Question 1

1)

My program has two methods declared – getValidInput and getVocab.

The getVocab method retrieves an ArrayList of words from the google-10000-english-no-swears.txt file. This is done by reading each line individually, then splitting the line by regex space characters. Then each word is trimmed by removing one leading and trailing punctuation character, and converting the word to lower case. This is so as to minimise any assumptions made about the nature of the file.

The getValidInput method takes an ArrayList of dictionary words to compare against. It then reads the file Input219.txt line by line, splitting the line by regex space characters. Then we create a new variable to store a modified word, replacing one leading and trailing punctuation character and converting to lower case. If this modified word is in the ArrayList of dictionary words, then we add this to our valid words list to return. This uses the ArrayList contains method, which has O(n) complexity.

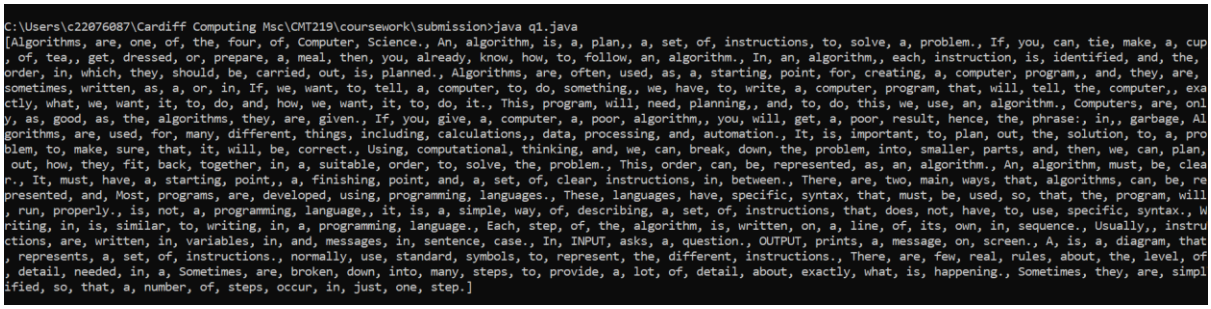
2)

Checking whether a word is contained within the ArrayList dictionary has O(n) complexity for *n* words in our dictionary. This must be checked for *m* words in our input text. Therefore our overall complexity for checking whether a word is contained in the dictionary can be expressed as O(n*m), for *n* dictionary words, and *m* input words.

3)

The efficiency could be improved by using a different data structure to store the dictionary words in. For example, a HashSet seems ideal here. This has a .contains() method with constant (O(1)) time complexity, and would reduce the overall complexity to O(m) for *m* words in our input.

Below is an example of the Command Prompt output for my program using the files provided.

