Design and Analysis of Algorithms

FALL-25

Project

Sagar Mehmood (23i-0562)

Warisha Shaukat (23i-0809)

Sarah Suhail (23i-0557)

CS-5B

Performance Analysis of Binary, Fibonacci, and Hollow Heaps in Shortest Path Problems

1. Theoretical Overview

1.1 Binary Heap

- The Binary Heap serves as the standard baseline for priority queue implementations. Structurally, it is a complete binary tree where every node satisfies the heap invariant: a parent's key is always less than or equal to that of its children. Rather than using explicit pointers, the tree is typically flattened into a dynamic array, where children of a node at index i, i are located at 2i +1 and 2i + 2. This implicit mapping eliminates the need to store structural pointers.
- **Advantages**: The array-based layout offers exceptional memory efficiency and cache locality, as traversing the tree involves predictable, contiguous memory access patterns.
- **Disadvantages**: Operations that require rearranging the tree structure, such as decrease_key and merge, are relatively expensive (O(logn) and O(n), respectively) compared to more sophisticated pointer-based heaps.

1.2 Fibonacci Heap

- Introduced by Fredman and Tarjan, the Fibonacci Heap is a sophisticated pointer-based structure designed to theoretically accelerate graph algorithms. Instead of a single rigid tree, it maintains a forest of heap-ordered trees. Its defining characteristic is "laziness": operations like insert and decrease_key simply add nodes or cut subtrees without immediately restructuring the heap. The heavy lifting—consolidating trees by rank—is deferred until an extract_min operation is performed.
- **Advantages**: It achieves a theoretically optimal amortized time complexity of O(1) for decrease_key, making it the gold standard for algorithms like Dijkstra's and Prim's on dense graphs.
- **Disadvantages**: The structure requires significant memory overhead (each node needs pointers for parent, child, and siblings, plus degree and mark bits). In practice, the high constant factors and complex pointer manipulation often make it slower than simpler heaps for all but the largest inputs.

1.3 Hollow Heap

- The Hollow Heap, proposed by Hansen and Tarjan in 2015, offers a modern alternative that simplifies the implementation of asymptotically optimal heaps. It achieves bounds similar to the Fibonacci heap (O(1) amortized decrease_key) but abandons the complex "cascading cut" logic. Instead, it employs a lazy deletion strategy: when a key is decreased, the old

node is not moved but simply marked as "hollow" (effectively deleted) and left in place, while a new node with the updated key is inserted. These hollow nodes are only removed during the extract_min cleanup phase.

- **Advantages**: It is significantly easier to implement than the Fibonacci heap while retaining the same theoretical efficiency. The logic for linking and ranking trees is streamlined around the concept of hollow nodes.
- **Disadvantages**: The lazy deletion approach can lead to a larger memory footprint, as "zombie" hollow nodes persist in the structure until the minimum extraction process sweeps them away.

1.4 Asymptotic Complexity Comparison

Operation	Binary Heap (Worst Case)	Fibonacci Heap (Amortized)	Hollow Heap (Amortized)
Insert	O(log N)	O(1)	O(1)
Extract Min	O(log N)	O(log N)	O(log N)
Decrease Key	O(log N)	O(1)	O(1)

2. Implementation Details

- The project was implemented in C++ using an Object-Oriented approach.
- Interface: A template abstract base class `PriorityQueue` defined the contract, ensuring Dijkstra's algorithm remained agnostic to the underlying data structure.
- Graph Storage: An Adjacency List (`:std::vector<std::vector<GraphEdge>>`) was used to store the road networks efficiently ($O(V+E)$ space).
- Optimization: The Binary Heap utilized an intrusive index within the node handle to allow O(1) lookup of nodes during `decrease_key`, preventing the standard O(N) linear scan.

