A project on

**OPTIMIZED AIRLINE RESERVATION**

**SYSTEM**

**Submitted by**

Byraju Guna Vardhan(AP23110010744)

**Bachelor of Technology**

**In**

**Computer Science and Engineering**

**School of Engineering and Sciences**



Under the guidance of

Mr. PAMULA UDAY RAJU

Assistant Professor-Ad Hoc

Department of CSE

**[November, 2024]**

SRM University, AP, Neerukonda, Mangalagiri, Guntur.

Andhra Pradesh – 522 240

Department of Computer Science and Engineering

SRM University, AP



**CERTIFICATE**

*This is to certify that the Project report entitled “****0ptimized airline reservation system****” is being submitted by Byraju Guna Vardhan(****AP23110010744)*** *student of Department of Computer Science and Engineering, SRM University, AP, in partial fulfillment of the requirement for Design and Analysis of Algorithms Lab for II-B. Tech (CSE), Semester III. carried out by him during the academic year 2024-2025.*

Signature of the Lab In charge

Mr. PAMULA UDAY RAJU

Assistant Professor-Ad Hoc

Department of CSE

TABLE OF CONTENTS

**Name of the content Page.no**

List of figures i

List of tables ii

Abstract iii

* Introduction
* Module organization
* Module overview
* Module description
* Module implementation
* Performance Evaluation
* Conclusion
* References

Appendix

* Sample code

Sample screenshots

LIST OF TABLES

|  |  |  |
| --- | --- | --- |
| S. No | Table Name | Page. No |
| 1 | Module Overview | 3 |

LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| S. No | Figure Name | Page. No |
| 1 | Flowchart -Flight Schedule Display | 4 |
| 2 | Flowchart - Reservation and Booking Management | 5 |
| 3 | Sample Output 1 | 15 |
| 4 | Sample Output 2 | 16 |
| 5 Sample Output 3 17 | | | |

ABSTRACT

The Optimized Airline Reservation System Using Hash Table and Sorting Algorithms is designed to streamline the flight reservation process by offering users an efficient, user-friendly platform for selecting and booking flights based on their preferences.

This system leverages the power of data structures and algorithms to enhance the overall reservation experience. The hash table is central to efficient seat management, allowing quick lookups and updates on seat availability. This data structure reduces the complexity of the reservation process by ensuring rapid access to seat information, thereby minimizing processing time during bookings. Additionally, sorting algorithms are applied to organize flights by economy price, presenting users with the most affordable options first and facilitating budget-friendly flight selection.

The report provides an in-depth exploration of each module within the system, covering flight scheduling, passenger detail collection, seat selection, fare calculation, and seat availability reporting. These modules work together to form an organized and optimized reservation process.

The implementation showcases the practical benefits of using data structures like hash tables and sorting algorithms in real-world applications, contributing to both system efficiency and enhanced user satisfaction. Each section of the report delves into the technical details, explaining the logic and purpose behind each major component. The system demonstrates a foundation for creating scalable reservation platforms, with potential for further functional expansion.

CHAPTER 1

INTRODUCTION

* 1. Brief introduction

This report presents the development of an **Optimized Airline Reservation System** implemented using C++ programming to enhance the efficiency of flight reservations. Traditional reservation systems often struggle with slow response times, ineffective seat management, and difficulties in handling large-scale data. This project addresses these issues by utilizing advanced data structures and algorithms, specifically hash tables for seat management and sorting algorithms for arranging flight options.

The system employs a **hash table** to enable real-time seat availability checks and prevent booking errors. This structure ensures rapid lookups and updates, significantly reducing processing time and enhancing accuracy during reservations. Meanwhile, **sorting algorithms** are integrated to present flight options in ascending order of economy price, making the booking process intuitive and user-friendly.

The program’s design revolves around an adjacency matrix representation, which facilitates efficient storage, retrieval, and manipulation of data related to flights, destinations, and seat allocations. By combining computational efficiency with a user-centric approach, the system offers an optimized solution for managing reservations in a scalable and reliable manner.

This project demonstrates the power of combining hash tables and sorting techniques to streamline operations, improve response times, and deliver a seamless reservation experience. It highlights how well-designed algorithms can effectively address real-world challenges in complex applications like airline reservations.

