Data structures and Algorithms used:

Data structures implemented within FeedAnalyser.java incorporate the use of a HashMap(Integer, FeedItem), HashTable(Long, FeedItem), LinkedList(Char), Array and a TreeMap(Integer, FeedItem). Algorithms used include the Boyers Moore's Algorithm, which is a string matching algorithm that determines if one string is a substring of another (Www-igm.univ-mlv.fr, 2019).

Worst Case Time Complexity:

The constructor makes use of a TreeMap which has a worst-case time complexity of O(log(n)) where ‘n’ is the number elements mapped in the tree. The other data structures such that of the Hash family and LinkedList have a Big O worst-case time complexity of O(n), where n is the number of elements contained/mapped. Furthermore the constructor makes use of a while loop iterator which is represented as O(n), where n is the number of FeedItems being iterated over. Hence mitigating the lower order terms of the LinkedList and Hash family data structures, the constructor has a worst-case time complexity of O(n • log(n)). The search method ‘getPostsBetweenDates’ makes use of the Boyers Moore Algorithm, and the username is preprocessed into a character array, which is done before the matching process optimise runtime efficiency, this call is done in O(n + m), where n is the number of characters in the username, and m is for preprocessing. The loop iterates through the TreeMap data structure which has a worst case time complexity of O(log(z)), where ‘z’ is the number of elements in the tree mapping. The inner loop function call to the Boyers Moore's algorithm runs in O(n + m) where n is the number of FeedItem and m is for preprocessing the pattern, hence having an overall worst-case time complexity of O((n + m) • log(z)). The search method ‘getPostAfterDate’ runs similarly with a worst-case time complexity of O((n + m) • log(z)) where z is the number of elements in the TreeMap, and n and m are for the Boyers algorithm. This involved accessing a TreeMap to iterate over each FeedItem and Boyers Moore's algorithm to find the corresponding username (Bigocheatsheet.com, 2019). The search ‘getHighestUpVote’ has a worst-case time complexity of O(n^2) where n is the number of FeedItems in the HashTable. This worst-case is not possible since the key is unique prompting for no collisions to occur, the amortised average case will be later discussed (Tallapalli, 2019). The final search method ‘getPostsWithText’ utilizes a TreeMap which has a Big O worst-case time complexity iterating over of O(log(z)) where z is the number of elements per key. The inner loop, calls the Boyer Moore Algorithm which runs in O(n + m) where n is the number of comparisons with characters, and m for preprocessing. The inner loop has a worst case time complexity of O(n + m), thus this method has a worst case time-complexity of O((n +m) • log(z)).

Expected Worst Case Time Complexity Differences:

The search ‘getHighestUpVote’ has a worst-case time complexity of O(n^2) where n is the number of FeedItems in the HashTable. But, the amortised expected-case complexity would be O(n) which is linear, in comparison to its worst case is quadratic. This is subsequent to the implementation of the HashTable which has a key and element (O(1) et al., 2019). Within the for loop, when searching the HashTable for whether the maximum upvote has been inputted exists, the worst case derives from the possibility of hash collisions between hash keys, however this table is guarantees as unique key, since the post ‘id’ is unique for each post. Thus, this inner loop operation runs in time complexity of O(1) constant time and the for loop is O(n) where n is the number of keys mapped with FeedItems. Thus, the amortised expected time complexity of the ‘getHighestUpVote’ is O(n) which is linear (Tallapalli, 2019). The other search methods utilize a TreeMap in which the best and worst case for operations is the same and constant, also producing a best case time complexity of O(nlog(n)).

