Technical Report: Final Project
EECE 2560: Fundamentals of Engineering Algorithms

Paarth Soni, Fils Paul
Department of Electrical and Computer Engineering
Northeastern University
soni.paa@northeastern.edu, paul.fil@northeastern.edu

December 4, 2024

Contents

1 Project Scope

The objective of this project was to design and implement a **Hospital Emergency Room Management System** using **priority queues** to dynamically prioritize patients based on the following criteria:

- **Condition Severity**: Patients with more critical conditions (lower severity numbers) are prioritized over less critical cases.
- **Arrival Time**: In cases of identical severities, patients who arrived earlier are prioritized.

This system was inspired by real-world hospital workflows where patients are triaged based on the urgency of their conditions. By simulating this process in a programming environment, the project provides insights into both the strengths and challenges of implementing queue-based prioritization systems.

Objectives:

- Design a system capable of handling real-time patient admission and treatment using efficient data structures.
- Implement features such as dynamic prioritization, patient queue status display, and treated patient logs.
- Evaluate the efficiency and limitations of the implemented solution, identifying areas for potential improvement.

Expected outcomes included a functional emergency room simulation, analysis of the implemented algorithms, and insights into the challenges of real-time prioritization systems.

2 Project Plan

2.1 Timeline

The project was executed in four key phases:

- **Week 1**: Define scope, assign team roles, and set up the development environment.
- **Week 2**: Implement patient admission functionality and basic queue operations.
- **Week 3**: Develop patient treatment, logging, and queue visualization features.
- **Week 4**: Conduct final testing, analyze results, and prepare the report.

2.2 Milestones

- **Week 1**: Completed project scope and initial setup.
- **Week 2**: Implemented priority queue and admission functionality.
- **Week 3**: Developed treatment and logging components.
- **Week 4**: Completed testing and finalized the report.

3 Team Roles

- **Paarth Soni**: Algorithm development and testing.
- **Fils Paul**: Documentation and simulation design.

4 Methodology

4.1 Pseudocode and Complexity Analysis

1. Patient Admission

```
AdmitNewPatient(name, severity, checkInTime):
    Create Patient object with given details.
    Add Patient to PriorityQueue.
    Display "Patient admitted" message.
```

Complexity: $O(\log n)$ for adding a patient to the priority queue.

2. Treat Next Patient

```
TreatNextInLine():
    If PriorityQueue is empty:
        Print "No patients to treat."
    Else:
        Get top patient (highest priority).
        Remove patient from PriorityQueue.
        Add patient to TreatedPatientsLog.
        Display "Treating patient" message.
```

Complexity: $O(\log n)$ for removal from the priority queue.

3. Show Queue Status

```
showQueueStatus():
    If PriorityQueue is empty:
        Print "Queue is empty."
        Return
    Else:
        tempQueue = PriorityQueue.copy()
        Print "=== Current Queue ==="
        While tempQueue is not empty:
            patient = tempQueue.pop()
            Print patient details.
```

Complexity: $O(n \log n)$ for iterating and printing queue details.

4. Prompt for New Patients

```
promptForNewPatients():
    Print "Do you want to admit a new patient? (y/n)"
    choice = userInput()
    If choice == 'y':
        Print "Enter patient name: "
        name = userInput()
        Print "Select an injury from options:"
        For each injury in injuryList:
            Print "Option Number: Injury"
        selectedInjury = userInput()
        injury = injuryList[selectedInjury]
        currentTime = getCurrentTime()
        Call admitNewPatient(name, injury, currentTime)
        Return True
    Else:
        Return False
```

Complexity: $O(\log n + m)$, where m is the number of injury options.

5. Display Treated Patients

```
showTreatedLog():
    If TreatedPatients is empty:
        Print "No patients treated yet."
        Return
    Else:
        Print "=== Treated Patients Log ==="
        For each treatedPatient in TreatedPatients:
            Print details including waiting time.
```

Complexity: $O(p)$, where p is the number of treated patients.

