# **TASK B: APPLICATION OF LIFO-FIFO CONCEPT**

## **PROBLEM STATEMENT: AIR TRAFFIC CONTROL SYSTEM**

*You are tasked with designing an algorithm/pseudocode for air traffic control system (ATCS) for an airport to determine which plane should land next from several incoming flights. Each aircraft is assigned a priority based on two key factors: how far it is from the runway and how much fuel it has left. When a plane requests clearance to land, the ATCS calculates its priority based on these factors.*

*These requests are organized in a priority queue, where planes are arranged by priority, ensuring that higher-priority planes are cleared to land before lower-priority ones. In a priority queue, the planes are sorted based on their calculated priority. This process helps optimize runway usage while ensuring safe and efficient air traffic flow.*

*Planes facing emergencies or running low on fuel are given top priority for landing. Those airplanes without any urgent conditions are assigned lower priority and are queued accordingly. The priority queue is constantly updated as new planes request landing or conditions change, allowing the ATCS to effectively manage the dynamic nature of air traffic and prioritize flights as needed. This system ensures that critical situations are handled promptly while maintaining proper order for all flights.*

Planes are prioritised by fuel level of "low", "normal" and "high" (figure 1) then proximity to the airport.

A priority queue implemented as a binary heap is a suitable data type (Baka, 2022), using low values for high priority planes, extending towards infinity for low priority planes.

Landing requests are managed by appending a plane using min\_heap\_insert(key) followed by heap\_decrease\_key() to maintain the heap property. On landing, the root is removed using heap\_extract\_min() followed by min\_heapify() to maintain the heap property.

Heap addition/removal operations run in O(n log n) time (McDowell, 2015).

**[92]**

|  |  |
| --- | --- |
| Low Fuel: | ≤ 20% |
| Normal Fuel: | > 20 & ≤ 60% |
| High Fuel: | > 60% |

Figure 1

When the planes are ready for landing, plane A is removed from the heap since it has the highest priority (figure 2).

A diagram of a diagram

Description automatically generated

Figure 2

Once Plane A is dequeued, the selected last node becomes the root (figure 3). The heap reorganises and heapifies to maintain priority (figure 4), percolating down as needed until a new root is established (figure 5).

A diagram of a diagram

Description automatically generatedA diagram of a diagram

Description automatically generated A drawing of a diagram

Description automatically generated

Figure 3, Figure 4, Figure 5

Assuming the new plane (E) with low fuel is close to the airport, it becomes the highest priority flight.

The heap will now be violated as the element is inserted at the last position (figure 6). The heapify process rearranges the nodes to resume satisfying the heap property (figures 7 and 8), swapping nodes upwards until the sort is complete (figure 9) (Agarwal, 2022).

**[122]**

A diagram of a diagram

Description automatically generatedA diagram of a diagram

Description automatically generatedA diagram of a diagram

Description automatically generatedA diagram of a diagram

Description automatically generated

Figure 6, Figure 7, Figure 8, Figure 9

The MinHeap class implements a min-heap priority queue, where the underlying data structure is a binary heap represented as a list. This binary heap ensures that the smallest element is always at the root.

The Plane class represents an aircraft with attributes for its ID, fuel level, and distance from a destination. It has three fuel level categories (low, normal, high) and has methods for comparing planes based on their fuel levels and distances. These custom comparator methods are essential for MinHeap to order planes in the priority queue, maintaining its property. In detail this means:

Landing requests - insertion using min\_heap\_insert(): When a new plane is inserted, the heap\_decrease\_key() method uses the \_\_gt\_\_ method to move the plane up the heap until the heap property is restored.

Landing completion - heapify using min\_heapify(): When the highest priority plane is removed, the min\_heapify() method uses the \_\_lt\_\_ method to move the (now swapped) lowest priority plane down the heap until the heap property is restored.

**[165]**

The solution is partly optimised: Plane objects with custom comparators are stored directly in the heap. This is an optimisation over storing priorities as integers in the heap, avoiding the need to implement some arbitrary external mapping mechanism between planes and priorities. Considering batch insertion as an optimisation is impractical since planes arrive individually.

Should the number of planes scale, dedicating a portion of memory to the heap upon creation may be a consideration. For small numbers of planes, this optimisation is unnecessary.

One key limitation the solution overlooks is dynamic fuel decreasing, which would better reflect real-time air traffic control.

Cormen et al (2001) states Fibonacci heaps offer a better amortised time complexity of O(1), offering efficiency for larger-scale problems.

**[121]**

**[500]**

**References**

Agarwal, B. (2022). *Hands-on data structures and algorithms with Python* (3rd ed.). Packt Publishing.

Baka, B. (2017). *Python Data Structures and Algorithms.* Packt Publishing.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction To Algorithms* (3rd ed). MIT Press.

McDowell, G. L. (2015). *Cracking the Coding Interview* (6th ed.). CareerCup.