# ASSIGNEMENT 2:

# IFT 2015: Data Structures

Name: Gbian Bio Samir

Identifier: 20250793

1. Auto evaluation

   The code works perfectly.

2. Time complexity analysis

   The code used to treat the specified transactions is below the transaction name.

   Command: PRESCRIPTION

```
// Complexity : O(m(n log n))
  private void prescriptionManager(List<List<String>> drugs, String writingFile){
     drugs.remove(0);
     int last = drugs.size() - 1;
     drugs.remove(last);
     List<Prescription> prescriptions = convertToPrescription(drugs); //O(m)
     stock.checkPrescriptions(prescriptions, currentDate);  //O(m(n log n))
     List<String> listPrescriptions = stock.getPrescriptions();
     listPrescriptions.add(0, "PRESCRIPTION " + i);
     Writer.write(writingFile, listPrescriptions);
     stock.emptyPrescriptions();
     ++i;
  }
```

```
// Complexity : O(m(n log n))
   public void checkPrescriptions(List<Prescription> prescriptions, String date){
      LocalDate currentDate = LocalDate.parse(date);  //O(1)
      prescriptions.forEach(prescription -> {    //O(m)
         String prescriptionName = prescription.getName();  //O(1)
         LocalDate minDate = currentDate.plusDays(prescription.getQuantity()); //O(1)
         if (stock.containsKey(prescriptionName)){
            if (validatePrescription(prescription, minDate)){  //O(n log n)
               prescription.setStatus("OK");
               prescriptionStringList.add(prescription.parseString());  //O(1)
            } else {
               updateOrder(prescription); //O(log k)
            }
         } else {
            updateOrder(prescription); //O(log k)
         }
      });
   }

// Complexity : O(n log n)
   public boolean validatePrescription(Prescription prescription, LocalDate minDate){
      String prescriptionName = prescription.getName();  //O(1)
      TreeMap<String, Medication> treeMapList = stock.get(prescriptionName); //O(logn)
      SortedMap<String, Medication> subMap = treeMapList.tailMap(minDate.toString());
//O(log n)
      return subMap.entrySet().stream().anyMatch(entry -> {   //O(n)
         Medication medication = entry.getValue();  //O(1)
         int newQuantity = Integer.parseInt(medication.getQuantity()) -
prescription.getQuantity();  //O(1)
         if (newQuantity > 0){   //O(1)
            medication.setQuantity(String.valueOf(newQuantity)); //O(1)
            return true;   //O(1)
         } else if (newQuantity == 0) {    //O(1)
            removeFromStock(medication);  //O(log n)
            return true;   //O(1)
         }
         return false;
      });
   }
```

```
   // Complexity : O(log n)
   public void removeFromStock(Medication medication){
      stock.get(medication.getName()).remove(medication.getExpirationDate(),
medication); //O(log n)
   }

   //Complexity : O(log k)
   private void updateOrder(Prescription prescription) {
      String prescriptionName = prescription.getName(); //O(1)
      prescription.setStatus("COMMANDE");   //O(1)
      prescriptionStringList.add(prescription.parseString());  //O(1)
      if (!orders.containsKey(prescriptionName)){  //O(log k)
         orders.put(prescriptionName
,prescription.parseDateCommandStringFormat().trim());  //O(log k)
      } else {
         String medication1 = orders.get(prescriptionName);  //O(log k)
         String[] splittedMedication = medication1.split(" ");  //O(1)
         splittedMedication[1] = String.valueOf(Integer.parseInt(splittedMedication[1]) +
prescription.getQuantity()); //O(1)
         String newMedication = String.join(" ", splittedMedication);  //O(1)
         orders.put(prescriptionName, newMedication.trim()); //O(log k)
      }
   }
```

Globally, the PRESCRIPTION transaction is **O(m*n log n)** the worst case.

Command: APPROV

```
// Complexity : O(n log n)
   private void supplyManager(List<List<String>> drugs, String writingFile){
      drugs.remove(0);
      int last = drugs.size() - 1;
      drugs.remove(last);
      List<Medication> drugsToAdd = convertToDrug(drugs);
      stock.add(drugsToAdd); //O(n log n)
      List<String> text = new ArrayList<>();
      text.add("APPROV OK");
      Writer.write(writingFile, text);
   }

// Complexity : O(n log n)
   public void add(List<Medication> medications){
      medications.forEach(medication -> {   //O(n)
         if (stock.containsKey(medication.getName())){ //O(log n)
            updateQuantity(medication);   //O(log n)
         } else {
            TreeMap<String, Medication> medicationTree = new TreeMap<>();
            medicationTree.put(medication.getExpirationDate(), medication);
            stock.put(medication.getName(), medicationTree);
         }
      });
   }

// Complexity : O(log n)
   public void updateQuantity(Medication newMedication){
      TreeMap<String, Medication> tree = stock.get(newMedication.getName()); //O(log
n)
      if (tree.containsKey(newMedication.getExpirationDate())){      //O(log n)
         Medication medication = tree.get(newMedication.getExpirationDate()); //O(log n)
         int newQuantity = Integer.parseInt(medication.getQuantity()) +
Integer.parseInt(newMedication.getQuantity()); //O(1)
         medication.setQuantity(String.valueOf(newQuantity));    //O(1)
      } else {
         tree.put(newMedication.getExpirationDate(), newMedication); //O(log n)
         stock.put(newMedication.getName(), tree);  //O(log n)
      }
   }
```

Globally, the APPROV transaction is **O(n log n)** the worst case.

Command: DATE

```
// Complexity : O(n)
   private void dateManager(List<List<String>> dates, String writingFile){
      String date = dates.get(0).get(1);
      if(DateTools.isValid(date)){
         this.currentDate = date;
         List<String> orders = stock.getOrders(); //O(n)
         if (orders.size() > 0){
            orders.add(0,date + " COMMANDES :");
            Writer.write(writingFile, orders);
            stock.emptyOrder();
         } else {
            orders.add(date + " OK");
            Writer.write(writingFile, orders);
         }
      }
   }

   // Complexity : O(n)
   public List<String> getOrders(){
      return new ArrayList<>(orders.values());
   }

   // Complexity : O(1)
   public void emptyOrder() {
      orders.clear();
   }
```

Globally, the DATE transaction is **O(n)** the worst case.