6. Custom Comparator

```
CustomComparator(Patient A, Patient B):
    If A.severity == B.severity:
        Return A.arrivalTime < B.arrivalTime
    Else:
        Return A.severity < B.severity
```

Complexity: $O(1)$, constant time for comparison.

5 Results

The implemented system achieved the following:

- Successfully prioritized patients dynamically based on severity and arrival time.
- Enabled real-time updates to the queue as patients were admitted, treated, or discharged.
- Provided comprehensive logs of treated patients, including waiting times, to evaluate the system's efficiency.
- Simulated realistic ER workflows, showcasing how critical cases can delay less urgent ones.

6 Discussion

The project highlights several critical aspects of emergency room management:

- **Effectiveness of Priority Queues**: The system efficiently prioritized patients, with operations like insertion and removal completed in $O(\log n)$ time, making it scalable for larger queues.
- **Real-Time Challenges**: Continuous updates to the queue emphasized the need for advanced mechanisms to handle resource allocation and mitigate delays for non-critical cases.
- **Limitations**:
  - Simplified injury classification lacked real-world variability.
  - Assumption of perfect severity rankings may not reflect actual ER scenarios.

Future Work:

- Incorporating hospital resource constraints, such as room and staff availability.
- Implementing a graphical user interface (GUI) for better usability and visualization.
- Introducing thresholds for maximum waiting times to prevent indefinite delays for non-critical patients.

7 References

1. GeeksforGeeks. *Applications of Priority Queue.* Retrieved from https://www.geeksforgeeks.org/applications-priority-queue/.

2. BMC Systematic Reviews. *A Systematic Review of Patient Prioritization Tools in Non-Emergency Healthcare Services.* Retrieved from https://systematicreviewsjournal.biomedcentral.com/articles/10.1186/s13643-020-01482-8.

3. SpringerLink. *Queueing Problems in Emergency Departments: A Review of Practical Approaches and Research Methodologies.* Retrieved from https://link.springer.com/article/10.1007/s43069-021-00114-8.

