs;
}

/**
 * Returns the list of rhs that can be obtained by this rule
 * @return The list of rhs that can be obtained by this rule
 */
public ArrayList<String> getRight() {
        return this.rhs;
}

/**
 * Adds a string to the list of rhs that can be obtained by this rule
 * @param right A string that can be obtained by this rule
 */
public void addRight(String right) {
        rhs.add(right);
}

/**
 * Returns the number of strings that can be obtained by this rule
 * @return The number of strings that can be obtained by this rule
 */
public int ruleLegth() {
        return rhs.size();
}

/**
 * Checks whether a string can be obtained from a rule
 * @param right The string that we want to check whether it can be obtained by this rule
```

```
     * @param right The string that we want to check whether it can be obtained by this rule
     * @return True, if the string can be obtained by this rule. Else, false
     */
    public boolean inRule(String right) {
            for(int i = 0; i < this.rhs.size(); i++) {
                    //check if the string is in the list of rhs of this rule
                    if(rhs.get(i).equals(right)) {
                            return true;
                    }
            }
            return false;
    }

    /**
     * Returns a String representation of the rule
     */
    public String toString() {
            String str = lhs + " -> " + rhs.get(0);

            for(int i = 1; i < rhs.size(); i++) {
                    str += " | " + rhs.get(i) + " ";
            }

            return str;
    }

}
```

O(n)

**if the total length of rule is n**
**then left hand side is 1 symbol so the length of lhs will be 1**
**and rest of symbols will be included in rhs.**
**so rhs.size() will be n-1.**
**So inRule() function has O(n) time complexity**

**Conclusion : Rule class is very simple class that used for divide lhs with rhs rule, therefore this class will take approximatly O(n) time complexity.**

4. Driver.java

```java
package srcCode;
import java.util.ArrayList;
import java.util.Scanner;
import java.io.*;

/**
 * Main class used to run the program
 * @author anush
 */
public class Driver {

	/**
	 * Read a given file and parse the rules of the CNF grammar from the file
	 * @param filename The name of the file to read
	 * @return The list of cnf rules of the grammar contained in the file
	 */
	public static ArrayList<String> readFile(String filename) {

		try {
			//open file
			File file = new File(filename);
			Scanner filesc = new Scanner(file);

			//for loop to read lines in file and form an array of strings (rules)
			ArrayList<String> lines = new ArrayList<String>();
			while(filesc.hasNext()) {
				lines.add(filesc.nextLine());
			}

			filesc.close();
			return lines;

		} catch (FileNotFoundException e) {
			//Catch exception if the file is not found
			System.out.println("Specified file not found. Please try again.");
```

O(n)

```
        } catch (Exception e) {
                //Catch any other errors and display the error message to user
                System.out.println(e.getMessage());
        }

        return null;

}

/**
 * Main function which is called when the program is run
 * @param args A string of arguments
 */
public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);
        ArrayList<String> rules;
        String filename;

        do {
                //ask user for file name
                System.out.println("Enter name of file containing grammar in CNF");
                 filename = sc.nextLine();

                 //Read and parse the rules in the file
                rules = readFile(filename);

        }
        //loop continues to run until a file has been read and parsed successfuly
        while (rules == null);


        String cont = "n";

        do {
                //accept string to be checked from user
```

```
                //call CYK(checkstring, rules[])
                System.out.println("Enter string that you want to check: ");
                String checkStr = sc.nextLine();

                //If the string is empty, give it E terminal for empty string
                if(checkStr.length() == 0) {
                        checkStr = "E";
                }

                //Build a cyk table for the given grammar and string
                CYK cyk = new CYK(checkStr, rules);

                //Check if the given string can be formed through this grammar
                if(cyk.checkInGrammar()) {
                        System.out.println(checkStr + " CAN be obtained through this grammar!");

                } else {
                        System.out.println(checkStr + " CANNOT be obtained through this grammar.");
                }

                System.out.println("--------------------------------------------------------------------------------");

                //Ask if the user would like to continue checking more strings
                System.out.println("Would you like to continue checking another String?\n"
                                + "Enter Y/N");
                cont = sc.nextLine();

                if(cont.equalsIgnoreCase("y")) {
                        //Ask if the user would like to use a different grammar
                        System.out.println("Would you like to use a different grammar?\n"
                                        + "Enter Y/N");
                        filename = sc.nextLine();

                        //Parse the rules of the new grammar
                        if(filename.equalsIgnoreCase("y")) {
                                do {
                                        //ask user for file name
                                        System.out.println("Enter name of file containing grammar in CNF");
                                         filename = sc.nextLine();

                                        //Read and parse the rules in the file
                                        rules = readFile(filename);

                                }
                                //loop continues to run until a file has been read and parsed successfuly
                                while (rules == null);
                        }
                }

        }
        //Loop continues to run until the user indicates they wish to quit the program
        while(cont.equalsIgnoreCase("y"));

        sc.close();

    }
}
```

CYK class takes approximately O(n^3)