Benchmark cấu trúc dữ liệu cho Leaderboard Realtime

1. Phương pháp

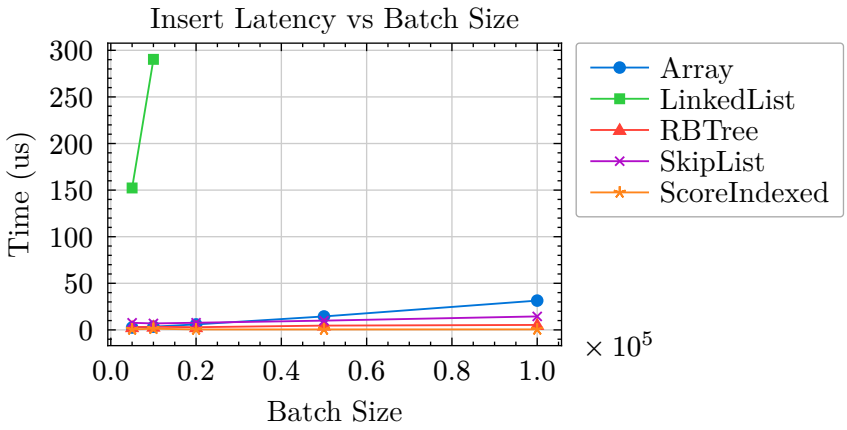
Ta đánh giá hiệu năng của 5 cấu trúc dữ liệu (Sorted Array, Linked List, Red-Black Tree, Skip List, Score-Indexed Array) thông qua hai kịch bản kiểm thử:

1. **Micro-benchmark**:
   - Đo thời gian thực thi trung bình của từng thao tác đơn lẻ: **Insert**, **Search**, **Delete**.
     - Thực hiện trên các tập dữ liệu có kích thước tăng dần: 5,000, 10,000, 20,000, 50,000 và 100,000 phần tử.
     - Mục tiêu: Đánh giá chi phí cơ bản của từng thao tác.
2. **Realtime Simulation**:
   - Mô phỏng môi trường thực tế với tải cao (High Churn).
   - **Tỷ lệ Churn**: 30% người dùng cập nhật điểm số mỗi giây (Update = Delete cũ + Insert mới).
   - **Quy mô**: Kiểm thử khả năng chịu tải với 5,000, 10,000, 20,000, 50,000 và 100,000 phần tử.
   - **Tiêu chí**: Đo độ trễ (latency) của thao tác Update và Search. Hệ thống được coi là "quá tải" (Falling Behind) nếu thời gian xử lý batch 1 giây vượt quá 1 giây thực tế.

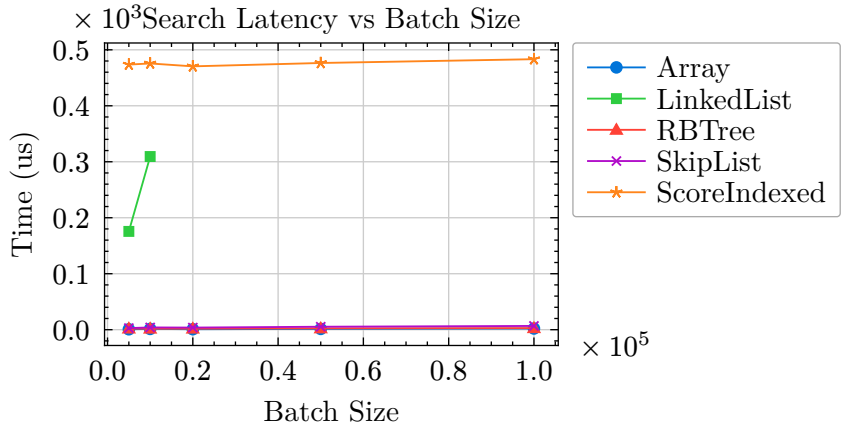
2. Kết quả thực nghiệm

2.1. Micro-benchmark (Độ trễ thao tác đơn lẻ)

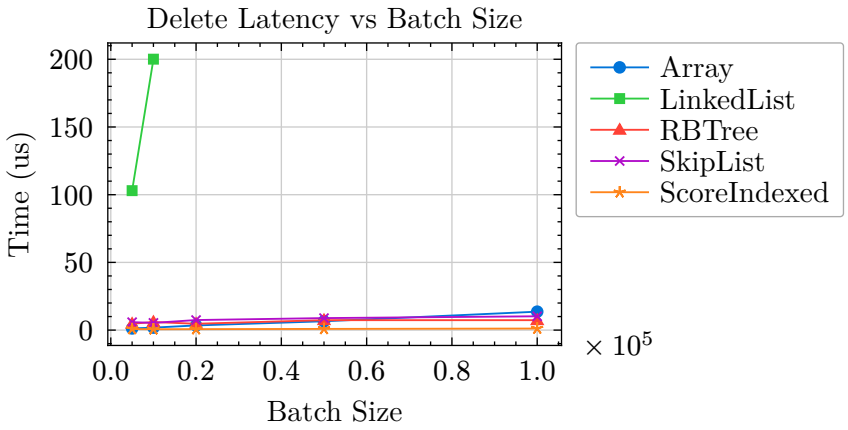
Biểu đồ dưới đây thể hiện sự thay đổi thời gian thực thi (Average Latency) theo kích thước dữ liệu (Batch Size).



