# **COMP3506/7505 Project**

## Due: 23rd September 2022

## Report Template

|  |  |  |
| --- | --- | --- |
|  | Full Name | Student ID |
| Details | **Jamie Katsamatsas** | **s4674720** |

# Overview

This document is the ***mandatory*** template which must be used to submit the report section of your project.

This template is automatically synced with Gradescope to identify the location of each section of the report; therefore, it is imperative that the overall format/layout of this document not be modified. Modification of the template **will** result in a penalty being applied.

You are permitted to make changes inside the purple boxes for each question provided however the overall size of the box cannot change. Your report should easily fit within the boxes provided however please be aware the minimum font size allowed is Arial 10pt. If you are exceeding the box size, then this may be a good indication your response is not succinct.

# Submission

Once you have completed your report this document must be exported as a pdf and uploaded to the Gradescope Report Portal for Part B of this assignment. This document **should not** be uploaded to the autograder.

# Marking

The report will be hand marked by the teaching team. Information regarding the rubrics used while marking will be made available after grades are released. While this report will indicate the relative weighting of each section of the report, should there be any discrepancy with the official assignment specification, the assignment specification shall take precedent.

# Plagiarism

The University has strict policies regarding plagiarism. Penalties for engaging in unacceptable behaviour can range from cash fines or loss of grades in a course, through to expulsion from UQ. You are required to read and understand the policies on academic integrity and plagiarism in the course profile (Section 6.1).

If you have any questions regarding acceptable level of collaboration with your peers, please see either the lecturer or your tutor for guidance. Remember that ignorance is not a defence!

# Task

You are required to complete all sections of this report in line with the programming tasks completed in the project.

## Hospital Appointment System (16 Marks)

### Hospital 1

### State the data structure used to store patients and explain why this data structure is the best for the task in hand.

A sparse array is used to store the patients, where every possible 20 min time slot in the day has a unique index. This is the best for the task at hand as it allows for each 20 min appointment time slot to map to an index allowing ease of insertion of the patients. An array has been selected since the problem only requires a limited number of appointments to be possible per day, meaning that once the array has been initialised there is no need to grow the array.

### Describe the algorithm used to order the patients. Briefly explain how it works, from where it is called (from iterator/\_\_iter\_\_ or from addPatient/add\_patient) and why it is the best algorithm in comparison with the other known algorithms.

When addPatient is called the first bit of logic checks time on the given patient to check if it lies within the allowed times. This is done by comparing the patient time string with the hardcoded allowed times for the hospital. If the time provided does not fall within “08:00” to “11:40” or “13:00” to “17:40” addPatient returns false.

If the time check is successful then the time string is converted to an index for the array by convertin the appopintment time and the first hospital time to minutes of the day and dividing the difference by 20mins.

This calculated index is then used to index into the appointments array and check if the spot contains a null or a Patient object. If the index contains a patient addPatient returns false. If the appointments array contains null at the checked index insert the patient at the index and return true.

When an iterator is constricted the current index of the iterator is set to 0 which is used to keep track of the iterator location.

hasNext will iterate through the list starting at the currentIndex saved in the iterator up to the number of possible time slots that is saved in the hospital. If hasNext finds a location in the appointments array that contains a Patient object (no null) it returns true, else false. Call to next() in the iterator will first check if hasNext() returns false, if so a NoSuchElementException() is thrown, else the patient at the currentIndex in the appointments array is returned.

1. Assuming n to be the number of patients, state the best-case (Ω) and worst-case (O) time complexity of iterator/\_\_iter\_\_ and addPatient/add\_patient with respect to n.

addPatient: the best-case and worst case time will be O(1) since all operations in addPatient are constant time:

* Check if time is valid = O(1)
* Check if the appointments array contains a Patient at the calculated index = O(1)
* Assign the patient to the appointments array = O(1)

Iterator: I believe the best and worst case time for the iterator is also O(1) as the time for the iterator won’t change with respect to n. The appointments list has been initialised to hold all the possible 20min appointments for the day, and the iterator will need to iterate over all the indexes of the appointments array (27 total 20min time slots) regardless of the number of booked appointments.

### Hospital 2

### State the data structure used to store patients and explain why this data structure is the best for the task in hand.