4. SpringerLink. *Queueing for Healthcare.* Retrieved from https://link.springer.com/article/10.1007/s10916-010-9499-7.

A Appendix A: Code

Listing 1: C++ Implementation of the Hospital ER System

```
#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <ctime>
#include <map>
#include <thread>
#include <chrono>
#include <iomanip> // Required for setprecision

using namespace std;

// List of injuries and their severities
map<string, int> injuryList = {
        {"Gunshot␣Wound", 1},
        {"Heart␣Attack", 1},
        {"Stroke", 1},
        {"Severe␣Allergic␣Reaction", 1},
        {"Traumatic␣Brain␣Injury", 1},
        {"Severe␣Burn", 1},
        {"Sepsis", 1},
        {"Major␣Bleeding", 2},
        {"Pneumothorax␣(Collapsed␣Lung)", 2},
        {"Compound␣Fracture", 2},
        {"Severe␣Asthma␣Attack", 2},
        {"Severe␣Dehydration", 2},
        {"Appendicitis", 3},
        {"Kidney␣Stone", 3},
        {"Severe␣Migraine", 3},
        {"Broken␣Bone", 3},
        {"Laceration␣Requiring␣Stitches", 3},
        {"High␣Fever␣(Adult)", 4},
        {"Mild␣Concussion", 4},
        {"Sprained␣Ankle", 4},
        {"Dislocated␣Shoulder", 4},
        {"Nosebleed␣(Severe)", 4},
        {"Ear␣Infection", 5},
        {"Minor␣Cut", 5},
        {"Skin␣Rash", 5},
        {"Mild␣Food␣Poisoning", 5},
        {"Mild␣Allergic␣Reaction", 5},
        {"Cold␣or␣Flu", 5},
        {"Minor␣Burn", 5},
        {"Muscle␣Strain", 5}
};

// Struct to store all the patient info
struct ERPatient {
    string fullName;        // Patient's full name
    string injuryType;      // Type of injury
    int conditionSeverity;  // Lower = more critical
    time_t checkInTime;     // When the patient showed up (UNIX timestamp)

    // Constructor to initialize a new patient
    ERPatient(string name, string injury, int severity, time_t arrivalTime)
        : fullName(name), injuryType(injury), conditionSeverity(severity), checkInTime(arrivalTime) {}

    // Formats the arrival time into something more readable
    string readableCheckInTime() const {
        char buffer[80];
        struct tm* timeinfo = localtime(&checkInTime);
        strftime(buffer, sizeof(buffer), "%Y-%m-%d␣%H:%M:%S", timeinfo);
        return string(buffer);
    }
};

// Comparator for sorting patients by urgency and arrival time
struct CompareERPatients {
    bool operator()(const ERPatient& p1, const ERPatient& p2) const {
        if (p1.conditionSeverity == p2.conditionSeverity) {
            return p1.checkInTime > p2.checkInTime; // If severity is the same, earlier gets priority
        }
        return p1.conditionSeverity > p2.conditionSeverity; // More critical conditions come first
    }
};

// This class handles the ER queue and all patient interactions
class ERQueueHandler {
private:
    priority_queue<ERPatient, vector<ERPatient>, CompareERPatients> patientQueue; // The main patient line
    vector<pair<ERPatient, time_t>> treatedPatients; // List of treated patients with their treatment time
```

```

public:
    void admitNewPatient(const string& name, const string& injury, time_t checkIn);
    void treatNextInLine();
    void showQueueStatus() const;
    void showTreatedLog() const;
    bool promptForNewPatients();
    bool isQueueEmpty() const; // Added method to check if the queue is empty
};

// Adds a new patient to the queue
void ERQueueHandler::admitNewPatient(const string& name, const string& injury, time_t checkIn) {
    int severity = injuryList[injury];
    ERPatient newPatient(name, injury, severity, checkIn);
    patientQueue.push(newPatient);
    cout << "Added␣patient:␣" << name << "␣(Injury:␣" << injury
         << ",␣Severity:␣" << severity
         << ",␣Check-in:␣" << newPatient.readableCheckInTime() << ")." << endl;
    this_thread::sleep_for(chrono::milliseconds(1500));
}

// Treats the patient with the highest priority
void ERQueueHandler::treatNextInLine() {
    if (patientQueue.empty()) {
        cout << "Queue␣is␣empty.␣No␣one␣left␣to␣treat!" << endl;
        this_thread::sleep_for(chrono::milliseconds(1500));
        return;
    }
    ERPatient nextPatient = patientQueue.top();
    patientQueue.pop();
    time_t treatmentTime = time(nullptr); // Record the treatment time
    treatedPatients.push_back({nextPatient, treatmentTime});
    cout << "Treating␣patient:␣" << nextPatient.fullName
         << "␣(Injury:␣" << nextPatient.injuryType
         << ",␣Severity:␣" << nextPatient.conditionSeverity
         << ",␣Check-in:␣" << nextPatient.readableCheckInTime() << ")." << endl;
    this_thread::sleep_for(chrono::milliseconds(1500));
}

// Displays the current queue
void ERQueueHandler::showQueueStatus() const {
