James Schubach (29743338) – FIT2004 – Assignment 1

Task 1: Preprocessing

- My algorithm for this opens the file for reading, then reads line and within each line reads each letter. It then compares if that letter is part of a list of punctuation. If it isn't it adds it to a variable called word. However, if the next letter is apart of that it will then check to make sure that the word is not a verb/ article and that it isn't a space. If it passes that, it will add the word to a list and then clear that word. Otherwise it will clear the word anyway,
- Its time complexity is O(nm) being the number of words times by the max number of letters in a word. The comparing of a letter in a list is O(1) because that list size is fixed. And comparing a word to a list is O(m) which is still smaller then O(nm)
- The space complexity is O(nm) as well as the list size is n and the max amount per word is m. Once again the verbs and punctuation lists are O(1) because they are fixed and then we only have variables.

Task 2: Sorting the words

- My implementation for wordSort finds the max length of a word, and goes through a loop starting from the last index of the max length word and starts a radixSort at that index.
- The radix sort then goes through and implements a variation of count sort. It will then sort the words by the ith column. It will continue this till it gets to the first column and thus have it sorted alphabetically.
- The time complexity is also O(nm) as radix sort is O(n) and it must do radix sort m times, m being the max number of letters in a word. Thus, we have O(n*m) and finding the max number is n. Therefore O(n*m + n) which is still O(nm)
- Space complexity for the algorithm is O(nm) as we have a bunch of variables which are O(1) and then we have 2 lists of size m, and an list of size n. Therefore we have O(nm + m + m + n) which reduces to O(nm)

Task 3: Showing the number of total words including the frequency of each word

- This task I made an algorithm that loops through the list and checks if the word equals the current word and if it does ups the count. As soon as the word doesn't equal the current_word it updates the count at the start of the list, and then adds the current word and current count. It then resets the count and updates the current word. After the for loops finishes it adds up the final count and then adds the current word and count.
- Time complexity for this algorithm is O(nm) this is because it loops through the list once and will also loop through the letters up to m. Therefore nm.
- Space complexity of the algorithm is also O(nm), this because we have a bunch of variables which are O(1) and then we have a word count the same size as the alist. Thus we have O(nm + nm) and reduces to O(nm)

Task 4: Showing the k top most words appears in the writing

- My implementation for this problem uses a min-heap and pushes the first k elements to the heap, we then pop the root and push one by one the k+1 element if they are larger than the root of the heap. At the end this gives us a list of the k largest elements in the input list. However, this does not give them in stable order. So we have to resort it so that we get a stable order.

- Time complexity of this algorithm is O(nlogk), as we have to build the heap of size k, which is log(k) and we have to do that n times. This gives us nlogk. This is because pushing an element is log(k) and so is popping an element is logk. Then we count sort the list which is O(n). We then have wordSort which is km Therefore we have O(nlogk + logk + km), in which we reduce down to O(nlogk + km).
- Space complexity for this algorithm is O(km). As we create a heap of size O(k) and has max of m words thus we have km. wordSort2 has a complexity of O(k+m) and counting sort has a complexity of O(k + r) , where r is the highest number of occurrences, r <=n Therefore we have O(km + km + km + k + r), where r,m <= km then we have O(km) otherwise O(km + r + m)