22AIE205
INTRODUCTION TO PYTHON

MEDIQ: EMERGENCY MEDICAL PRIORITY SCHEDULING SYSTEM

Presented by:
Diya Prakash - CB.SC.U4AIE24111
Dondluru Keerthana - CB.SC.U4AIE24112
Jyothsna V - CB.SC.U4AIE24117
V R Sridevi - CB.SC.U4AIE24166

08-10-2025

Introduction

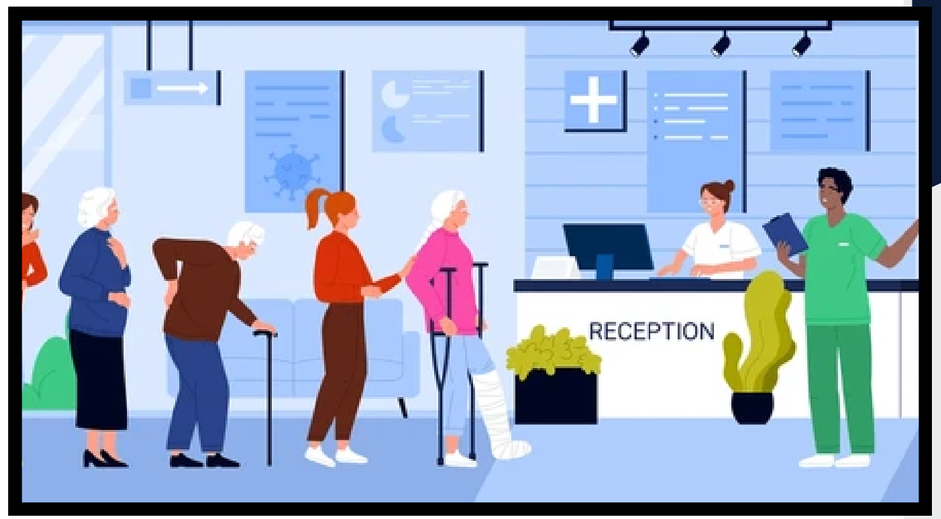
- **Background:** Emergency departments struggle with efficient patient triage, often leading to delays for critical cases and inefficient resource allocation.
- **Motivation:** Traditional first-come-first-served systems fail to account for medical urgency, patient vulnerability, and wait times, potentially endangering lives.
- **Scope:** Develop a Python-based intelligent scheduling engine using Binomial Heap for which has amortized O(1) time complexity for insertion and O(log n) time complexity for extraction operations.
- **Target Users:** Emergency departments, hospital triage systems, urgent care clinics, and emergency medical services.

Abstract

- This project implements an intelligent medical priority scheduling system using Python and advanced data structures.
- We generated a synthetic dataset of 10,000 patient records with realistic attributes including severity levels, waiting times, ages, and special needs (insurance status).
- A Binomial Heap data structure dynamically prioritizes patients based on a multi-factor scoring algorithm that considers severity, age, waiting time, and special needs.
- **Outcome:** A scalable emergency scheduling system ensuring critical patients receive priority while maintaining fairness across all cases.

Objectives

- Implement a Binomial Heap data structure for efficient patient priority queue management
- Generate a synthetic dataset of 10,000 patient records with realistic medical attributes
- Design a multi-factor priority scoring algorithm considering severity, age, waiting time, and special needs
- Ensure fair patient scheduling with proper tie-breaking mechanisms for equal priorities



Methodologies Used (Python Concepts)

1. **Object-Oriented Programming:** Custom BinomialHeapNode and BinomialHeap classes with encapsulated priority logic
2. **Data Structures:** Binomial Heap with tree linking, merging, and extraction operations
3. **File I/O:** CSV generation using Pandas for 10,000 synthetic patient records
4. **Libraries:**

- Pandas for data manipulation
- Random for synthetic data generation
- Datetime for appointment scheduling

5. **Control Structures:** Complex conditional logic for priority tie-breaking and heap operations

Priority Scoring Algorithm

Multi-Factor Priority Score Calculation:

Severity Weight:

- Emergency: 100 points
- Critical: 50 points
- Urgent: 20 points
- Normal: 0 points

Age Factor:

- Seniors (60+): age/5 = 12-20 points (60-year-old gets 12, 100-year-old gets 20)
- Children (<12): 10 points
- Normal adults (12-59): age/10 = 1.2-5.9
- Priority: Senior > Child > Normal Adult

Special Needs Bonus: +10 points for patients with special needs/insurance

Waiting Time Bonus: 0.6 × waiting_time (minutes)

