**a. So sánh HashMap của Java và Dictionary của C#**

1. **Khái niệm**:
   * **Java HashMap**: Là một lớp trong Java Collections Framework, dùng để lưu trữ các cặp khóa-giá trị (key-value). Mỗi khóa chỉ có một giá trị duy nhất, và không thể chứa các khóa trùng lặp.
   * **C# Dictionary**: Là một lớp trong .NET Collections, cũng sử dụng các cặp khóa-giá trị tương tự như HashMap của Java. Không cho phép khóa trùng lặp và lưu trữ khóa-giá trị theo cách không có thứ tự.
2. **Tính năng chung**:
   * Cả HashMap và Dictionary đều sử dụng hàm băm (hashing) để lưu trữ và truy cập các cặp key-value, cung cấp hiệu năng truy cập theo thời gian trung bình là O(1).
   * Cả hai đều không đảm bảo thứ tự các phần tử khi lặp qua chúng.
   * Đều không cho phép khóa trùng lặp nhưng có thể có giá trị trùng lặp.
3. **Khác biệt**:
   * **Xử lý null**:
     + HashMap (Java) cho phép lưu trữ null như một khóa hoặc giá trị.
     + Dictionary (C#) không cho phép null là khóa. Nếu bạn cố gắng thêm một khóa là null, sẽ xảy ra ngoại lệ.
   * **Thread Safety**:
     + HashMap không phải là thread-safe. Nếu cần sử dụng trong môi trường đa luồng, cần sử dụng lớp Collections.synchronizedMap() hoặc ConcurrentHashMap.
     + Dictionary cũng không phải là thread-safe. Để làm việc trong môi trường đa luồng, C# cung cấp ConcurrentDictionary.
   * **Triển khai khác nhau**:
     + HashMap là một phần của thư viện java.util, trong khi Dictionary của C# nằm trong System.Collections.Generic.

**b. So sánh HashMap và HashSet của Java**

1. **Mục đích**:
   * **HashMap**: Lưu trữ các cặp khóa-giá trị. Mỗi khóa sẽ trỏ đến một giá trị và không cho phép các khóa trùng lặp.
   * **HashSet**: Lưu trữ các phần tử duy nhất (giá trị) mà không có sự trùng lặp. Không lưu trữ cặp khóa-giá trị mà chỉ lưu giá trị duy nhất.
2. **Cấu trúc nội bộ**:
   * **HashMap**: Dựa trên bảng băm (hash table) để lưu trữ các cặp khóa-giá trị.
   * **HashSet**: Dựa trên một HashMap bên trong để lưu trữ các phần tử. Trong đó, các phần tử được lưu làm khóa của HashMap, và giá trị mặc định là một constant dummy object.
3. **Hiệu năng**:
   * Cả **HashMap** và **HashSet** đều có hiệu năng thời gian truy cập trung bình O(1) khi tìm kiếm hoặc thêm/xóa phần tử nhờ sử dụng hàm băm.
4. **Sử dụng**:
   * **HashMap**: Sử dụng khi cần ánh xạ các khóa với các giá trị tương ứng.
   * **HashSet**: Sử dụng khi chỉ cần quản lý một tập hợp các giá trị duy nhất.

**c. So sánh tốc độ tìm kiếm của HashMap và TreeMap trong Java**

1. **HashMap**:
   * **Cấu trúc**: Sử dụng bảng băm (hash table) để lưu trữ các cặp khóa-giá trị.
   * **Tốc độ tìm kiếm**: Có thời gian tìm kiếm trung bình là O(1), vì việc tìm khóa trong bảng băm dựa trên hàm băm, cho phép truy cập trực tiếp.
   * **Thứ tự**: Không duy trì thứ tự của các phần tử. Khi lặp qua các phần tử, thứ tự sẽ là ngẫu nhiên.
2. **TreeMap**:
   * **Cấu trúc**: Dựa trên cấu trúc cây nhị phân cân bằng (Red-Black Tree).
   * **Tốc độ tìm kiếm**: Có thời gian tìm kiếm O(log n), vì TreeMap phải duyệt qua cây để tìm kiếm các khóa theo thứ tự sắp xếp.
   * **Thứ tự**: Duy trì các phần tử theo thứ tự tự nhiên của khóa hoặc theo một bộ so sánh tùy chỉnh (Comparator).
3. **Hiệu năng**:
   * **HashMap** có hiệu năng tốt hơn cho các thao tác tìm kiếm thông thường, đặc biệt khi chỉ cần tìm hoặc lưu trữ mà không cần quan tâm đến thứ tự.
   * **TreeMap** có hiệu năng thấp hơn cho các thao tác tìm kiếm vì phải duy trì thứ tự, nhưng thích hợp nếu cần truy cập dữ liệu theo thứ tự sắp xếp của khóa, chẳng hạn như tìm kiếm tuần tự hoặc lấy khóa lớn nhất/nhỏ nhất.

**Tổng kết:**

* **HashMap** nhanh hơn **TreeMap** khi tìm kiếm vì nó sử dụng hàm băm, có tốc độ O(1) so với O(log n) của TreeMap.
* **TreeMap** hữu ích trong trường hợp cần duy trì thứ tự khóa và thực hiện các thao tác tuần tự như tìm khóa lớn nhất/nhỏ nhất.