* 1. Software Requirements

**Operating System:** Windows

**Software/IDE:** Code::Blocks or Visual Studio

**Compiler:** GCC or MSVC

**RAM:** Minimum 4GB

**Data Structures:** Hash Table, Adjacency Matrix, and Vectors

CHAPTER 2

MODULE ORGANIZATION

# **2.1. Module Overview**

The Algorithm of The Code Can Be Divided Into five Major Part:

1. Passenger Details Collection
2. Seat Management
3. Flight Scheduling and Display
4. Billing and Fare Calculation

Table 1: Module Overview

|  |  |  |
| --- | --- | --- |
| Module | Data Structures used | Algorithms Used |
| Passenger Details  Collection | String and Numeric Variables | Validation Logic |
| Seat Management | Hash Table | **\_\_** |
| Flight Scheduling and Display | Array | Sorting Algorithm  (Merge Sort) |
| Billing and Fare Calculation. | Variables for tracking | Arithmetic Operations |

# **2.2. Module Description**

Every module is explained below in terms of its primary functions and specific components.

**3.2.1 Flight Schedule Display**

* Purpose: To read and display available flight details, sorted by economy price, to present the user with budget-friendly options.
* Functionality: Uses a flight schedule() function to load flight data from a file. Flights are displayed in ascending price order using the std::sort algorithm with a lambda function.

**start**

**Read flight Data from a file**

**Sort flights by economy price**

**Display sorted flight details**

**End**

**3.2.2 Reservation and Booking Management**

* Purpose: To facilitate the reservation process, collecting passenger information and managing seat selection.
* Functionality: Prompts the user for passenger details (e.g., name, age, seat selection). Validates seat availability through the seats hash table to avoid double-booking.

Start

Gathering passenger details

Validate seat availability

Seat is available? (Yes/No)

Confirm booking Inform user, select a new

Update hash table seat, re-enter details

Display confirmation details End

**3.2.3 Passenger and Seat Data Handling**

* Purpose: To manage passenger data and ensure efficient seat allocation.
* Functionality: Uses a hash table (unordered map<int, bool>) to track seat availability. When a seat is booked, the hash table is updated, making the seat unavailable to other users.

**3.2.4 Billing**

* Purpose: To calculate the total fare based on the destination and number of passengers.
* Functionality: Adjusts the base fare according to the destination. The total fare is computed and displayed to the user for transparency in billing.

**3.2.5 Report Generation for Seat Availability**

* Purpose: To provide real-time information on available and reserved seats.
* Functionality: By iterating through the seats hash table, the system can generate a list of available seats, helping users make informed seating choices.

# **3. Module Implementation**

**Algorithm: Flight Schedule Display**

1. **Start**
2. **Initialize**:
   * Define a structure (e.g., Flight) to represent flight details:
     + Attributes: Flight number, destination, departure time, economy price, etc.
   * Create a container (e.g., vector flights) to store flight data.
3. **Load Flight Data**:
   * Call the flightSchedule() function.
     + Open the flight data file in read mode.
     + For each line in the file:
       - Parse the details and populate a Flight object.
       - Add the Flight object to the flights vector.
   * Close the file after loading all data.
4. **Sort Flight Data**:
   * Use the std::sort function with a lambda expression:
     + **Lambda Function**: Compare two flights based on their economy price.
     + Syntax: std::sort(flights.begin(), flights.end(), [](Flight a, Flight b) { return a.price < b.price; });
   * This will arrange flights in ascending order of economy price.
5. **Display Flights**:
   * Loop through the sorted flights vector.
   * For each flight, display details (e.g., flight number, destination, price) in a readable format.
6. **End**