Tie-Breaking Rules:

Severity → Special Needs → Check-in Time → Age (children/seniors first) → AppointmentID

Synthetic Dataset Generation

Generated 10,000 realistic patient records with:

1. **Patient Attributes:**
   - Unique Patient IDs (P001 to P10000)
   - Severity: Normal, Urgent, Emergency, Critical
   - Waiting Time: 5-120 minutes (random distribution)
   - Age: 1-100 years
   - Special Needs/Insurance: Yes/No
   - Appointment Time: September 2025, 8 AM - 10 PM
   - Doctor ID: 1-20 (random assignment)
2. **Data Format:** CSV file with 10,000 rows, 7 columns
3. **Advantages:** Reproducible testing, no privacy concerns, controlled distribution of severity levels

Binomial Heap Implementation

Key Operations Implemented:

- Insert - Add new patient to priority queue
- Extract Max - Retrieve highest priority patient for treatment
- Union - Merge two heaps after extraction
- Tree Linking: Combine binomial trees of same degree
- Root List Merging: Maintain heap property across trees

Why Binomial Heap?

- Efficient insertion and extraction
- Better worst-case performance than binary heaps
- Supports efficient merging operations
- Suitable for dynamic priority queues

Results and Validation

✅ Generated 1000 synthetic patient records and saved to 'synthetic_patients1.csv'

	PatientID	sevierity	WaitingTime	Age	Special Needs	AppointmentTime	DoctorID
0	P001	Normal	109	3	Yes	2025-09-04 20:00:00	18
1	P002	Normal	116	39	No	2025-09-06 15:30:00	7
2	P003	Urgent	66	100	Yes	2025-09-04 19:45:00	10
3	P004	Emergency	65	87	Yes	2025-09-05 13:00:00	14
4	P005	Normal	89	72	No	2025-09-19 09:45:00	2
...	...	...	...	...	...	...	...
995	P996	Emergency	84	62	No	2025-09-30 07:15:00	13
996	P997	Emergency	84	33	No	2025-09-24 07:15:00	17
997	P998	Critical	107	19	No	2025-09-30 15:45:00	3
998	P999	Critical	55	84	No	2025-09-23 21:30:00	7
999	P1000	Urgent	70	38	No	2025-09-23 17:30:00	5

1000 rows × 7 columns

```
Serving order (Patient ID, Priority Score, Age, Severity):
P5928 200.0 96 Emergency
P9186 199.2 98 Emergency
P8772 199.0 100 Emergency
P8672 199.0 88 Emergency
P9658 198.4 82 Emergency
P6186 198.0 89 Emergency
P2273 197.8 88 Emergency
P7995 197.4 83 Emergency
P3326 196.8 83 Emergency
P9287 196.8 95 Emergency
P2673 196.8 74 Emergency
P9402 196.6 94 Emergency
P5137 196.6 79 Emergency
P735 196.4 84 Emergency
P8157 196.2 80 Emergency
P4945 195.6 77 Emergency
P4315 195.6 83 Emergency
P9936 195.4 70 Emergency
P9270 195.2 66 Emergency
P7662 195.0 83 Emergency
P3752 195.0 65 Emergency
P6839 195.0 80 Emergency
P2946 194.8 79 Emergency
P5396 194.8 67 Emergency
```

Results and Validation

System Performance:

- **Successfully processed 10,000 patient records**
- **Patients extracted in correct priority order**
- **Highest priority: Emergency cases with long wait times**
- **Special consideration for children and seniors**

Sample Output (Top 24 Patients Served):

- **Emergency patients served first regardless of other factors**
- **Critical patients with high waiting times prioritized**
- **Tie-breaking correctly applied (severity → special needs → time)**
- **Fair distribution across age groups when severity equal**

Verification: Manual inspection of first 24 extractions confirms algorithm correctness

Performance: Sub-second insertion and extraction times even with 10,000 records

Successfully made website using HTML for hospital staff to use

Conclusion

We successfully developed a Python-based medical priority scheduling system using Binomial Heap data structure, capable of efficiently managing 10,000 patient records with amortised O(1) time complexity for insertion and O(logn) time complexity for extraction operations.

The system implements a sophisticated multi-factor priority algorithm that considers severity, age, waiting time, and special needs to ensure both medical urgency and fairness in patient scheduling.

Through comprehensive testing and validation, we demonstrated that advanced data structures can be effectively applied to solve real-world healthcare challenges, providing a scalable foundation for intelligent hospital triage systems that could potentially save lives through optimized resource allocation.

THANK YOU