Algorithms and Data Structures Justification:
The algorithm Boyers Moore's Algorithm was implemented to utilize its search efficiency (Www-igm.univ-mlv.fr, 2019). The basic algorithm has a runtime complexity of O(nm +s) where n and m are the number of characters in the first string and pattern, and s is the ASCII characters used. This yields a worst case time complexity of O(nm) which is quadratic. To optimize the method calls, this algorithm was modified so that the preprocessing task of filling in a bad heuristic array with the pattern was done inside the for loop and split separate from the matching process, which runs in O(m + s) (COMP3506 Lecture, 2019). This ensures a single method call to the Boyers Moore search algorithm runs in O(n) where n is the number of characters in the search string. Thus, the overall algorithms integration into a method call which is iterative ensures the preprocessing is done before the matching process and the Boyers Moore search call is done inside the loop after preprocessing, which ensures the method call is linear O(n). The Boyers Moore's algorithm implemented has a space complexity of O(m + s), where m is the number of characters in the pattern and s is the sum of the ascii characters used (Hussain, Hassan Kazmi, Khan & Mehmood, 2013). This space complexity was favoured as the algorithms memory usage is mostly done in preprocessing. Since this , this linear extra space increases proportionally to the number of comparators required, and is only accessed when a string is required to be compared to a pattern, otherwise this memory is not accessed. Tradeoffs using this algorithm incorporates the extra space in memory growing due to the number of patterns to match, for a large amount of patterns matching this can cause a significant amount of memory allocated for this method call.

The data structures of the HashMap and HashTable was used to map with key (id) and element(FeedItem). This has a space complexity of O(n) in memory where n is the number of elements in the Hashmap/Hashtable. The reason for using this data structure was the purpose of searching and inserting within it, which has a linear expected time complexity of O(1). To ensure space complexity in memory is optimised the Integer Data Type is used instead of the Long for the 'id', the 'id' is casted to an Integer which takes up less space in memory (Cpp.edu, 2019). Tradeoffs for these two data structures that need to be considered is the case of duplicate keys, this would increase the time complexity of the search and insert tasks to O(n), which in the search methods would decrease the efficiency of the method to be quadratic. The space complexity is constant linear, which was another reason for using these data structures. If the 'id' was not unique there would be a major tradeoff with efficiency with the hash collisions causing a quadratic O(n^2) worst case time complexity (Tallapalli, 2019). The TreeMap data structure was selected due to the certainty of it access and search time complexity. The TreeMap is similarly implemented to the Red-Black Tree with the search and access having static time complexity of O(log(n)), where n is the number of elements mapped. This ensures in the implementation that the time complexity if never quadratic, but will always be at least O(log(n)). The tradeoffs with this data structure is the use of an unordered map (HashMap) provides better access and search time complexity of O(n) in comparison with O(log(n)). The TreeMap also has a space complexity in memory of O(n). The data structures of Array was used to store the characters of a pattern and a search string, this has a constant access time of O(1) which ensures that the call to the Boyers Moore's Algorithm does not produce a O(n^2) when iterating over each character. This was best optimised in the implementation by initializing the character array before the loop, maintain the search method being either O(log(n)) or O(n) time complexity. The space complexity in memory is O(n) where n is the number of characters in the Array. The final data structure used was the LinkedList which has a constant O(1), worst case case time complexity in terms of insertion which is efficient in the constructor when adding the FeedItems and 'id' to the LinkedList. The tradeoff in terms of efficiency is the search which has an expected time complexity of O(n). The access time complexity doesn't influence the overall time complexity of the method since the inner loop's call to the Boyers Moore's Algorithm is O(n), the LinkedList access is irrelevant to the methods overall call time complexity is maintained as O(nlog(n)) (Bigocheatsheet.com, 2019).

Algorithms and Data Structures Potential Implementations :

A potential solution for the 'getHighestUpVote' to improve the time complexity would be to use a skip list instead of the HashMap, which in best case would improve the efficiency of the method call from amortised expected case O(n) to O(log(n)). However, there are some tradeoffs that need to be considered when using the skiplist. The skiplist has average case O(log(n)) where n is the number of elements in the list, however the skip list has a worst case of O(n), which in turn would cause the efficiency to decrease to O(n^2). This worst case is unpredictable and not impossible, hence by using a HashMap the probability of a quadratic time complexity is avoided (Lcm.csa.iisc.ernet.in, 2019). Another trade is the space complexity of the skip list which is O(nlog(n)), this is much more efficient in memory usage in comparison to the HashMap's O(n) (Bigocheatsheet.com, 2019).