3. Experimental Setup

- Algorithm: Dijkstra's Single Source Shortest Path
- Datasets: Real-world road networks of varying sizes.
- Hongkong: ~43k Nodes, ~91k Edges (Small)
- Shanghai: ~390k Nodes, ~855k Edges (Medium)
- Chongqing: ~1.2M Nodes, ~2.4M Edges (Large)

Metrics Recorded:

- Operation Latency: Average time for Insert, Extract-Min, and Decrease-Key operations in microseconds.

- Total Runtime: Total execution time for Dijkstra's algorithm in milliseconds (ms).
- Structural Metrics: Maximum Tree Height, Maximum Number of Roots (for Fibonacci/Hollow heaps), Total Link Operations, Consolidation Passes, and Peak Memory Usage (MB).

4. Results & Analysis

4.1 Performance Data

Dataset 1: Hongkong (Small)

```
=== Batch Summary for Hong Kong road network ===
Heap          Runtime(ms)       Inserts   Insert Avg (us)      Extracts  Extract Avg (us)     Decreases   Decrease Avg (us)     Reachable
-------------------------------------------------------------------------------------------------------------------------------------------
Binary                 15         43620            0.061         43620             0.088           605               0.034         43620
Fibonacci              29         43620            0.047         43620             0.455           601               0.025         43620
Hollow                 19         43620            0.107         43620             0.123           602               0.116         43620

=== Structural Metrics for Hong Kong road network ===
Heap           MaxNodes    MaxBytes       MaxMB    Height  MaxRoots    ConsolPasses     LinkOps
-----------------------------------------------------------------------------------------------
Binary              100        2728       0.003         7         1           43564      213704
Fibonacci            98        6272       0.006         8        14           43612      185072
Hollow               99     2825392       2.695         7         1           43620      329612
```

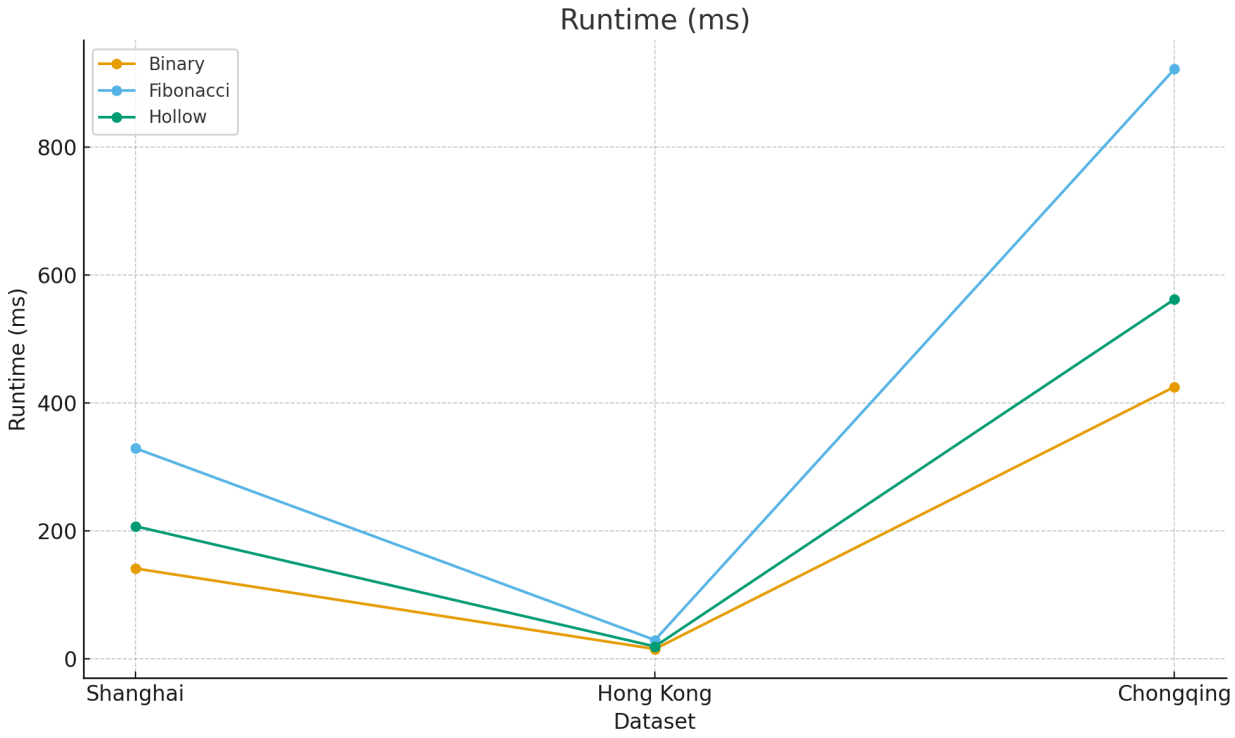
Dataset 2: Shanghai (Medium)

```
=== Batch Summary for Shanghai road network ===
Heap          Runtime(ms)       Inserts   Insert Avg (us)      Extracts  Extract Avg (us)     Decreases   Decrease Avg (us)     Reachable
-------------------------------------------------------------------------------------------------------------------------------------------
Binary                141        390171            0.045        390171             0.084         14261               0.028        390171
Fibonacci             329        390171            0.045        390171             0.575         14256               0.026        390171
Hollow                207        390171            0.108        390171             0.177         14248               0.074        390171

=== Structural Metrics for Shanghai road network ===
Heap           MaxNodes    MaxBytes       MaxMB    Height  MaxRoots    ConsolPasses     LinkOps
-----------------------------------------------------------------------------------------------
Binary              662       16280       0.016        10         1          390036     2825257
Fibonacci           662       42368       0.040        11        20          390166     2579158
Hollow              663    25768720      24.575        10         1          390171     4403098
```