An array has been selected to hold the patients of the hospital 2. An array will allow storage of the patients in the order that they have been inserted into the array. If order is maintatined an array will allow for linear access of the patients throughout the day.

### Describe the algorithm used to order the patients. Briefly explain how it works, from where it is called (from iterator/\_\_iter\_\_ or from addPatient/add\_patient) and why it is the best algorithm in comparison with the other known algorithms.

When a patient is inserted into the array of appointments a check for valid time is done similar to hospital 1.

Next the number of appointments is checked against the array length. If the appointments array is full then the doubling strategy is applied to resize the array. When the array size is doubled all the elements from the original array are copied over to the new larger array before the addPatient can continue. The patient is then added to into the next available empty slot of the array so that the array is filled up from the 0th index up to n (number of appointments).

Once the patient is inserted at the next available slot in the array, compare the patient with the patient to its left, if the patient to the left is larger you swap the indexes of the patients. You continue to compare and swap the patients until you find a patient to the left that has a time either smaller or equal to the patient to the right. This method will maintain order of the patients in the array and will ensure that if a patient books for a time slot that has already been taken priority will be given to the patient already in the system.

Since the appointments array is filled from the beginning to the end and the order of patients is maintained the iterator will simply start iterating at the beginning of the array and stop once it has iterated over the number of patients booked and finds a null.

hasNext will check if the currentIndex of the iterator is less than the numAppointments booked in the system and return true if this is the case and false otherwise.

Next() will check the return value of hasNext, if its false then NoSuchElementException is thrown otherwise the appointments[currentIndex] value is returned and currentIndex is incremented.

An array implementation is better in this scenario than a linked list since it is simpler to implement than a linked list. A hash table is not suitable since it will provide us a slower iterator than the array method we have implemented.

1. Assuming n to be the number of patients, state the best-case (Ω) and worst-case (O) time complexity of iterator/\_\_iter\_\_ and addPatient/add\_patient with respect to n.

The best-case and worst case time complexity of addPatient and iterator is O(n).

addPatient:

* Valid time check is O(1)
* Check if the array is large enough is O(1)
* Amortised growth of the doubling strategy for array resizing is O(1)
* Insertion of the patient at the next available slot is O(1)
* Comparing and swapping the new Patient with the Patients to its left is O(n)

Therefore the best and worst case time complexity is O(n) since the addPatient will complete the same tasks on every call.

Iterator:

* hasNext(): check if the currentIndex < number of appointments is O(1)
* next(): returning the next patient in the array if hasNext returns true is O(1)

The iterator best and worst case time complexity is O(n) as it will have to perform the above tasks regardless of n.

### Hospital 3

### State the data structure used to store patients and explain why this data structure is the best for the task in hand.

A doubly linked list is selected for hospital 3. The task requires insertion of O(1) and this is enabled by allowing insertion of the patient at the end of the linked list. Linked list has been selected over an array as O(1) insertion was a requirement and although this could be technically possible in an array by using amortised doubling array growth but this is less efficient than using a linked list.

### Describe the algorithm used to order the patients. Briefly explain how it works, from where it is called (from iterator/\_\_iter\_\_ or from addPatient/add\_patient) and why it is the best algorithm in comparison with the other known algorithms.

When a patient is added to the system the time is checked for validity similar to hospital 2.

A new node containing the patient data is created for the doubly linked list. And the patient is then inserted into the linked list by adding the new node to the tail.

When patients are inserted into the linked list they are added to the end meaning that the linked list is not sorted and must then be sorted in the iterator.

When an iterator is created the first thing to occur is mergeSort is called on the doubly linked list containing the patients. Merge sort is selected as it is a stable sort and will maintain the order of the patients that were inserted with the same appointment times.

Merge sort then works by splitting the linked list by the middle node recursively until a linked list containing a single node is reached (base case). Then once base case has been reached all the linked lists of size 1 are merged with a neighbouring linked list of size 1 to make a linked list of size 2. This merging of sorted linked lists continues until you end up with a linked list that is the same size as the original one but now all the elements are sorted.

In the iterator hasNext() will keep track of the current node and check if the current node in the linked list is a null. If the current node is not null, true is returned, otherwise false.