The search methods 'getPostsBetweenDates', 'getPostsAfterDate' and 'getPostsWithText' all make use of the TreeMap data structure. Alternative implementation data structures for this include the use of brute force. The Brute Force algorithm in each of these searches incorporates traversing a list of elements looking for a string of n character in a string of characters, this would have a worst case time complexity of O(mn) which is quadratic. The TreeMaps's implementation offers a worst case time complexity of O((n+m)log(z)), where z is the number of traversals in the TreeMap and m and n is the matching process for the Boyers Moore Algorithm. Even though the TreeMap offers better overall computational time complexity, it also offers better access, search and insertion time complexity (O(log(n))) in comparison to the brute force method which uses Lists in O(n) access, insertion and search where n is the number of elements. A potentially more efficient solution for the search methods would be to use the KMP algorithm. The KMP algorithm, which has a worst case time complexity of O(n +m) where n and m are the number of characters in the text and pattern (COMP3506 Lecture, 2019). This has less comparisons than the Boyers Moore's Algorithm, however since the preprocessing for the 'BMA' is independent of the matching process, both algorithms overall would have a linear worst case time complexity for the search methods O(n).

An algorithm to consider implementing would be sort algorithm in the constructor, this could include a radix sort for placing the max upvotes in a stack in ascending or descending order. The radix sort uses buckets as a sorting algorithm and runs in linear time O(n +k) where n is the number of elements and k is the number of bits required for the largest element in the array (GeeksforGeeks, 2019). If this was implemented in conjunction with a stack, the pop call in the method 'getHighestUpVote' would be O(1) constant time. However, the tradeoff is the constructor's efficiency would decrease due to the enqueue in the stack and radix sort, furthermore the space complexity O(n+k) which increases the amount of memory assigned in the FeedAnalyser, this can be a problem for a larger data set input of social media posts (Bigocheatsheet.com, 2019).

References:

Baktashmotlagh, M. (2019). Lecture Week 8. Presentation, UQ Centre

Www-igm.univ-mlv.fr. (2019). *Boyer-Moore algorithm*. [online] Available at: https://www-igm.univ-mlv.fr/~lecroq/string/node14.html [Accessed 25 Sep. 2019].

Bigocheatsheet.com. (2019). *Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell*. [online] Available at: https://www.bigocheatsheet.com/ [Accessed 25 Sep. 2019].

GeeksforGeeks. (2019). *Boyer Moore Algorithm for Pattern Searching - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/ [Accessed 25 Sep. 2019].

Tallapalli, N. (2019). *order of hashmap/hashtable in BIG-O Notation*. [online] coding algorithms. Available at: https://tekmarathon.com/2012/10/05/order-of-hashmaphashtable-in-big-o-notation/ [Accessed 25 Sep. 2019].

Lcm.csa.iisc.ernet.in. (2019). *3.8 Skip Lists*. [online] Available at: http://lcm.csa.iisc.ernet.in/dsa/node52.html [Accessed 25 Sep. 2019].

Hussain, I., Hassan Kazmi, S., Khan, I., & Mehmood, R. (2013). *Improved-Bidirectional Exact Pattern Matching* [Ebook] (4th ed.). International Journal of Scientific & Engineering Research. Retrieved from chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://pdfs.semanticscholar.org/d7ac/f6192fa5708e5af145ec1129965a7b961781.pdf

GeeksforGeeks. (2019). *Boyer Moore Algorithm for Pattern Searching - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/ [Accessed 25 Sep. 2019].

O(1)?, C., Byers, M., M, D. and D, E. (2019). *Can hash tables really be O(1)?*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/2771368/can-hash-tables-really-be-o1 [Accessed 26 Sep. 2019].
Cpp.edu. (2019). *CS240: Data Structures & Algorithms I*. [online] Available at: https://www.cpp.edu/~ftang/courses/CS240/lectures/hashing.htm [Accessed 26 Sep. 2019].