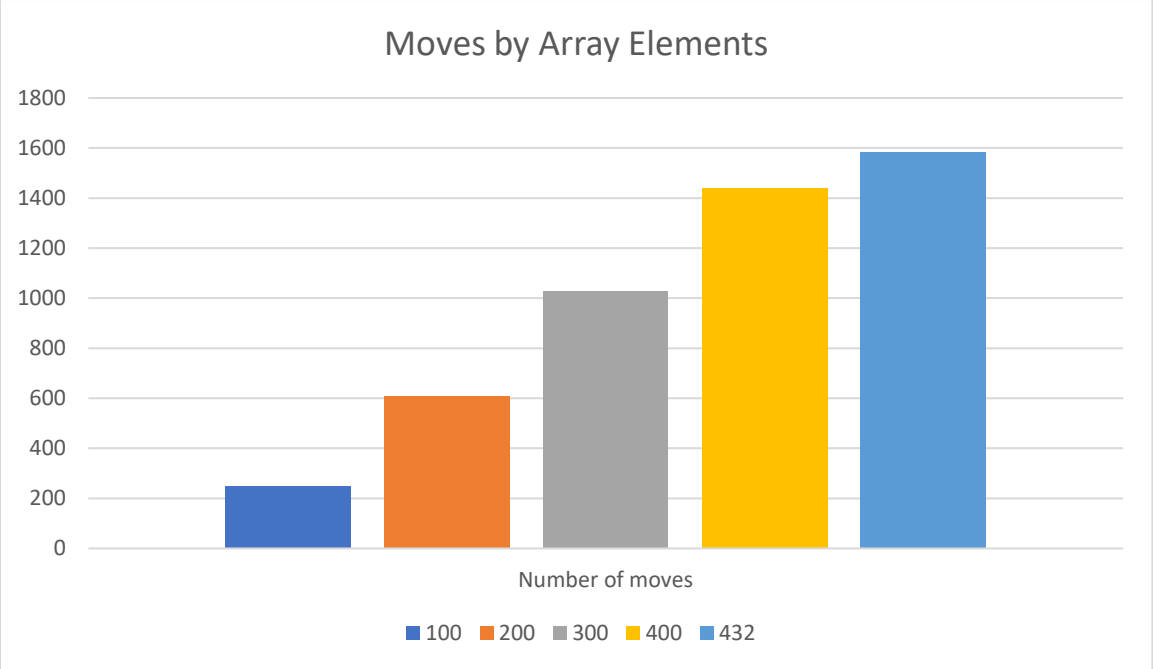


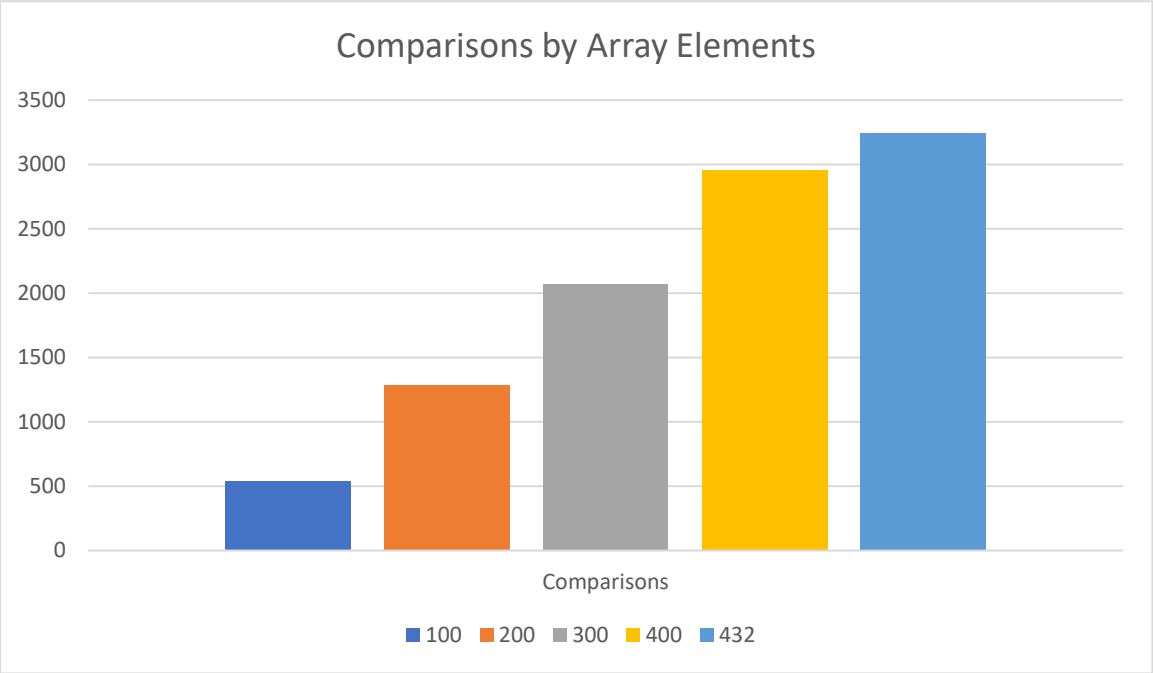
Question 2

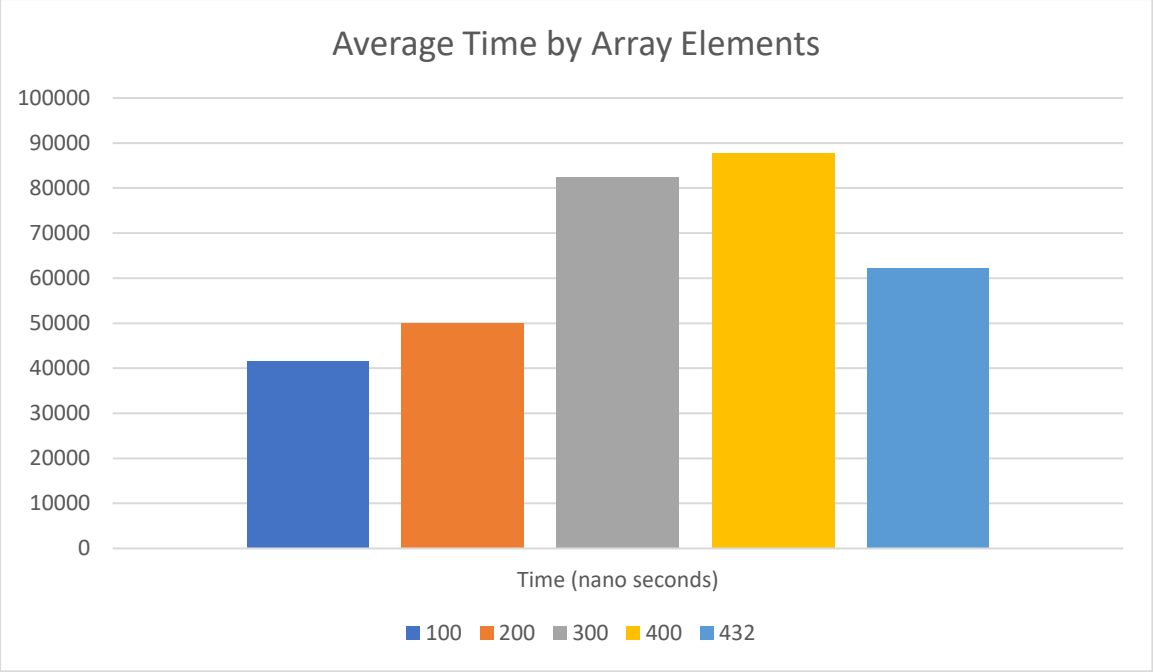
1.