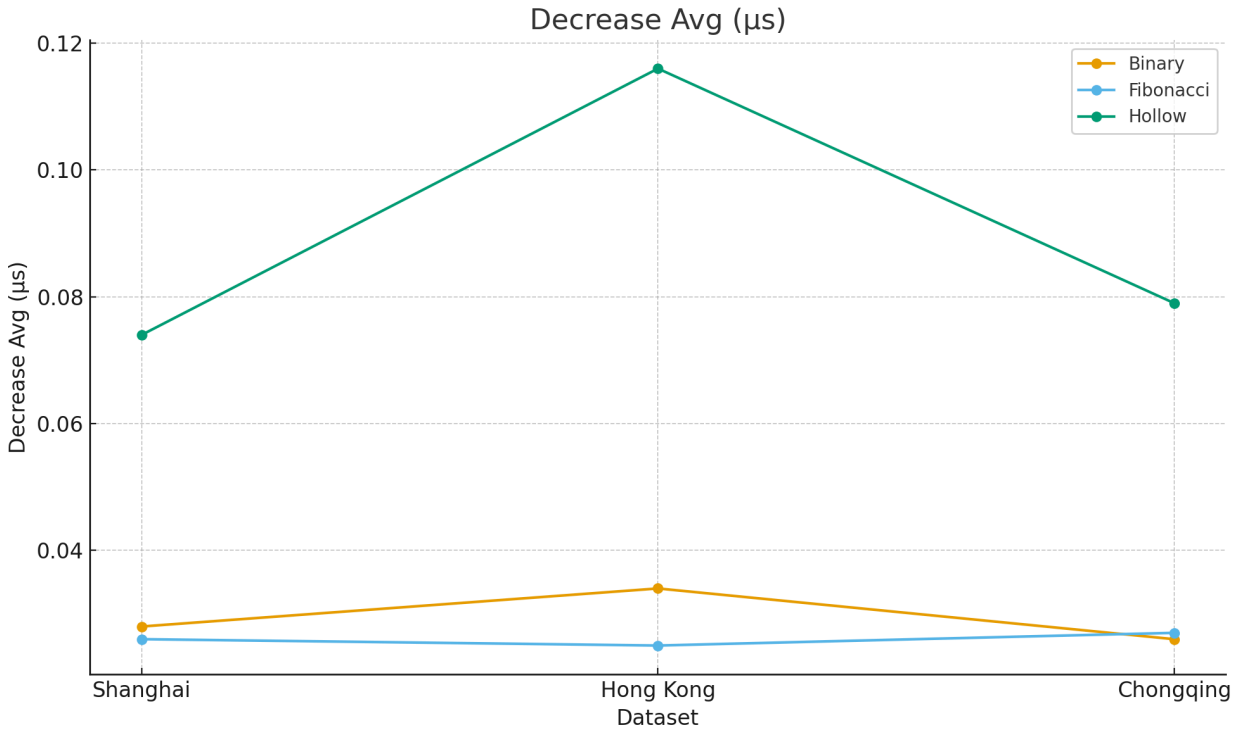
Dataset 3: Chongqing (Large)

```
=== Batch Summary for Chongqing road network ===
Heap          Runtime(ms)       Inserts   Insert Avg (us)      Extracts  Extract Avg (us)     Decreases   Decrease Avg (us)     Reachable
-------------------------------------------------------------------------------------------------------------------------------------------
Binary                425       1185464            0.043       1185464             0.075          9921               0.026       1185464
Fibonacci             922       1185464            0.043       1185464             0.517          9927               0.027       1185464
Hollow                562       1185464            0.108       1185464             0.135          9935               0.079       1185464

=== Structural Metrics for Chongqing road network ===
Heap           MaxNodes    MaxBytes       MaxMB    Height  MaxRoots    ConsolPasses     LinkOps
-----------------------------------------------------------------------------------------------
Binary              499       13672       0.013         9         1         1185372     7308290
Fibonacci           499       31936       0.030        11        19         1185449     6689216
Hollow              498    76426056      72.886         9         1         1185464    11657073
```