**Algorithm: Reservation and Booking Management**

1. **Start**
2. **Initialize**:
   * Create a hash table (seatsHashTable) to store seat availability with seat numbers as keys and values indicating whether the seat is available (e.g., true or false).
3. **Prompt User**:
   * Ask the user to input:
     + **Name**
     + **Age**
     + **Desired seat number**
4. **Validate Seat Availability**:
   * Check the hash table for the selected seat:
     + **If the seat is available (seatsHashTable[seatNumber] == true)**:
       - Proceed to Step 5.
     + **Else**:
       - Inform the user that the seat is already booked.
       - Prompt the user to select a different seat.
       - Repeat Step 4.
5. **Confirm Reservation**:
   * Update the hash table to mark the selected seat as unavailable:
     + seatsHashTable[seatNumber] = false.
   * Store passenger details (e.g., name, age, seat number) in a separate data structure (e.g., a list or database).
6. **Display Confirmation**:
   * Show the user the booking confirmation with their name, seat number, and other relevant details.
7. **End**

CHAPTER 3

PERFORMANCE EVALUATION

The performance of the **Optimized Airline Reservation System** is primarily influenced by the algorithms used to manage seat reservations and sort flight options. The system relies on a **hash table** for seat management and **sorting algorithms** for flight price organization, both of which contribute to the overall efficiency of the system.

The **hash table** used for seat management ensures quick access to seat availability and reservation updates. This provides an average time complexity of **O(1)** for checking and updating seat statuses, which is efficient for real-time reservation management. However, sorting flights by price using a sorting algorithm has a time complexity of **O(n log n)**, where **n** is the number of flights. This ensures that users are presented with the most affordable options first.

**T1(n) = n log n** (Sorting flight prices)  
**T2(n) = O(1)** (Seat management via hash table)

The **overall time complexity** of the system is the sum of the time complexities of these two main operations:

**T(n) = T1(n) + T2(n)**  
**T(n) = n log n + O(1)**  
**T(n) = O(n log n)**

Regarding **memory usage**, the system uses a **hash table** for seat management, which requires **O(n)** space, where **n** is the number of seats or flights. The **sorting process** uses **O(n)** space for storing flight details. Overall, the memory requirements are efficient for moderate datasets.

**T3(n) = O(n)** (Space complexity for hash table and sorting)

In summary, the overall time complexity is dominated by the sorting algorithm, yielding a performance of **O(n log n)**. The space complexity remains efficient with **O(n)** space usage for both the hash table and sorting operations. The system performs well for typical flight reservation scenarios, though for larger datasets or more complex reservation needs, additional optimizations may be needed to handle increased data efficiently.

CHAPTER 4

CONCLUSION

This report outlines the development of the **Optimized Airline Reservation System**, which utilizes **hash tables** for efficient seat management and **sorting algorithms** to organize flights by economy price. The system provides an optimized solution for handling flight reservations, focusing on efficient seat availability tracking and cost-effective flight selection for users.

The use of a **hash table** ensures fast seat reservation management with **O(1)** time complexity for lookups and updates, allowing the system to handle bookings in real-time with minimal delays. The sorting algorithm used for arranging flights by price ensures that the most affordable options are displayed first, with a time complexity of **O(n log n)**. These design choices result in an efficient and responsive reservation system that scales well for moderate datasets.

While the system performs efficiently for small to medium datasets, it may face limitations in terms of memory usage and performance as the number of flights and passengers increases. The **O(n log n)** time complexity for sorting, while optimal for most scenarios, could be inefficient for extremely large datasets. Future work may involve exploring more advanced data structures or algorithms to improve scalability and handle large datasets more effectively.

In conclusion, the **Optimized Airline Reservation System** demonstrates the practical application of **hash tables** and **sorting algorithms** in a real-world scenario, balancing efficiency and simplicity. The system is a functional and scalable solution for managing flight reservations and has the potential for further optimization as the requirements grow.