There were 432 total words to be sorted from question 1, so I considered the first 100, 200, 300, 400 and all 432 elements in my tests.

I tested the timings of my code over an average of 1000 runs per array length. However I received some interesting fluctuations within my results, which could be caused by a variety of environmental factors on my machine.


Moves by Array Elements
1800
1600
1400
1200
1000
800
600
400
200
0
Number of moves
100
200
300
400
432


Comparisons by Array Elements
3500
3000
2500
2000
1500
1000
500
0
Comparisons
100
200
300
400
432



2. There are two main functions within my program, mergeSort and merge.

The mergeSort function kicks off a sort of an ArrayList of strings. It takes an ArrayList, start and end index. If this start index is not less than the end index, then the array has a length of one and does not require sorting. Otherwise we split the given array into two sections, and finding a middle index. Then we call the mergeSort algorithm again on the left and right arrays. Once these are sorted we then merge the two halves using the merge algorithm, creating copies of the arrays to use as references when switching elements.

The merge function first declares two iterables, i and j as indexes for the left and right arrays. Then we compare the element at position i to the one at position j, adding the alphabetically first element to the array we are merging into. Each time we add an element we increment either i or j for the respective array that the element came from. Once we have finished adding from one of the arrays, we finally add the remaining elements from the other array.

This sorts the given array via the merge sort algorithm.

Our class has two additional variables, comparisons and swaps. The comparisons variable measures the number of times we compare two elements from the left and right arrays of our merge function. The swaps variable measures each time we swap the order of the elements in two arrays, so this is incremented every time we merge an element from the right array into the target array while there are still elements to be merged in the left array.

There are also three additional functions. We can reset the variables swaps and comparisons by calling the resetVars function, and print them to the console by calling the printVars function. The getAverageTime function takes a number of iterations, an ArrayList, and a length. Then it runs the mergeSort function to sort the copied ArrayList until the number of iterations have been reached, and outputs an average time in nanoseconds for the function to be run.

The main function of our class declares several ArrayList variables using the output from question 1, for 100, 200, 300, 400 and 432 elements. Then it runs the mergeSort function on each list, counting

the number of swaps and comparisons, and finally outputting them to the console. Then each list is run 1000 times and an average time printed to the console.

3. Below is the Command Prompt output for my program.

```
C:\Users\c22076087\Cardiff Computing Msc\CMT219\coursework\submission>java -
Mergesort 100 elements
comparisons: 542
swaps: 248
Mergesort 200 elements
comparisons: 1284
swaps: 610
Mergesort 300 elements
comparisons: 2075
swaps: 1027
Mergesort 400 elements
comparisons: 2959
swaps: 1442
Mergesort 432 elements
comparisons: 3246
swaps: 1582
Average Time over 1000 runs for first 100 elements:
32371
Average Time over 1000 runs for first 200 elements:
50868
Average Time over 1000 runs for first 300 elements:
70357
Average Time over 1000 runs for first 400 elements:
87658
Average Time over 1000 runs for first 432 elements:
69218
```

Question 3

A)

The Strategy design patten allows us to choose an algorithm to run from a selection of different "Strategies" in our code depending on our needs. If we have a section of code that handles multiple types of problems which necessitate different behaviours, then we can apply a set strategy for each problem type. For example, if we need to save files in different locations or formats depending on some input, we can apply a selection of prewritten strategies to handle this for us. Each file format would in this instance be a strategy, and the differing formats would be the problem.

This allows us to better apply the DRY principle of writing code, as we only need to write each strategy algorithm separately. The rest of our code can be reused making our code more readable and maintainable. For example if a new strategy is required in the future, or an existing one needs modification, then it is only the individual strategy code that needs to be updated.

B)

Completed code:

return(-a.cost);

C)

Completed code:

return(a.practicality);

D)

Completed code:

Line 25 in ProductRecommender:

int best_score = scoringStrategy.getScore(first_product);

Line 32 in ProductRecommender:

int current_score = scoringStrategy.getScore(current_product);

References:

Code:

_Gaurav_Tiwari. 2018. *ArrayList clone() method in Java with Examples.* Available at: https://www.geeksforgeeks.org/arraylist-clone-method-in-java-with-examples/ [Accessed: 10 May 2023]

Oracle. 2023. *Pattern (Java Platform SE 8).* Available at: https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#split(java.lang.String) [Accessed: 12 April 2023]

Oracle. 2020. *String (Java Platform SE 7).* Available at: https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html [Accessed: 12 April 2023]

Report:

Baeldung. 2019. *Performance of contains() in a HashSet vs ArrayList.* Available at: https://www.baeldung.com/java-hashset-arraylist-contains-performance [Accessed: 10 May 2023]

IONOS. 2020. *Strategy pattern: software design patterns for variable behavior strategies.* Available at: https://www.ionos.com/digitalguide/websites/web-development/strategy-pattern/ [Accessed: 11 April 2023]