

Technical Report: Final Project

EECE 2560: Fundamentals of Engineering Algorithms

Taina Nieves
Department of Electrical and Computer Engineering
Northeastern University
`nieves.ta@northeastern.edu`

December 4, 2024

Contents

1	Project Scope	2
2	Project Plan	2
2.1	Timeline	2
2.2	Milestones	2
3	Team Roles	3
4	Methodology	3
4.1	Pseudocode and Complexity Analysis	3
4.2	Data Collection and Preprocessing	6
5	Results	6
6	Discussion	6
7	Conclusion	7
8	References	7
A	Appendix A: Code	7
B	Appendix B: Additional Figures	11

1 Project Scope

The aim of this project is to design and develop a Concert Ticket Reservation System. that allows the user to choose from the available seats and buy whichever one they desire. Users are also able to cancel tickets. The system will provide the user with an easy way to book and cancel ticket.

The project's main objectives are:

- To give customers a reliable form of booking tickets.
- To allow customers to cancel their tickets easily.
- To show real-time data of available seating.

The expected outcomes include a functioning web-based system, proper documentation, and a final project report.

2 Project Plan

2.1 Timeline

The overall timeline for the project is divided into phases:

- **Week 1 (October 7 - October 13):** Define project scope, establish team roles, set up the project repository, and outline skills/tools.
- **Week 2 (October 14 - October 20):** Begin development and start coding basic system functionalities.
- **Week 3 (October 21 - October 27):** Continue coding, work on backend integration, and start writing technical documentation.
- **Week 4 (October 28 - November 3):** Complete the backend and integrate frontend elements. .
- **Week 5 (November 4 - November 10):** Finalize the system, conduct testing, and continue with the report. Begin PowerPoint presentation
- **Week 6 and 7 (November 11 - November 22):** Revise and finalize the technical report and the PowerPoint presentation.
- **Week 9 (December 2 - December 4):** Final presentation, report submission, and project closure.

2.2 Milestones

Key milestones include:

- Project Scope and Plan (October 7).
- GitHub Repository Setup and Initial Development (October 7).

- Backend Completion (December 3).
- Final System Testing and Report Draft (December 3).
- Final Presentation and Report Submission (December 4).

3 Team Roles

- **Ester:** Responsible for frontend design and user interface development.
- **Taina:** Backend developer, focusing on the database and server-side logic.

4 Methodology

4.1 Pseudocode and Complexity Analysis

Algorithm 1 addSeat(rowNum, seatNum, price)

```

1: procedure ADDSEAT(rowNum, seatNum, price)
2:   Create a new node with given parameters.
3:   if tail is NULL then
4:     Set head and tail to the new node.
5:   else
6:     Append the new node to the tail of the list.
7:     Update tail to the new node.
8:   end if
9: end procedure

```

This function creates a new seat and appends it to the end of the row. It handles both the case of an empty list and the general case with existing seats. The Time Complexity is $O(1)$ and the Space Complexity is $O(1)$ as it holds a constant space for a new node.

Algorithm 2 removeSeat(seatNum)

```
1: procedure REMOVESEAT(seatNum)
2:   Initialize temp to the head of the list.
3:   while temp is not NULL and temp.seatNum  $\neq$  seatNum do
4:     Move temp to temp.next.
5:   end while
6:   if temp is not NULL then
7:     Remove temp by updating pointers.
8:   end if
9: end procedure
```

The function searches for a seat with the specified number and removes it by adjusting the pointers. The Time Complexity is $O(n)$ as in the worst case, it will iterate through the entire list and the Space Complexity is $O(1)$.

Algorithm 3 quickSort(head, tail)

```
1: procedure QUICKSORT(head, tail)
2:   if head  $\neq$  tail then
3:     Split the list around a pivot node.
4:     Recursively sort sublists before and after the pivot.
5:   end if
6: end procedure
```

The algorithm recursively sorts the linked list by selecting a pivot node and rearranging nodes such that prices/ seats below the pivot are to the left and those above are to the right. The Time Complexity is $O(n \log n)$ and the Space Complexity is $O(\log n)$.

Algorithm 4 initRowSeats(rowNum, numOfSeats)

```
1: procedure INITROWSEATS(rowNum, numOfSeats)
2:   for each seat from 1 to numOfSeats do
3:     Generate a random price.
4:     Call addSeat to add the seat.
5:   end for
6: end procedure
```

The Time Complexity is $O(n)$ and the Space Complexity is $O(1)$.