# APPENDIX

# SAMPLE CODE

#include <iostream>

#include <iomanip>

#include <fstream>

#include <unordered\_map>

#include <algorithm>

using namespace std;

#define SIZE 100

int reserve = 999;

struct Flight {

string name;

int code;

int price;

};

class Airlines {

public:

int to;

string name[100];

int age[100];

long long int phone\_number[100];

char gender[100];

int seat\_number[100];

int meal[100];

int choice;

int n = 0;

int bill;

unordered\_map<int, bool> seats;

void flight\_schedule();

void flight\_details();

void get\_details(int n);

void printdetails(int n);

void billing(int n);

void print\_list(int n);

void print\_report(int n);

void initializeSeats();

};

class Menu : public Airlines {

public:

void displayMenu();

};

void Airlines::initializeSeats() {

for (int i = 1; i <= SIZE; ++i) {

seats[i] = true;

}

}

void Airlines::flight\_schedule() {

ifstream file("Flight\_data.txt");

string line;

if (file.is\_open()) {

while (getline(file, line)) {

cout << line << endl;

}

file.close();

} else {

cout << "Error: Unable to open flight data file." << endl;

}

}

void Airlines::flight\_details() {

cout << "Available Flights (sorted by economy price):" << endl;

Flight flights[] = {

{"IndiGo", 1, 1000},

{"Air India", 2, 1250},

{"Go First", 3, 1100},

{"SpiceJet", 4, 950},

{"Vistara", 5, 1300}

};

sort(begin(flights), end(flights), [](Flight a, Flight b) {

return a.price < b.price;

});

for (const auto& flight : flights) {

cout << "\t" << flight.name << " ------ [" << flight.code << "] - Price: " << flight.price << endl;

}

cout << "Enter your flight choice: ";

cin >> choice;

if (choice < 1 || choice > 5) {

cout << "Error: Invalid choice. Please enter a valid choice." << endl;

}

}

void Airlines::get\_details(int n) {

cout << "Enter destination preference (1-3): ";

cout << "\n1. Ganavaram to Hyderabad\n2. Ganavaram to Renigunta\n3. Ganavaram to Chennai" << endl;

cout << "Enter Destination Choice: ";

cin >> to;

cout << "Enter passenger details: " << endl;

for (int i = 1; i <= n; i++) {

cout << "Passenger " << i << " name: ";

cin >> name[i];

cout << "Passenger " << i << " age: ";

cin >> age[i];

cout << "Passenger " << i << " phone number: ";

cin >> phone\_number[i];

cout << "Passenger " << i << " gender (M/F): ";

cin >> gender[i];

do {

cout << "Enter seat number (1-100): ";

cin >> seat\_number[i];

if (!seats[seat\_number[i]]) {

cout << "Seat taken. Choose another." << endl;

}

} while (!seats[seat\_number[i]]);

seats[seat\_number[i]] = false;

cout << "Meal preference (1=Veg, 2=Non-Veg, 3=No meal): ";

cin >> meal[i];

}

}

void Airlines::billing(int n) {

int base\_price = 1000;

if (to == 2) base\_price += 200;

if (to == 3) base\_price += 300;

bill = n \* base\_price;

cout << "Total Bill: " << bill << endl;

}

void Airlines::printdetails(int n) {

cout << "\nPassenger Details:" << endl;

for (int i = 1; i <= n; i++) {

cout << "Passenger " << i << " - Name: " << name[i]

<< ", Age: " << age[i]

<< ", Phone: " << phone\_number[i]

<< ", Gender: " << gender[i]

<< ", Seat No: " << seat\_number[i]

<< ", Meal: " << (meal[i] == 1 ? "Veg" : meal[i] == 2 ? "Non-Veg" : "No meal") << endl;

}

}

void Airlines::print\_list(int n) {

cout << "\nList of Reserved Seats:" << endl;

for (int i = 1; i <= n; i++) {

cout << "Passenger " << i << " - Seat No: " << seat\_number[i] << endl;

}

}

void Airlines::print\_report(int n) {

cout << "\nSeat Availability Report:" << endl;

for (int i = 1; i <= SIZE; i++) {

if (seats[i]) {

cout << "Seat " << i << " is available." << endl;

}

}

}

void Menu::displayMenu() {

int menu\_option;

do {

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* AIRLINE RESERVATION SYSTEM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "Options:\n";

cout << "1. Flight Schedule\n";

cout << "2. Make Reservation\n";

cout << "3. Print Reserved Seats\n";

cout << "4. Seat Availability Report\n";

cout << "5. Quit\n";

cout << "Enter your option: ";

cin >> menu\_option;

switch (menu\_option) {

case 1:

flight\_schedule();

break;

case 2:

flight\_details();

cout << "Enter Number of Passengers: ";

cin >> n;

get\_details(n);

printdetails(n);

billing(n);

break;

case 3:

print\_list(n);

break;

case 4:

print\_report(n);

break;

case 5:

cout << "Exiting system. Thank you!" << endl;

break;

default:

cout << "Invalid choice, try again." << endl;

break;