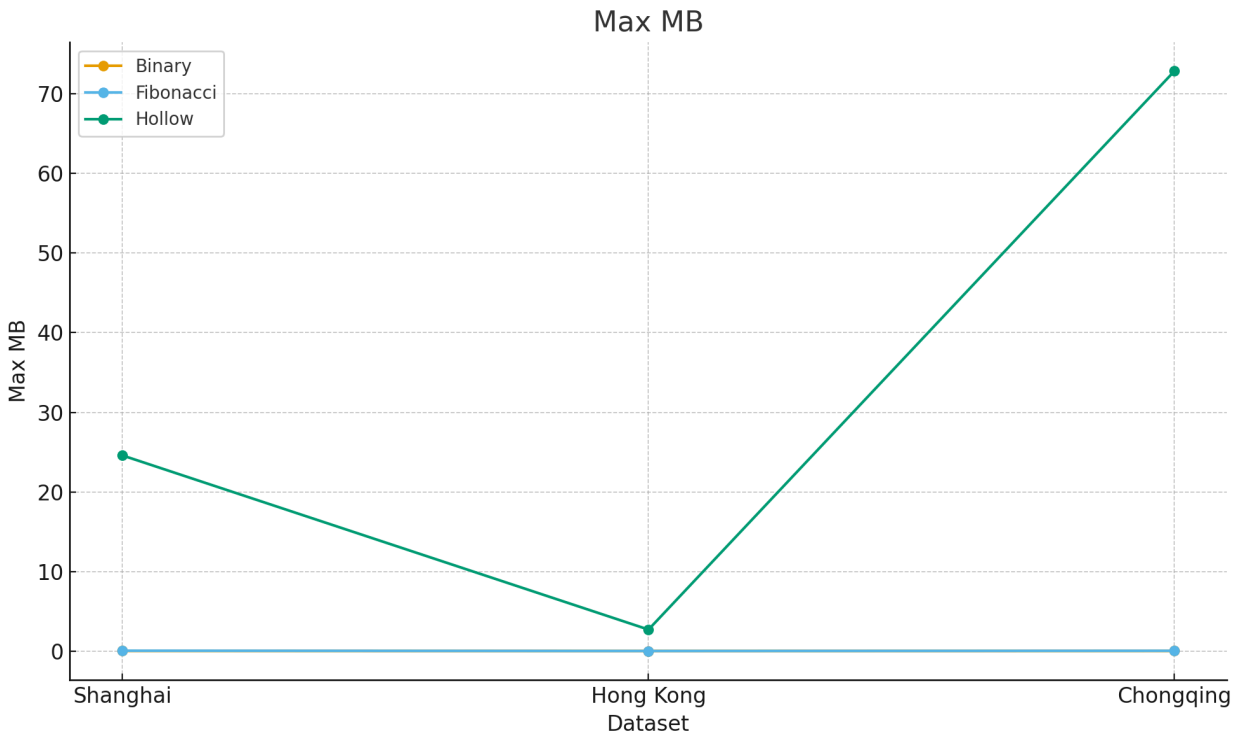
RunTime Analysis :