Algorithm 5 createRows(numOfSeats)

```
1: procedure CREATEROWS(numOfSeats)
2:   for each row from 1 to numOfRows do
3:     Initialize a new row.
4:     Call initRowSeats(rowNum, numOfSeats) to populate seats.
5:     if this is not the first row then
6:       Link the previous row's head to the current row's head
7:     end if
8:   end for
9: end procedure
```

This uses the multilevel links to create the rows. The Time Complexity is $O(n^2)$ since the outer loop iterates and sets row numbers and the inner loop initializes seats in the row. The Space Complexity is $O(n)$ since it is storing rows and seats.

Algorithm 6 printRowSeats(rowNum)

```
1: procedure PRINTROWSEATS(rowNum)
2:   Goes to the specified row.
3:   Traverse the list of seats in the row.
4:   for each seat in the row do
5:     Print details of the seat (row, seat number, price, etc.).
6:   end for
7: end procedure
```

The Time Complexity is $O(n)$, since it traverses once through the row. The Space Complexity is $O(1)$ as no additional data structures are used.

Algorithm 7 insertSortedSeat(rowNum, seatNum, price)

```
1: procedure INSERTSORTEDSEAT(rowNum, seatNum, price)
2:   Create a new seat node.
3:   if the row is empty or new seat is smallest then
4:     Insert the seat at the beginning.
5:   else
6:     Traverse the list to find the correct position.
7:     Insert the new seat in its sorted position.
8:   end if
9: end procedure
```

The Time Complexity is $O(n)$ since it traverses through the row to find the specific position. The Space Complexity is $O(1)$ as there is a constant space for the new seat.

Algorithm 8 removeRowSeat(rowNum, seatNum)

```
1: procedure REMOVE_ROW_SEAT(rowNum, seatNum)
2:   if this list is empty then
3:     print no seats to remove.
4:   end if
5:   Go to the specified row.
6:   Traverse the row to find the seat with the matching number.
7:   if seat is found then
8:     Remove the seat and update pointers.
9:   end if
10: end procedure
```

The Time Complexity is $O(1)$ and the Space Complexity is $O(1)$.

The project uses efficient algorithms for seat management and sorting. While most operations are efficient and have a time complexity of $O(1)$ or $O(n)$, the worst case is $O(n^2)$ (at initialization) and the average case is $O(n)$ for the user.

4.2 Data Collection and Preprocessing

For this project we did not use any external data collection. For preprocessing we needed to create a function to initialize the rows and seats before the user can make use of the interface. The number of rows and number of seats per roll can be changed depending on the venue.

5 Results

We tested our system with various size of venues, changing the number of rows from 2 to 50 rows, as well as varying the number of seats from 2 to 50. In all cases, we tested that each function performed its assigned operation and that the system could handle bigger venues without issues. We found that all the function worked properly and that our reservation ticket was well implemented. Refer to Appendix 2 for screenshots of results

6 Discussion

The result of the project was an effective implementation of the main objectives, such as managing seat reservations and cancellations, maintaining seat attributes, and sorting seats by prices and seat numbers within rows. The use of linked lists allowed for flexible data management. The Quick Sort algorithm successfully ordered seats by price and seat number, which mirrors real-world scenarios. The code also prevented invalid operations, such as booking an unavailable seat. However, the multilevel linked lists introduced complexities and challenges, which required special attention to row connections. We did not

implement a map or a VIP attribute to the tickets, as were stated in the original scope. Overall, the system met its intended goals, providing a user-friendly solution.

7 Conclusion

Throughout the time spent on the assignment, we were able to find the advantage of using Doubly Linked Lists. The main one is the ability to easily traverse forward and backward when canceling a seat and adding it back to the list, as well as using quick sort for the price and seat number. Using Double and Multilevel Linked Lists allowed for efficient insertion, deletion, and traversal. Breaking the code down into classes also helped keep the code neat, and made it easy for us to troubleshoot the program.

This code has some limitations for the user as they are not able to view where their seats are and it has little flexibility for the seat maps. For future implementations, there are many ways we can build upon this code. One of which is adding a user interface and a visualization of the seat map, this includes adding further division for the tickets, such as general admission, mezzanine, etc, which can be customized to each venue. We could also add VIP options as well as the ability to sort by VIP, highest price, and best seat. We could also filter options by price, and number of seats the user is looking for.

