# Data Structures and Algorithms

## Coursework 2 – 100122248

## Contents


Coursework 2 – 100122248 ..... 1
Question 1: ..... 2
Pseudocode ..... 2
Formal Analysis ..... 3
Fundamental Operation ..... 3
Case description ..... 3
Run Time Complexity Function and Order of Algorithm ..... 3
Question 2: ..... 4
Pseudocode ..... 4
contains() ..... 5
outputBreadthFirstSearch() ..... 5
outputDepthFirstSearch() ..... 6
getSubTrie() ..... 7
getAllWords() ..... 7

# Question 1:

## Pseudocode

Here is the pseudocode for the formDictionary algorithm:

**Input:** An arrayList from a file which has been called from the function readWordsFromCSV.

**Output:** A list of the sorted words and frequencies sortedList.

```
Hashmap wordDictionary
ArrayList words, sortedList
integer i
Set wordSet

FOR i=0 to words.size do
      IF wordDictionary CONTAINS KEY(words.get(i))
            wordDictionary ADD (words.get(i) + frequency)
      ELSE IF
            wordDictionary ADD (words.get(i))
      END IF
END FOR
FOR (word Object in the wordSet)
      ADD word Object to sortedList
END FOR
Sort the sortedList
return sortedList
```

## Formal Analysis

## Fundamental Operation

The fundamental operation of the code I have found is after the 1st For Loop and is the comparison if the wordDictionary contains an existing word or not.

```
IF wordDictionary CONTAINS KEY(words.get(i))
```

## Case description

The Worst Case would be that the wordDictionary have to add an extremely high amount of words, with the Best case being that the wordDictionary would only have to add one word. If the text had a very long length, the time taken to complete the algorithm would be very high.

## Run Time Complexity Function and Order of Algorithm

I calculated the Run Time Complexity using the for loop before the fundamental operation:

```
FOR i = 0 to words.size
```

Using the summation, I used this formula to work out the Run Time Complexity function, using the "words.size" constant to equal 5.

$$O(n) = \sum_{i=1}^{x}$$

$$O(n) = x$$

From this, we can deduce that the Order of the Algorithm would be $O(n)$ as the highest power of n is $n^1$.

# Question 2:

## Pseudocode

### add()

**Input:** String key which is the word going to be input into the Trie

**Output:** A Boolean addSuccessful to see whether the word has been added successfully

```
Boolean addSuccessful
TrieNode temp, next

temp <- root

FOR (character in the key) do
      next <- temp.getOffspring(character)
      IF(next node is null)
            temp.getOffspring(character)
            next <- temp.getOffspring(character)
      END IF
      temp <- next
END FOR
      set the end of word
IF(temp.endOfWord is TRUE)
      addSuccessful = TRUE
END IF

return addSuccessful
```

## Order of Algorithm

The algorithm's order is O(n) due to one for loop in the algorithm

## contains()

**Input:** String key which is the word going to be input into the Trie

**Output:** A Boolean to see whether the word is in the trie

```
Boolean addSuccessful
TrieNode temp, next

temp <- root

FOR (character in the key) do
      next <- temp.getOffspring(character)
      IF(next node is null)
            return false
      END IF
END FOR

return true
```

## Order of Algorithm

The algorithm's order is O(n) due to one for loop in the algorithm

## outputBreadthFirstSearch()

**Input:** A trie of words starting with TrieNode root

**Output:** A string of stringBuild of the result of the search.

```
String stringBuild
TrieNode temp
LinkedList queue

temp <- root
ADD temp to queue

WHILE (queue is not Empty) do
      REMOVE temp from queue
      FOR(node in the temp.offspring)
            IF(node is not NULL)
                  add node to stringBuild
                  add node to queue
            END IF
      END FOR
      temp <- head of the queue
END WHILE
return stringBuild
```

## Order of Algorithm

The algorithm's order is O(n) due to one for loop in the algorithm

## outputDepthFirstSearch()

**Input:** A trie of words starting with TrieNode root and a string.

**Output:** A string of stringBuild of the result of the search.

```
String stringBuild
TrieNode temp

temp <- root
FOR(node in the temp.offspring)
      IF(node is not NULL)
            add node to stringBuild
            add node to queue
      END IF
END FOR
      temp <- head of the queue
END WHILE
return stringBuild
```

## Order of Algorithm

The algorithm's order is O(n) due to one for loop in the algorithm

## getSubTrie()

**Input:** A string prefix

**Output:** A trie subTrie which is formed from the prefix

```
TrieNode temp
Trie subTrie

temp <- root
FOR(each character in the prefix) do
      temp <- temp.getCharacter(character)
END FOR
      subTrie <- new Trie from root temp

return subTrie
```

## Order of Algorithm

The algorithm's order is O(n) due to one for loop in the algorithm

## getAllWords()

**Input:** A trie, a StringBuilder

**Output:** A List of all the words called allWords within the trie

```
List allWords
StringBuilder string
TrieNode node

APPEND to the string the node
IF(node is the leaf)
      ADD to allWords(string)
END IF
FOR(each TrieNode n in the node offspring)
      IF(n is not NULL)
            getAllWords()
      END IF
END FOR
IF(string length is more than 0)
      delete last character of the string
END IF
```

## Order of Algorithm

The algorithm's order is O(n) due to one for loop in the algorithm

# Question 3:

## Pseudocode

```
Input: csv files containing words and the prefix queries
Output: a csv file which contains the query result

DictionaryMaker dictionary
Trie trie, subTrie
HashMap freqValues
ArrayList words, wordlist, prefixList, wordPartList, freqList,
sortedList
Scanner scan
String[] part
String wordPart
int freqPart, wordCounter
List allWords
Set freqSet


WHILE(scanner has a newLine)
      add line to the prefixList
END WHILE
FOR(each word in the wordList)
      SET splitter for string
      wordPart <- part[0]
      freqPart <- part[1]
      ADD to wordPartList the wordPart
      ADD to freqList the freqPart
END FOR
FOR(each prefix in the prefixList)
      subTrie <- trie.getSubTrie(prefix)
      allWords <- subTrie.getAllWords()
      FOR(each word in allWords)
            freqValues.put(value of frequency in the word, word)
      END FOR
      freqSet <- freqValues.entrySet()
      FOR(each wordObject in the freqSet)
            ADD to sortedList(wordObject)
      END FOR
      Sort the sortedList
      FOR (each string in the sortedList)
            wordCounter += 1
            print string to File
            IF(wordCounter = 5)
                  break out of loop
            END IF
      END FOR
```

```
      clear the freqValues
END FOR
```

## Order of Algorithm

The algorithm has multiple for loops in them and a nested for loop, so the Order of the algorithm would be $O(n^2)$ .