Algorithm Comparison Report: Hospital Resource Optimization

1. Administrative

- **Team Name:** CodeCure
- **Team Members + GitHub Usernames:**
  - Rishabh Fuke (`Rishabh-Fuke`)
  - Neerav Gandhi (`neerav-g`)
  - Siddhant Pallod (`siddhantpallod`)
- **Link to GitHub Repository:** Rishabh-Fuke/DSA_Project2: DSA Project 2
- **Link to Video Demo:** https://youtu.be/78r8KojETps

2. Extended and Refined Proposal

Problem and Motivation

Problem: The core challenge in acute healthcare is the dynamic and unpredictable allocation of **scarce, specialized resources** (e.g., ICU beds, specialized doctors) to patients with widely varying, time-sensitive needs. The goal is to minimize patient wait time and optimize resource utilization while maintaining fairness and efficiency. This is essentially a complex, real-time scheduling and optimization problem.

Motivation: During health crises like COVID-19, hospitals worldwide faced severe shortages and inefficient allocation of critical resources. Many facilities relied on manual scheduling systems that failed to prioritize high-urgency patients effectively. A data-driven optimization system can significantly improve hospital management by automatically prioritizing patients based on urgency, treatment duration, and resource availability, improving both fairness and patient outcomes.

Features Implemented and Description of Data

Features Implemented

- **Two Core Allocators:** `greedy_allocation.py` and `dynamic_algo.py` providing distinct trade-offs between execution speed and global optimality.

- **Input/Output:** Handles patient input data and generates an optimized allocation schedule for doctors and ICU beds.
- **Comparison Metrics:** `analysis.py` measures and compares algorithms based on total waiting time, resource utilization rate, and fairness of distribution.
- **Data Generation Utility:** `data_generator.py` for creating synthetic datasets that accurately mimic patient arrival times, urgency, and resource demands.

Description of Data The system relies on a standardized input format derived from a synthetic CSV, detailing five critical attributes per patient:

1. `patient_id`: Unique identifier.
2. `urgency_score`: Float (1.0 to 10.0), patient criticality (higher is more urgent).
3. `arrival_time`: ISO timestamp of entry.
4. `treatment_duration`: Required time (in minutes) for the treatment.
5. `resource_type`: Specifies the required resource: `Doctor` or `ICU`.

The dataset size used for robust performance testing is **100,000** rows.

Tools, Languages, and Algorithms

Tools, Languages, and Libraries Used

- **Language:** Python
- **Core Libraries:** NumPy, Pandas (data handling), Matplotlib (visualization).

Algorithms Implemented

1. **Greedy Allocator (`greedy_allocation.py`):** It uses a max-heap priority queue to assign the highest-urgency patients first. This prioritizes quick local decisions and minimal computation time.
2. **Dynamic Programming Optimization (`dynamic_algo.py`):** Uses a DP approach to minimize total waiting time and maximize fairness under limited resources. It aims to find a globally optimal allocation across all patients and time periods.

Additional Data Structures/Algorithms Used

- **Priority Queue (Max-Heap):** Fundamental to the Greedy approach, ensuring O(log N) retrieval time for the highest priority patient.
- **Timsort:** Used by Python's list sort in the DP approach for iteratively re-sorting the remaining patients based on their dynamic priority scores.

- **Hash Map:** Used for tracking resource availability for quick lookup.

Distribution of Responsibility and Roles

- **Rishabh Fuke:** Data generation, initial Dynamic Programming implementation, and helped with performance comparison
- **Neerav Gandhi:** Implementation and testing of the Greedy algorithm.
- **Siddhant Pallod:** Refined Dynamic Programming implementation, performance comparison, and visualization using Matplotlib.

3. Analysis

Changes Made After the Proposal

The most significant refinement was made to the **Dynamic Programming priority function**.

- **Change:** The initially complex, exponential DP priority formula was simplified to a **linear combination**:
  {Priority} = {Urgency Score} + {alpha} * {Estimated Wait Time}

- **Rationale:** The initial formula resulted in unpredictable priority fluctuations that could lead to starvation of low-urgency patients. The simplified linear approach is more transparent and allows the alpha weight to act as a direct tuning parameter to balance the trade-off between serving highly urgent patients and improving overall system throughput.

Big O Worst-Case Time Complexity Analysis

The analysis assumes N is the number of patients and R is the total number of resources (doctors and ICUs).

Algorithm	Dominant Operation	Worst-Case Complexity	Scalability
Greedy Allocator	Priority Queue and Resource Checks	O(N * (log N + R))	Excellent

DP Allocator	Repeated Sorting of Remaining Patients	O(N^2 * R)	Poor for Large N

Greedy Allocator: The complexity is dominated by the main loop over N patients, each requiring a constant-time check against R resources. This makes it suitable for real-time operation.

Time-Indexed DP Allocator: The complexity is dominated by the fact that the list of remaining patients must be re-sorted based on their dynamic priority in every iteration of the allocation process. This N-times repetition of an O((N log N) operation results in quadratic complexity with respect to the number of patients, making it computationally expensive for large datasets (100,000 rows).

4. Reflection

Overall Experience and Challenges

The project was a highly challenging but rewarding exercise in applying discrete optimization techniques to a critical practical problem. The most engaging aspect was the necessity of **balancing computational complexity against global solution efficacy**; the core motivation behind comparing the Greedy and DP approaches.

Challenges Encountered

1. **Metric Synchronization:** Difficulties arose in ensuring that metrics from the continuous-time Greedy algorithm were perfectly comparable to the discrete-slot DP algorithm, requiring careful handling of time calculations.
2. **DP Complexity:** The O(N^2 log N) complexity of the DP model meant that running simulations with large datasets was computationally expensive, forcing the team to focus on internal loop optimization.
3. **Priority Tuning:** Selecting an appropriate value for the alpha weight in the DP model was challenging, as minor changes significantly altered the algorithm's scheduling behavior.

Changes to Project and Workflow

If we were to start the project again, we would implement the following changes:

- **Dedicated Configuration File:** Immediately replace hardcoded simulation parameters (e.g., number of doctors, simulation duration) with a dedicated configuration file (JSON/YAML). This would improve flexibility and simplify test script execution.
- **Earlier Integration Testing:** Implement end-to-end integration tests earlier, using small, manually validated patient datasets to confirm that the entire data to allocation to metric calculation pipeline worked correctly before scaling up to the required 100,000 row dataset.

Individual Learning Outcomes

- **Rishabh Fuke:** Deepened understanding of challenges in data optimization contexts, and practical experience in initial Dynamic Programming implementation by realizing the significance of exponential complexities for real-world scheduling.
- **Neerav Gandhi:** Mastered the performance implications of Python's `PriorityQueue` and gained practical experience in creating robust and fast heuristic algorithms for real-time systems.
- **Siddhant Pallod:** Enhanced skills in refining complex optimization algorithms (DP), performing large-scale performance comparison, and developing expertise in generating comparative, production-quality data visualizations using Matplotlib.

References

1. Python Software Foundation. Python Language Reference. https://www.python.org/
2. McKinney, Wes. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*.
3. Taha, Hamdy A. (2017). *Operations Research: An Introduction*. Pearson.