}

} while (menu\_option != 5);

}

int main() {

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WELCOME TO GANAVARAM INTERNATIONAL AIRPORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

Menu m;

m.initializeSeats();

m.displayMenu();

return 0;

}

SAMPLE OUTPUT

OUTPUT 1:

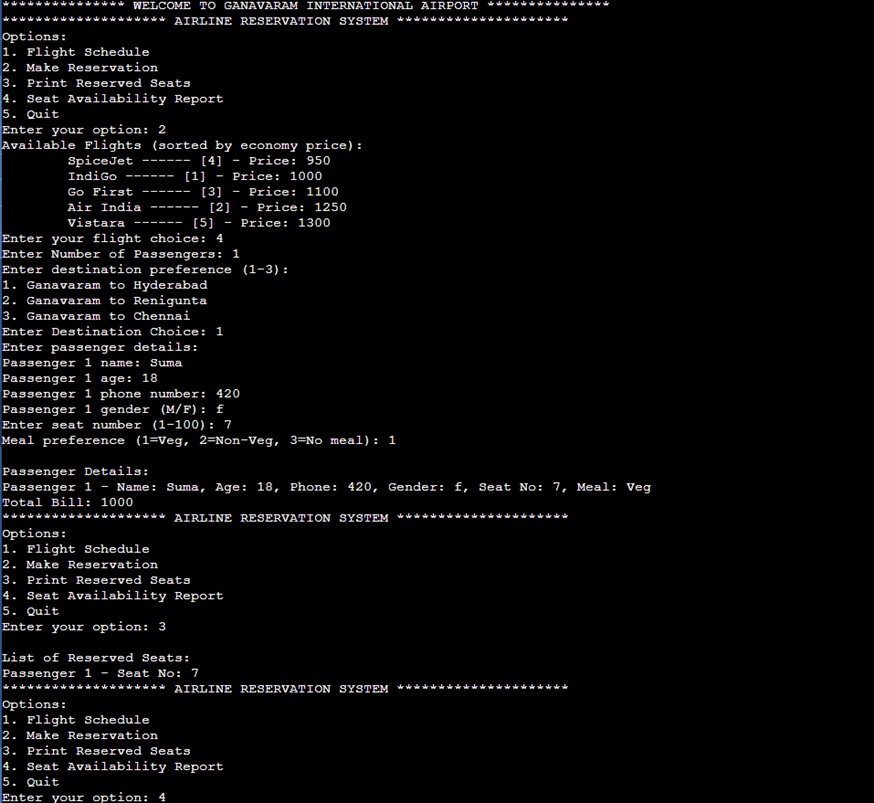


Fig.4 Sample output 1

OUTPUT 2:

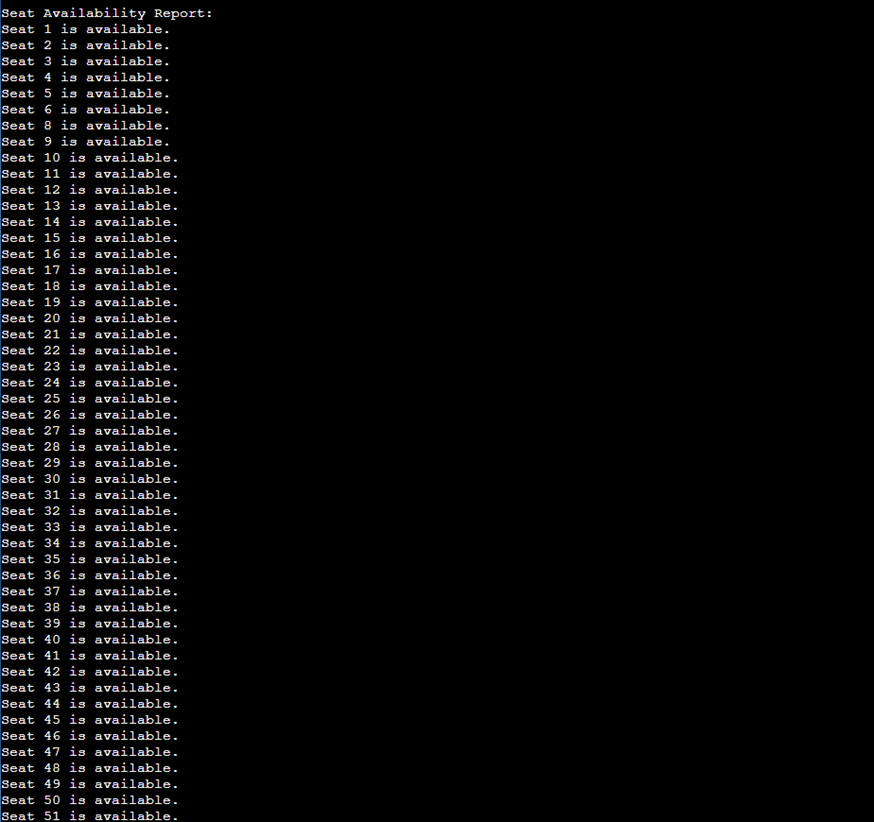


Fig.3 Sample Output 2

Output 3:

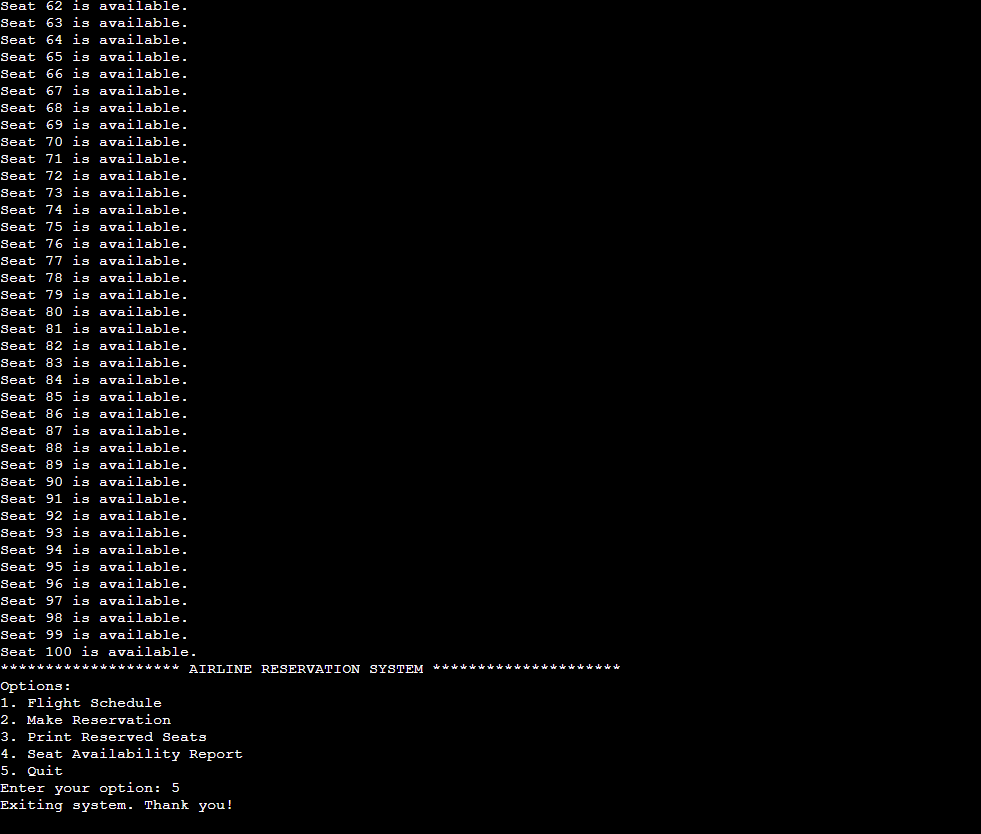


Fig.4 Sample Output 3