    if (patientQueue.empty()) {
        cout << "Queue␣is␣empty.␣All␣good␣here!" << endl;
        this_thread::sleep_for(chrono::milliseconds(1500));
        return;
    }

    priority_queue<ERPatient, vector<ERPatient>, CompareERPatients> tempQueue = patientQueue;
    cout << "===␣Current␣ER␣Queue␣===" << endl;
    this_thread::sleep_for(chrono::milliseconds(1500));
    while (!tempQueue.empty()) {
        ERPatient current = tempQueue.top();
        tempQueue.pop();
        cout << "Patient:␣" << current.fullName
             << ",␣Injury:␣" << current.injuryType
             << ",␣Severity:␣" << current.conditionSeverity
             << ",␣Check-in:␣" << current.readableCheckInTime() <<
                 endl;
        this_thread::sleep_for(chrono::milliseconds(1500));
    }
    cout << "========================" << endl;
    this_thread::sleep_for(chrono::milliseconds(1500));
}

// Displays the log of treated patients
void ERQueueHandler::showTreatedLog() const {
    if (treatedPatients.empty()) {
        cout << "No␣patients␣have␣been␣treated␣yet." << endl;
        this_thread::sleep_for(chrono::milliseconds(1500));
        return;
    }

    cout << "===␣Treated␣Patients␣Log␣===" << endl;
    this_thread::sleep_for(chrono::milliseconds(1500));
    for (const auto& record : treatedPatients) {
        const ERPatient& patient = record.first;
        time_t treatmentTime = record.second;
        double waitingTimeMinutes = difftime(treatmentTime, patient
            .checkInTime) / 60.0; // Convert to minutes

        cout << "Patient:␣" << patient.fullName
             << ",␣Injury:␣" << patient.injuryType
             << ",␣Severity:␣" << patient.conditionSeverity
             << ",␣Waiting␣Time:␣" << fixed << setprecision(2) <<
                 waitingTimeMinutes << "␣minutes" << endl;
        this_thread::sleep_for(chrono::milliseconds(1500));
    }
    cout << "============================" << endl;
    this_thread::sleep_for(chrono::milliseconds(1500));
}

// Prompts the user to admit new patients
bool ERQueueHandler::promptForNewPatients() {
    char choice;
    while (true) {
        cout << "Do␣you␣want␣to␣admit␣a␣new␣patient?␣(y/n):␣";
        cin >> choice;

        if (tolower(choice) == ’y’ || tolower(choice) == ’n’) {
            break; // Valid input
        } else {
            cout << "Invalid␣input.␣Please␣enter␣’y’␣or␣’n’." <<
                endl;
        }
    }

    if (tolower(choice) == ’y’) {
        string name, injury;
        cout << "Enter␣the␣name␣of␣the␣new␣patient:␣";
        cin.ignore();
        getline(cin, name);

        // Show dropdown for injuries
        cout << "Select␣an␣injury␣from␣the␣following␣options:\n";
        int count = 1;
        for (const auto& entry : injuryList) {
            cout << count++ << ".␣" << entry.first << endl;
        }

        int injuryChoice;
        while (true) {
            cout << "Enter␣the␣number␣corresponding␣to␣the␣injury:␣";
            cin >> injuryChoice;

            if (injuryChoice >= 1 && injuryChoice <= injuryList.size()) {
                auto it = injuryList.begin();
                advance(it, injuryChoice - 1);
                injury = it->first;
                break; // Valid input
            } else {
                cout << "Invalid␣input.␣Please␣enter␣a␣number␣between␣1␣and␣" << injuryList.size() << "." << endl;
            }
        }

        admitNewPatient(name, injury, time(nullptr));
        return true; // New patient added
    }
    return false; // No patient added
}

// Checks if the queue is empty
bool ERQueueHandler::isQueueEmpty() const {
    return patientQueue.empty();
}

// Main function to run the simulation
int main() {
    ERQueueHandler erSystem;

    cout << "Emergency␣Room␣Simulation␣Starting...\n";
    this_thread::sleep_for(chrono::milliseconds(1500));

    time_t now = time(nullptr);

    // Admit initial patients
    erSystem.admitNewPatient("Paarth␣Soni", "Broken␣Bone", now - 10);
    erSystem.admitNewPatient("Zach␣Hasan", "Sprained␣Ankle", now - 20);
    erSystem.admitNewPatient("Kian␣Zarkani", "Heart␣Attack", now - 5);
    erSystem.admitNewPatient("Jason␣Ie", "Severe␣Burn", now - 15);
    erSystem.admitNewPatient("Ronin␣Lee", "Mild␣Concussion", now - 2);

    // Show the initial queue
    erSystem.showQueueStatus();

    // Treat patients and prompt for new admissions
    while (!erSystem.isQueueEmpty()) {
        if (erSystem.promptForNewPatients()) {
            erSystem.showQueueStatus(); // Update queue if new patients are added
        }
        erSystem.treatNextInLine();
    }

    erSystem.showTreatedLog();
    cout << "Simulation␣Complete." << endl;
    return 0;
}
```