COMP3506/7505 Project – Part A

Due: 13th September 2021 @ 4:00 PM AEST Version: 1.0

Report Template

	Full Name	Student ID
Details	**Benjamin Israel**	**45372995**

1. 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.

2. Submission

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

3. 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.

4. 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!

5. Task

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

Departure Display (20 Marks)

1. Provide a screenshot of the main sorting function for the partially sorted list (*DisplayPartiallySorted*).

```
class DisplayPartiallySorted(DisplayPartiallySortedBase):
    """
        Partial sort method, combines seld.schedule and self.extra_planes
        and sorts the resulting list.

        return: sorted arrray of planes
    """
    def sort(self):

        unsortedplanes = self.schedule + self.extra_planes

        for i in range(1, len(unsortedplanes)):
            key = unsortedplanes[i]
            k = i-1

            while k >= 0 and key > unsortedplanes[k]:
                    unsortedplanes[k + 1] = unsortedplanes[k]
                    k -= 1
            unsortedplanes[k + 1] = key
        return unsortedplanes
```

2. Describe which sorting algorithm you have used for the unsorted list and state its time and space complexity in the best case, average case and worst case, with respect to the number of planes, **n**.

I used a quick sort algorithm for the random unsorted list. It has an average time complexity of O(n*logn), a best time complexity of O(n*logn) also and a worst time complexity of O(n^2). Its space complexity is O(n).

3. Describe the modifications you have made to take advantage of the list being partially sorted. State the complexity of your algorithm with respect to **n** and **k**, where **n** is the number of sorted planes, **k** is the number of unsorted planes.

I used an insert sort algorithm to take advantage of the partially sorted list. The overall average complexity would be O(nk) where each element in the input is no more than k places away from is sorted position. When the comparison done in insert sort is frequently true (ie the case of a partially sorted array) it takes about half as many comparisons when compared to selection sort. In the best case this would be O(n) where the list is already sorted. However, in the worst case it would be O(n^2) where the list is in reverse order.

Flight Dispatcher (30 Marks)

1. Provide a screenshot of the functions *allocateLandingSlot/allocate_landing_slot* and *emergencyLanding/emergency_landing*.

```
def allocate_landing_slot(self, current_time: str):
    """
    Allocate the landing slot to the next plane in line if it is already waiting
    or if it arrives no later than 5 minutes from the current time.
    Remove the plane that has been granted a landing slot and return its number
    Otherwise return None

    The complexity must be O(1)

    :param current_time:string, represents the current time in 24h format.
                        Example: "9:24", "15:32"
    :return: Plane number or None
    """

    if (self.is_empty()):
        return None

    topPlane = self.queue[0]
    current_timeasint = int(current_time.replace(":",""))

    if(topPlane.getTimeAsInt() - current_timeasint) <= 5:
        self.queue = self.queue[1:]
        return topPlane.plane_number

    return None
```

```
def emergency_landing(self, plane_number: str):
    """
    Find and remove a plane by its number.
    The complexity must be O(n)

    :param plane_number: string with 3 letters, followed by 4 numbers.
                         Example: "ABC1236", "ENC3455"
    :return: Plane number or None
    """
    i=0
    for plane in self.queue:
        i=i+1
        if (plane.plane_number == plane_number):
            del self.queue[i:i+1]
            return plane.plane_number
    return None
```

2. Describe the data structure you used to represent a flight dispatcher.

I used a priority queue which was ordered according the aircrafts time. Insert (addplane) enqueued an aircraft which was shifted to its appropriate place in the queue. Remove (allocateLandingSlot) removed the first plane in the queue.

3. State the best case, average case and worst case time complexities of the functions *allocateLandingSlot/allocate_landing_slot* and *emergencyLanding/emergency_landing* with respect to **n**. Briefly explain how you achieved these complexities given the data structure you have chosen.

Allcoate landing slot will always run in fixed time because it just makes a comparison with the first element and the given time. Emergency landing will run in O(n) because it first has to find the plane in the queue then delete. In the best case EmergecyLanding will run in O(1) because the first element will be the one to land.

AllocateLandingSlot:

- Best case: O(1)
- Average case: O(1)
- Worst case: O(1)

emergencyLanding:

- Best case: O(1)
- Average case: O(n)
- Worst case: O(n)

4. If addPlane/add_plane had to run in constant time, which alternate data structure could be used? How would this affect the running time of *allocateLandingSlot/allocate_landing_slot* and *emergencyLanding/emergency_landing*?

To achieve a O(1) for addPlane a stack could be used. However, in allocatingLandingSlot and emergecnyLanding you'd have to then traverse the stack to return values. This could be done in a number of ways but the best time complexity for both would probably be O(n) in this case.

END OF REPORT

ALIGNMENT TEST BOX
DO NOT EDIT