Hình 1: Insert Latency (Average)



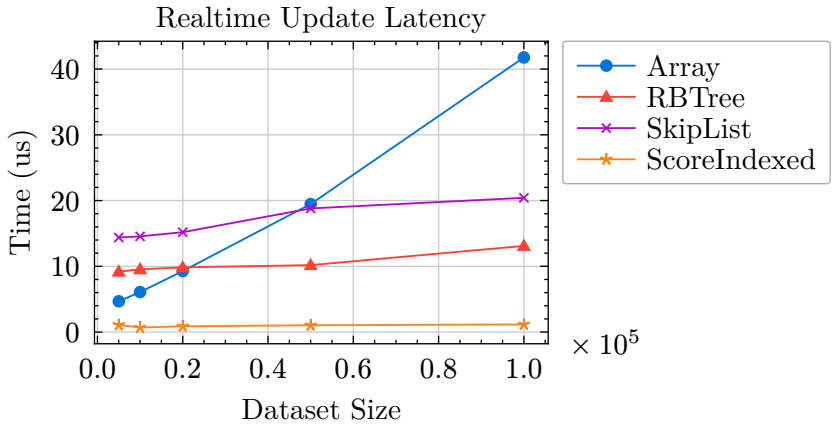
Hình 2: Search Latency (Average)



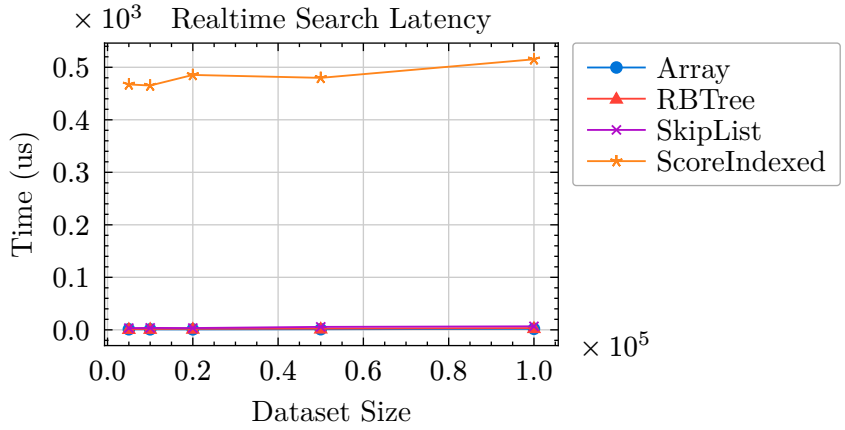
Hình 3: Delete Latency (Average)

2.2. Realtime Simulation (Độ trễ dưới tải cao)

Mô phỏng tải thực tế với 30% user cập nhật điểm số mỗi giây.



Hình 4: Update Latency under Load

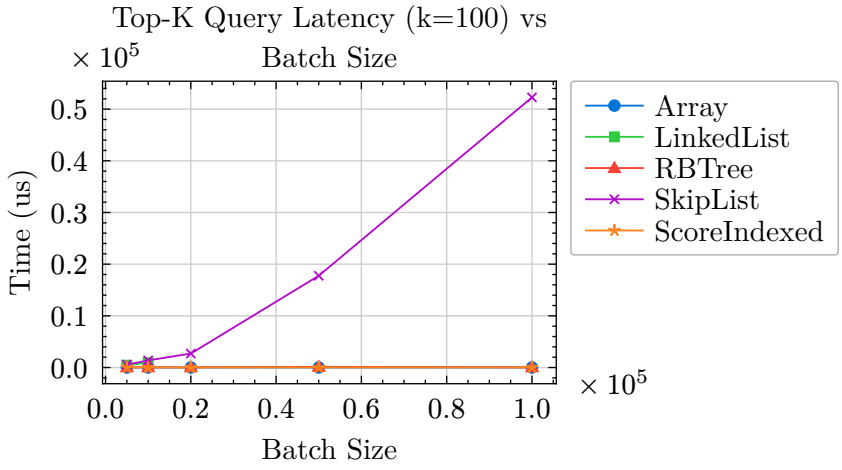


Hình 5: Search Latency under Load

2.3. Top-K Query Performance (k = 100)

Truy vấn Top-K là thao tác quan trọng trong leaderboard để hiển thị bảng xếp hạng. Ta đánh giá hiệu năng truy vấn Top-100 (k = 100) cho mọi kích thước dữ liệu.