8 References

[1] “Multilevel Linked List,” GeeksforGeeks, Aug. 06, 2021. <https://www.geeksforgeeks.org/multilevel-linked-list/>

A Appendix A: Code

addSeat

```
1 void addSeat(int rowNum, int seatNum, double price) {
2     Node* newNode = new Node(rowNum, seatNum, price);
3     if (tail == NULL) { // If the list is empty
4         head = newNode;
5         tail = newNode;
6     } else { // If there are items in the list
7         tail->next = newNode;
8         newNode->prev = tail;
9         tail = newNode;
10    }
11 }
```

removeSeat

```
1 void removeSeat(int seatNum) {
2     if (head == NULL) { // If the list is empty
3         cout << "No seats to remove!" << endl;
4         return;
5     }
6     Node* temp = head;
7     while (temp != NULL && temp->seatNum != seatNum) {
8         temp = temp->next;
9     }
10    if (temp == NULL) {
11        cout << "Seat number " << seatNum << " not found!"
12        << endl;
13        return;
14    }
15    if (temp->prev != NULL) {
16        temp->prev->next = temp->next;
17    } else {
18        head = temp->next;
19    }
20    if (temp->next != NULL) {
21        temp->next->prev = temp->prev;
22    } else {
23        tail = temp->prev;
24    }
25    delete temp;
26 }
```

sortRowPrice

```
1 void sortRowPrice(Row& row) {
2     if (row.head && row.tail) {
3         quickSort(row.head, row.tail);
4     }
5 }
```

sortRowSeatNum

```
1 void sortRowSeatNumber() {
2     if (head && tail) {
3         quickSort(head, tail, compareBySeatNum);
4     }
5 }
```


initRowSeats

```
1 void initRowSeats(int rowNum, int numOfSeats) {
2     int seatNum = 1;
3     for (int i = 0; i < numOfSeats; i++) {
4         double randPrice = 50 + (std::rand() % 201); // $50
           to $200
5         addSeat(rowNum, seatNum, randPrice);
6         seatNum++;
7     }
8 }
```

createRows

```
1 void createRows(int numOfSeats) {
2     for (int i = 0; i < numOfRows; i++) {
3         rows[i].rowNum = i + 1;
4         rows[i].initRowSeats(i + 1, numOfSeats);
5         if (i > 0) {
6             rows[i - 1].head->child = rows[i].head;
7         }
8     }
9 }
```

printRowSeats

```
1 void printRowSeats(int rowNum) {
2     if (rowNum > 0 && rowNum <= numOfRows) {
3         cout << "Printing seats for Row " << rowNum << ":"
           << endl;
4         rows[rowNum - 1].printSeats();
5     } else {
6         cout << "Invalid row number. Please choose a valid
           row." << endl;
7     }
8 }
```

insertSortedSeat

```
1 void insertSortedSeat(int rowNum, int seatNum, double price
  ) {
2     if (rowNum <= 0 || rowNum > numOfRows) {
3         cout << "Invalid row number." << endl;
4         return;
5     }
6     Row& row = rows[rowNum - 1];
7     Node* newNode = new Node(rowNum, seatNum, price);
8     if (row.head == NULL) {
9         row.head = newNode;
10        row.tail = newNode;
11        return;
12    }
13    if (seatNum < row.head->seatNum) {
14        newNode->next = row.head;
15        row.head->prev = newNode;
16        row.head = newNode;
17        return;
18    }
19    Node* temp = row.head;
20    while (temp != NULL && temp->seatNum < seatNum) {
21        temp = temp->next;
22    }
23    if (temp == NULL) {
24        newNode->prev = row.tail;
25        row.tail->next = newNode;
26        row.tail = newNode;
27        return;
28    }
29    newNode->next = temp;
30    newNode->prev = temp->prev;
31    if (temp->prev != NULL) {
32        temp->prev->next = newNode;
33    }
34    temp->prev = newNode;
35 }
```