Next() will first check the return type of hasNext and if false is returned then the current node of the iterator is set to be next node and the patient of the node before the newly set current node is returned.

A doubly linked list that is sorted by merge sort is the best algorithm as it allows for O(1) insertion with minimal memory used in comparison with an array implementation. An array implementation would result in more memory used as the array would be doubling in size when it is full. Additionally when considering insertion a linked list implementation results in simply O(1) insertion as all you have to do is insert the new patient to the end. Where in an array implementation you would have amortised O(1) insertion when considering doubling the array size on growth, this could be considered slightly less efficient than the O(1) insertion of the linked list. A hash table implementation would result in much more memory and a longer iterator than the linked list implementation.

1. Assuming n to be the number of patients, state the best-case (Ω) and worst-case (O) time complexity of iterator/\_\_iter\_\_ and addPatient/add\_patient with respect to n.

For addPatient the best and worst case time complexity is O(1)

* Check of valid time is O(1)
* Add patient to end of doubly linked list is O(1)

Since the above task are always executed regardless of n addPatient is O(1)

For iterator the best and worst case time complexity is O(nlogn)

* When an iterator is created merge sort is called on the linked list O(nlogn)
* hasNext checks if the current node is null which is O(1)
* next() returns the next node which is O(1)

Since the above tasks are always executed regardless of n, this makes the iterator worst and best case times run in nlogn.

## Login System (14 Marks)

### What is the advantage of storing a hash code of the password instead of the plain text password?

In a real login system the passwords would be required to be hashed as a security measure. If the system is compromised and an attacker gains access to the data stored the would only have the emails of the users and not the passwords if they are stored as hashes.

### Do we need to store the email in the hash table? If your answer is yes, explain why it is necessary. If your answer is no, explain how you use the email (if you use it at all).

Yes it is necessary to store the email in the hashtable. The email is required for when the hash table is resized. Once a hash table is resized the hashes of all the emails in the original hash table are recalculated in order to find the new locations of the existing emails in the now larger hash table. If you did not store the email in the hash table when you resized the hash table all those original emails would then be in the incorrect position and you would have no way of recalculating the hash to find the new spots as hashing functions are one way.

### Which of the following is an example of a collision? Explain your answer for both cases.

(a) Two users have the same email hash

(b) Two users have the same password hash

1. If two users have the same email hash this will cause a collision as the email hash is used to find the location of the (email, hashed password) tuple in the hash table. Two users having the same email hash will mean that when the second email is inserted into the hash table it will first land on a spot that is already occupied by the original email. Then you will be required to linear probe in order to find the next available slot to put the second email that had the collision.
2. Two users having the same password hash will not cause a collision as the password hash is not used to find the location of the (email, password hash) tuple in the hash table. However, it may cause some security issues as this means that the users could log into each others accounts as their passwords create the same hash and they could use the other persons email with their password and the system would accept this.

### What is the type of hash code function being used? Explain why it is suitable for use in this hash table.

## Tree of Symptoms (10 Marks)

### What is the type of the restructured tree?

Binary search tree. All symptoms of less severity are to the left of a node and all symptoms of higher severity are to the right of a node. This is the same condition as a binary search tree.

### For any given binary tree, is the reconstructed tree **balanced**? If so, explain why. If not, give a counter-example.

No the reconstructed tree is not balanced. A counter example is imagine you are given a tree that contains the following nodes [1, 2, 3, 4, 5, 6, 7, 8, 9]. You can then reconstruct the tree on the node with severity 1 and when you reconstruct the tree in order to satisfy the conditions you must have all nodes with higher severity to the right. This means that evert other node will all be to the right with a minimum height of logn = 3 and the right will have a height of 1 which is unbalanced.

### For any given binary tree, is the reconstructed tree **unique**? If so, explain why. If not, give a counter-example.

Yes the reconstructed tree is unique. One of the conditions is that “Each symptom has a unique severity level” this means that since the reconstructed tree is a binary search tree depending on your method for reconstructing the binary search tree the tree height may be slightly different but it will result in the same nodes placed in the same order as the binary tree must maintain its binary search properties.

**END OF REPORT**

ALIGNMENT TEST BOX

DO NOT EDIT