Project Report

Rating system for an online book market place

The Rating System was implemented using the Java language. Primitive data types and structures including Integers, doubles, strings and arrays were used basically for the implementation.

Other Details

a) Main Data Structures used :
- Hash Table
- Linked List
- Queue

Algorithms used:
- Merge Sort
- Insertion Sort

b)

i) I used the **Hash table** data structure to store details of the books and vendors. This enables to search an item with nearly O(1) time. I used the **Linked List** data structure to **implement** the Hash Table and to **handle collisions**.

ii) I used **Linked List** to keep **a list of books** relevant to each vendor and **a list of vendors** relevant to each book.
At first I hoped to use the **Heap** data structure to store the vendors over a Linked list since it enables to apply **heap sort** on its elements. But I found out that **Merge sort generally outperforms heap sort**. The Merge sort algorithm could be easily applied on a linked list. But Heap sort has the advantage of being able to **sort in place.** But I gave the priority to time complexity over space complexity.

iii) I used the **queue** data structure to store the recent ratings of each item.
I used a queue over a linked list because only five ratings were required to be stored. A simple insertion sort algorithm could be applied to sort the ratings since the number of elements was quite small.

c) i) Sorting algorithm 01(similar to Insertion Sort)

```
Code                                      Cost   Times
for(int i=0;i<5;i++){                     C1     n
        int inTime=recentRates[i]/10;     C2     n-1
        if (time>inTime){                 C3     n-1
              int temp=recentRates[i];    C4     k
              recentRates[i]=time*10+rate; C5    k
              time=temp/10;               C6     k
              rate=temp%10;               C7     k
        }
}
```

$$T(n)=C_1n + C_2(n-1) + C_3(n-1) + C_4k + C_5k + C_6k + C_7k$$

But in the Rating system, n is always equal to 5. Therefore by doing a complexity analysis we get a running time of O(5).

ii) Sorting Algorithm 02 (Merge Sort)

```
Code                                                              Cost          Times

MergeSort(node headOriginal){                                     C1            T(n)
    if (headOriginal == null || headOriginal.next == null)        C2            1
      return headOriginal;                                        C3            1
    node a = headOriginal;                                        C4            1
    node b = headOriginal.next;                                   C5            1
    while ((b != null) && (b.next != null)) {                     C6            n
      headOriginal = headOriginal.next;                           C7            n-1
      b = (b.next).next;                                          C8            n-1
    }
    b = headOriginal.next;                                        C9            1
    headOriginal.next = null;                                     C10           1
    return merge(MergeSort(a), MergeSort(b));                 H(n)+2T(n/2)      1
 }

 merge(node a, node b){                                           H(n)          1
    node temp = new node();                                       K2            1
    node head1 = temp;                                            K3            1
    node c = head1;                                               K4            1
    while ((a != null) && (b != null)){                           K5            m
      if ((double)a.getData() >= (double)b.getData()){            K6            m-1
        c.next = a;                                               K7            m-1
        c = a;                                                    K8            m-1
        a = a.next;                                               K9            m-1
      }else{                                                      K10           ?
        c.next = b;                                               K11           ?
        c = b;                                                    K12           ?
        b = b.next;                                               K13           ?
      }
    }
    c.next = (a == null) ? b : a;                                 K14           1
    return head1.next;                                            K15           1
 }
```

The above table gives the following result:

$$T(n) = 2\,T(\tfrac{n}{2}) + H(n)$$

Here H(n) is varies according to best case worst case and average case. But This can be analyzed to get a worst case and average running time complexity of O(nlogn)

d) I faced several problems when implementing the Rating System.

- The main issue I faced was to calculate the value of **‘k’** and **‘n’** separately for **each user** since they were needed to calculate the overall aggregate rating using the following equation.

$$\boldsymbol{Overall\ Aggregate\ Rating} = \frac{\sum_{j=1}^{t} w_j\left(\sum_{i=1}^{1} r_i\right)}{\sum_{j=1}^{t} w_j K_j}$$

```
r = user rating
n = total number of ratings by the user
k = number of times the user has rated the given product/vendor
w = weight of the user rating calculated using $w = 2 - \frac{1}{n}$
t = number of ratings the given product/ vendor has
```

  I overcame this problem by keeping two separate lists; one to store all the users and another to store only the users who have rated the considered item. The first one was also used to extract the weight of the users and the second one was also used to calculate the total ratings and the number of ratings by a user to the product/vendor considered.

- Another difficulty I faced was to sort the vendors relevant to each book to output the top rated vendors. I appended the merge sort algorithm as a method to the Linked list data structure in order to overcome this issue.

- I used exception handling techniques provided by java to overcome issues related to file input streams and object references. The most common errors that popped up were null pointer exceptions while inserting data and searching data. I included an if-else statement to check for null values in each such instance to overcome runtime errors.

e) Several improvements can be added to the system in the future.
    - An option can be added to rate a product while viewing the rates. This may include the feature to add details about a book or a vendor which is currently unavailable in the system. This feature can be extended so that online ratings and user ratings are maintained separately.
    - A feature to comment about a vendor or a product would be useful. It is also possible to grade users so that only a few highly graded users with a good reputation can comment on a product or a vendor.
    - The program can be extended to output the results into a file.
    - Since the host of the Rating System is a market place, an option for the users to request for a book can be added. But this should be carefully taken care of since this might change the initial intention of the system.

Name – Dodangoda D.A.P.P.
Index Number – 110141R
CS2022_MiniProject