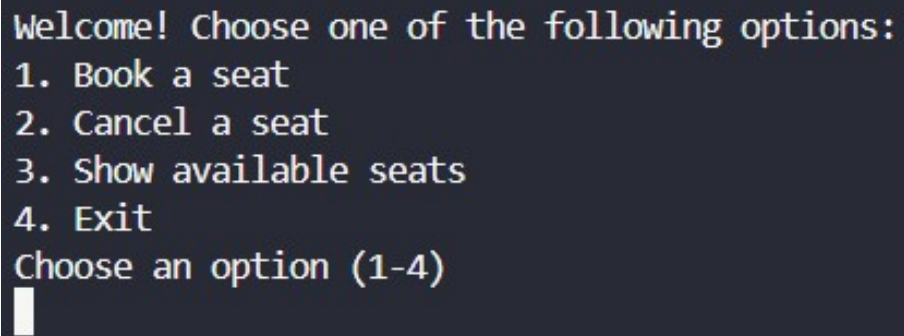
removeRowSeat

```
1 void removeRowSeat(int rowNum, int seatNum) {
2     if (rowNum > 0 && rowNum <= numOfRows) {
3         cout << "Removing seat " << seatNum << " from row "
4             << rowNum << endl;
5         rows[rowNum - 1].removeSeat(seatNum)
```

B Appendix B: Additional Figures

Figures demonstrating the system working. Example used: Booking Seat on row 12 seat 5 and then canceling it:

Step 1: Menu Prompt

A screenshot of a terminal window with a dark background and light-colored text. The text displays a welcome message and a list of four options: '1. Book a seat', '2. Cancel a seat', '3. Show available seats', and '4. Exit'. Below the list is a prompt 'Choose an option (1-4)' followed by a white cursor bar.

```
Welcome! Choose one of the following options:  
1. Book a seat  
2. Cancel a seat  
3. Show available seats  
4. Exit  
Choose an option (1-4)  
█
```

Step 2: Choosing Option 1 and selecting desired seat

Choose an option (1-4)

1

Rows 1 to 20 are available, choose your desired row: 12

Printing seats for Row 12:

Row: 12, Seat: 1, Price: \$247

Row: 12, Seat: 2, Price: \$52

Row: 12, Seat: 3, Price: \$241

Row: 12, Seat: 4, Price: \$228

Row: 12, Seat: 5, Price: \$89

Row: 12, Seat: 6, Price: \$50

Row: 12, Seat: 7, Price: \$94

Row: 12, Seat: 8, Price: \$146

Row: 12, Seat: 9, Price: \$122

Row: 12, Seat: 10, Price: \$80

Do you wish to sort seats by price? Type Y for yes and N for no: Y

Printing seats for Row 12:

Row: 12, Seat: 6, Price: \$50

Row: 12, Seat: 2, Price: \$52

Row: 12, Seat: 10, Price: \$80

Row: 12, Seat: 5, Price: \$89

Row: 12, Seat: 7, Price: \$94

Row: 12, Seat: 9, Price: \$122

Row: 12, Seat: 8, Price: \$146

Row: 12, Seat: 4, Price: \$228

Row: 12, Seat: 3, Price: \$241

Row: 12, Seat: 1, Price: \$247

Type the chosen seat from the ones available: 5

Removing seat 5 from row 12

Step 3: Printing row 12 to verify that the seat is no longer available

```
Printing seats for Row 12:  
Row: 12, Seat: 1, Price: $247  
Row: 12, Seat: 2, Price: $52  
Row: 12, Seat: 3, Price: $241  
Row: 12, Seat: 4, Price: $228  
Row: 12, Seat: 6, Price: $50  
Row: 12, Seat: 7, Price: $94  
Row: 12, Seat: 8, Price: $146  
Row: 12, Seat: 9, Price: $122  
Row: 12, Seat: 10, Price: $80
```

Step 4: Canceling the booked seat (row12 seat5)

```
Welcome! Choose one of the following options:
1. Book a seat
2. Cancel a seat
3. Show available seats
4. Exit
Choose an option (1-4)
2
Enter row of the seat you wish to cancel:
12
Enter seat number you wish to cancel:
5
Enter the price you paid for the ticket:
89
```

Step 5: Printing row 12 to verify that the seat was made available again

```
Printing seats for Row 12:
Row: 12, Seat: 1, Price: $247
Row: 12, Seat: 2, Price: $52
Row: 12, Seat: 3, Price: $241
Row: 12, Seat: 4, Price: $228
Row: 12, Seat: 5, Price: $89
Row: 12, Seat: 6, Price: $50
Row: 12, Seat: 7, Price: $94
Row: 12, Seat: 8, Price: $146
Row: 12, Seat: 9, Price: $122
Row: 12, Seat: 10, Price: $80
```