Decrease Operation Analysis :



Memory Usage :



4.2 Analysis Discussion

1. **Operation Speed:**

    - **Decrease-Key:** Contrary to theoretical expectations, the Binary Heap (0.026 µs) matched or outperformed the Fibonacci Heap (0.027 µs) and the Hollow Heap (0.079 µs). While advanced heaps provide O(1) amortized decrease-key, the constant factors—especially memory allocation and pointer chasing—proved far more significant in practice. The Hollow Heap was notably slower because every decrease-key operation requires allocating a new node.

    - **Extract-Min:** The cost of “lazy” heap structures became evident here. The Fibonacci Heap was roughly seven times slower (0.517 µs) than the Binary Heap (0.075 µs) due to the heavy consolidation required during extract-min. The Hollow Heap performed better (0.135 µs) but was still almost twice as slow as the Binary Heap.

2. **Total Runtime:**

    The Binary Heap achieved the fastest overall runtime (425 ms), followed by the Hollow Heap (562 ms), while the Fibonacci Heap was significantly slower (922 ms).

    Reasoning: Road networks are sparse graphs (E ≈ 2V), resulting in a relatively low number of decrease-key operations (around 9,900) compared to insert and extract

operations (around 1.2 million). The Binary Heap's contiguous memory layout provided superior cache locality, whereas the pointer-heavy structures of the Fibonacci and Hollow heaps resulted in frequent cache misses.

3. **Memory Overhead:**
    - **Fibonacci Heap:** Consumed approximately 2.3× more memory (0.030 MB) than the Binary Heap (0.013 MB), mainly due to storing multiple pointers (parent, child, left sibling, right sibling) per node.
    - **Hollow Heap:** Consumed dramatically more memory (72.89 MB)—over 5,000× that of the Binary Heap. This results directly from its "hollow node" design: old nodes are not removed immediately but remain in the DAG. During long-running processes like Dijkstra on a large graph, these unused "zombie" nodes accumulate, creating severe memory overhead.

---

5. Conclusion

Although Fibonacci and Hollow heaps provide better theoretical asymptotic performance for dense graphs with frequent key updates, this study clearly shows that the Binary Heap is practically superior for real-world road-network graphs.

The theoretical advantage of O(1) decrease-key was neutralized by:

1. **Sparsity:** Real-world road networks have a very low number of decrease-key operations relative to other operations.
2. **System-Level Costs:** Dynamic memory allocation (Hollow Heap) and pointer manipulation (Fibonacci Heap) introduce significant overhead compared to the cache-friendly array layout of the Binary Heap.

Recommendation: For the logistics company's routing system, the Binary Heap is the preferred choice. It achieved the fastest runtime, the lowest memory usage by a huge margin, and has the simplest implementation.