FIT2004 Assignment 1

FIT2004 Assignment 1

Student: Siew Ming Shern

StudentID: 28098552

Siew

Task 1

```
function preprocess(filename)

    Let specialChar be tuple that store collection of **punctuation** character.

    Let exclusionStr be tuple that store collection of **auxiliary verbs** and **articles**.

    Open filename and read the content, then store the content to filecontent variable

    Let currentString be empty string

    Let newarray be empty array

    ForEach Character in filecontent

        If Character is whitespace then
            If currentString is not in exclusionStr
                Add currentString in newarray
            Initialize currentString with empty string

        Else If Character in specialChar then
            Do nothing

        Else
            Add Character to currentString

        If Character is last character of filecontent and currentString is not empty then
            If currentString is not in exclusionStr then
                Add currentString in newarray
```

Time Complexity:

Best Case: O(nm)

Worst Case: O(nm)

where n is number of words and m is maximum number of characters in a word.

Best Case == Worst Case, File is open and read from character to character and will not be able to terminate early.

Space complexity:

Best Case: O(nm)

Worst Case: O(nm)

where n is number of words and m is maximum number of characters in a word.

Best Case == Worst Case, List is created in memory requires to stores string representing n words with m characters, there is no way to save up memory space to store these words.

Task 2

Findmaxwordslength(array) is implemented to find maximum length of words in array and return maximum length of words in array.

countingSort(array,position) is implemented to sort array using counting sort algorithm based on position of character of each words.

```
function wordSort(array)

        Let Maximum be return value of Findmaxwordslength(array)

        For position start from Maximum to 1, decrement by 1 at each iteration.

            countingSort(array,position)

        return array
```

Time Complexity:

Best Case: O(nm)
Worst Case: O(nm)
where n is number of words and m is maximum number of characters in a word.

Best Case == Worst Case, Both cases would requires programs to Go through each word, looking at from far right character and, sort it and the words into the temporary array(created in counting sort). Since we have n words, it is O(n) for going through every words inclusive with replacing original list with item counting sort list.
Then each n words has at most m characters for the longest word in n words and we only need to consider 26 character at most in counting sort array,
which will takes O(m). so we repeat m times giving us O(m) * O(n), resulting O(nm).

Space complexity:

Best Case: O(nm)
Worst Case: O(nm)
where n is number of words and m is maximum number of characters in a word.

Best Case == Worst Case, input varies from n number of words with m maximum characters, input array will have n words with at most m characters
which would be taken as consideration for space complexity and Building array of size 27 to represent each for blank character and a-z character
for counting sort array. therefore, constant size array is build and the constant build array will need to store n words with at most m characters in memory.
This resulting O(nm).

Task 3

```
function wordcount(array)
    Let newarray be empty array
    Add length(array) to newarray
    Let currentword be first word of array
    Let wordnumber be number of currentword appear in array.
    Wordnumber is initalise with 1
    Foreach item of array index 1 to length of array
        If item = currentword then
            Increase wordnumber by 1
        Else
            Add list(currentword,wordnumber) to newarray
            Currentword = item
            Wordnumber is initalise with 1

        If item is last item of array then
            Add list(currentword,wordnumber) to newarray
```

Time Complexity:

Best Case: O(n)
Worst Case: O(nm)
where n is number of words and m is maximum character in words.
Best Case: When every words have different length, it doesn't need to tranverse m times to compare two strings.
Worst Case: when every words have same length, it would need to tranverse m times to compare each n words Making it O(n)*O(m) to O(nm).

Space Complexity:

Best Case: O(nm)
Worst Case: O(nm)
where n is number of words.
Best Case == Worst Case, this function will need to store n words with atmost m characters with thier count to list. so, it takes O(nm) to store n words and atmost m character and there is no way to save up memory space complexity.

Task 4

Class code for Heap can be found under FIT1008, week 12 from *https://lms.monash.edu/course/view.php?id=42395&section=19#19*

setReverse in class method – Transform minheap to sorted Descending order.

ListofList – original list

Ktop – top kth item specify by user

```
function kTopWords(kTop, listofList):

      if kTop <= 0: kTop = 1

      newHeap = MinHeap(kTop)

      counter = 0

      for item in listofList:

          if len(newHeap) < kTop:
              newHeap.add(Frequency of item, counter)

          elif Frequency of item > newHeap.getmin():
              newHeap.extract()
              newHeap.add(Frequency of item, counter)
          counter += 1

      newArray = []

      for index in newHeap.setReverse():
          newArray.append(listofList[index])

      return newArray
```

Time Complexity:

Best Case: O(klogk)
Worst Case: O(nLogk)
where n is number of item in original list and k is kth number of top-most frequent words.

Best Case: Happens when first kth item is already in min-heap array data structure, rise() method is terminated as early as O(1) which result O(n) for first loops but delete-min still cost O(klogk) which cost O(n+klogk). Hence, time complexity of this best case resulting O(klogk).

Worst Case: Happens when at each n iteration, new child nodes is needed to swap all the way to becomes parent nodes or new child nodes is already bigger than parent nodes or had higher index value for same word frequency which would takes O(nlogk). Then second loops for delete minimum operation would require the last node as root to shift all the way to leaf node which would takes o(klogk). Nonetheless, worst case comes when k reaches n, where all item is inserted to min heap, rise() method has to be called at n iteration and delete min will takes O(klogk) resulting O((n+k)logk) but k never reaches n so, at worst, it just double n values which results O(nlogk).

Space Complexity:

Best Case: O(km)
Worst Case: O(km)
where m is maximum character of string and k is kth number of top-most frequent words.

Best Case == Worst Case, this function will always needs km memory space to fit k item with m maximum character and there are no way to avoid consumption of memory space.