Air Traffic Control Simulation Project Report

Design, Complexity, and Memory Analysis

Prepared by: Sarosh, Ahmed and Hamza

Date: April 24, 2025

Task Distribution:

Hamza: core_functions.py + add_landing and add_takeoff in main.py

Sarosh: main.py

Ahmed: gui_functions.py

Problem Statement & Significance

Efficiently managing the landing and takeoff sequences of aircraft at busy airports is critical for ensuring safety, minimizing fuel consumption, and maximizing runway utilization. The Air Traffic Control Simulation project addresses the challenge of dynamically prioritizing flight operations under varying conditions, including emergencies, VIP flights, and low-fuel scenarios. By simulating realistic traffic patterns and resource constraints, the system aids in testing scheduling algorithms and decision-making protocols that can be applied to real-world air traffic management.

Data Structure & Algorithm Justification

The core data structure employed is the Priority Queue ADT, implemented via a Max Binary Heap. Priority queues allow efficient access to the highest-priority element, with insertion and removal in O(log n) time. This is well-suited for managing landing and takeoff queues, where aircraft priorities evolve based on fuel status, emergencies, and scheduled times. Additional use of dictionaries (hash maps) supports constant-time lookup of active flights, while lists track runways and completed or diverted flights. The combination of these ADTs ensures scalable, responsive queue management under dynamic conditions.

Time Complexity & Memory Analysis

Complexity Analysis

Notation

- n = number of heap items (per queue)

- N = total number of planes across all landing/takeoff queues

- r = number of runways

- f = number of active flights

core_functions.py

Function	Time Complexity	Space Complexity	Notes
init_runways()	O(1)	O(1)	Always creates a fixed list of 7 runways.
generate_plane	O(1)	O(1)	Constant random choices and dict construction.
calculate_landing_priority	O(1)	O(1)	Fixed number of arithmetic operations and lookups.
calculate_takeoff_priority	O(1)	O(1)	Fixed arithmetic operations.
log_event	O(1)	O(1)	Logs events.
find_runway	O(n log n)	O(n)	Scans all runways and sorts the available list.

gui_functions.py

Function	Time Complexity	Space Complexity	Notes
update_treeview	O(n)	O(1)	Deletes and reinserts n items in the Treeview.
get_priority_queue_data	O(n log n)	O(n)	Copies heap (O(n)) and removes all elements (O(n log n)).
update_info_labels	O(1)	O(1)	Loops over a fixed number of runways and heap-length queries.
update_gui_elements	O(N log N)	O(N)	Processes N planes: heap data extraction, sort (O(N log N)), and Treeview updates.
setup_gui	O(1)	O(1)	Builds a fixed set of widgets.

maxheap.py

Function	Time Complexity	Space Complexity	Notes
_Item_init	O(1)	O(1)	Tuple construction.
_Item_gt	O(1)	O(1)	Key comparison.
is_empty	O(1)	O(1)	len check.
_parent/_left/_right	O(1)	O(1)	Simple arithmetic.
_has_left/_has_right	O(1)	O(1)	Bounds checks.
_swap	O(1)	O(1)	List element swap.
_upheap	O(log n)	O(log n)	Recursion stack depth = O(log n).

_downheap	O(log n)	O(log n)	Recursion stack depth = O(log n).
_heapify	O(n)	O(1)	Bottom-up build-heap.
create_heap_priority_queue	O(1)	O(1)	Returns an empty list.
__len__	O(1)	O(1)	Wrapper for len().
add	O(log n)	O(1)	append + upheap.
max / peek_max	O(1)	O(1)	Root access.
remove_max	O(log n)	O(1)	swap, pop, and downheap.
remove	O(n)	O(1)	Linear search + heapify.
update_priority	O(n)	O(1)	Linear search + upheap/downheap.

main.py

Function	Time Complexity	Space Complexity	Notes
add_landing	O(log N)	O(1)	heap.add into a queue of size ≤ N.
add_takeoff	O(log N)	O(1)	heap.add into a queue of size ≤ N.
update_runways	O(r) → O(1)	O(1)	Loops over a constant number of runways.
update_plane_state	O(f·N) → $O(N^2)$	O(f) → O(N)	Scans active_flights; updates may call heap updates/removals.

process_landing	O(log N)	O(1)	Peeks and possibly removes from queues.
process_landing_helper	O(log N)	O(1)	find_runway + remove_max.
process_takeoff	O(log N)	O(1)	Similar to process_landing.
process_takeoff_helper	O(log N)	O(1)	find_runway + remove_max.
generate_traffic	O(log N)	O(1)	At most one landing and one takeoff add.
simulation_step	$O(N^2)$	O(N)	Combines runway updates, state updates, traffic, and queue processing.
run_simulation	$O(N^2)$	O(1)	Calls simulation_step.
start_simulation	O(1)	O(1)	GUI state changes and log_event.
stop_simulation	O(1)	O(1)	GUI state changes and log_event.
create_emergency	O(N)	O(N)	Scans active_flights and updates priority.
create_flight	O(log N)	O(1)	generate_plane (O(1)) + add (O(log N)).