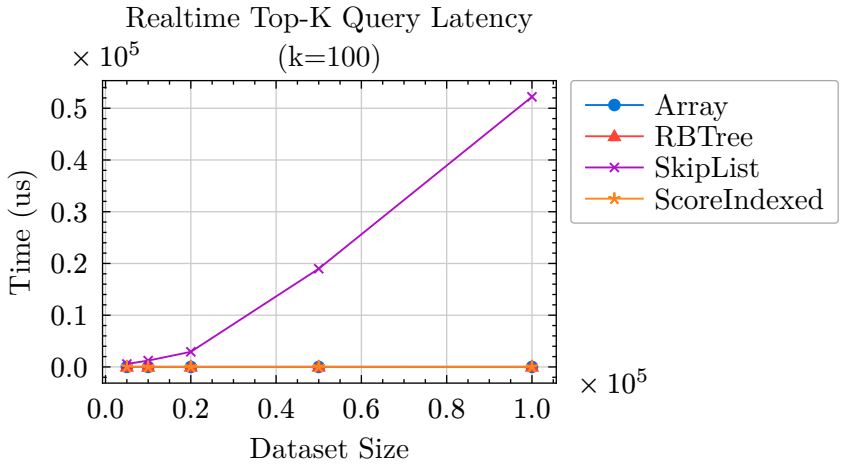
2.3.1. Micro-benchmark Top-K



Hình 6: Top-K Query Latency (Average, k=100)

2.3.2. Realtime Top-K Simulation

Mô phỏng tải thực tế với 30% user cập nhật và 30% truy vấn Top-K mỗi giây.



Hình 7: Top-K Query Latency under Load (k=100)

Phân tích Top-K Performance:

- **Sorted Array**: Cực nhanh ($O(k)$) vì chỉ cần copy k phần tử cuối mảng. Tốt nhất cho truy vấn Top-K.
- **Linked List**: Rất chậm ($O(N)$) vì phải duyệt toàn bộ danh sách để tìm k phần tử cuối.
- **Red-Black Tree**: Hiệu năng tốt ($O(k \log N)$) nhờ reverse in-order traversal.
- **Skip List**: Tương tự RB Tree, cần duyệt qua danh sách để lấy k phần tử.
- **Score-Indexed Array**: Hiệu năng trung bình, phụ thuộc vào phân bố điểm số.

3. Nhận xét và So sánh

3.1. Phân tích chi tiết

- **Sorted Array**: Rất nhanh ở các thao tác đọc (Search) nhờ Binary Search. Tuy nhiên, thao tác ghi (Insert/Delete/Update) cực chậm ($O(N)$) khi dữ liệu lớn. Trong kịch bản Realtime, nó bắt đầu quá tải khi dữ liệu đạt 100k phần tử.
- **Linked List**: Hiệu năng kém nhất ($O(N)$ cho mọi thao tác). Không vượt qua được bài test Realtime ngay cả ở mức 10k phần tử.
- **Red-Black Tree**: Cấu trúc ổn định nhất. Đảm bảo hiệu năng $O(\log N)$ cho mọi thao tác, chịu tải tốt ở mức 100k phần tử.

- **Skip List**: Hiệu năng tương đương Red-Black Tree, là một sự thay thế tốt với cài đặt có phần đơn giản hơn về mặt ý tưởng (dù tốn bộ nhớ hơn).
- **Score-Indexed Array**: Sử dụng mảng 15k phần tử, mỗi phần tử lưu danh sách ID người chơi có điểm tương ứng. Insert/Update cực nhanh ($O(1)$), nhưng Search rank chậm ($O(\text{max\_score})$). Phù hợp khi có ràng buộc điểm số và ưu tiên thao tác ghi.

3.2. Bảng so sánh tổng hợp

Thuật toán	**Độ phức tạp** (Search/Update)	**Micro-bench** (Read/Write)	**Realtime** (Scalability)	**Đánh giá**
Sorted Array	$O(\log n)$ / $O(n)$	Rất Tốt / Kém	Kém (Fail @ 100k)	Tốt cho ít ghi
Linked List	$O(n)$ / $O(n)$	Kém / Kém	Rất Kém (Fail @ 10k)	Không phù hợp
RB Tree	$O(\log n)$ / $O(\log n)$	Tốt / Tốt	Tốt (Pass @ 100k)	Khuyên dùng
Skip List	$O(\log n)$ / $O(\log n)$	Tốt / Tốt	Tốt (Pass @ 100k)	Thay thế tốt
Score-Indexed	$O(\text{max})$ / $O(1)$	Chậm / Rất Tốt	Tốt (Pass @ 100k)	Tốt cho nhiều ghi