

**컨테이너: C++**: Vector, **C#**: List<T>

**자료구조:** 동적 배열

**임의 접근**: (O(1))

**요소 추가/삭제**: 배열의 끝에서 이루어질 때 빠름 (O(1))

**중간 삽입/삭제**: 비효율적 (O(n))

**컨테이너: C++**: List<T>, **C#**: LinkedList<T>

**자료구조:** 이중 연결 리스트

**순차 접근**: (O(n))

**임의 접근**: 비효율적 (O(n))

**중간 삽입/삭제**: 효율적 (O(1))

**컨테이너: C++**: unordered\_map, **C#:** Dictionary<TKey, TValue>

**자료구조:** 해시 테이블

**중복 배제**: 그런 코드가 있으면 알아서 무시됨

**성능**: 평균적으로 O(1) 시간 복잡도로 빠른 검색, 삽입, 삭제를 지원합니다.

최악의 경우 O(n)의 시간이 걸림(해싱 충돌)

**컨테이너: C++**: map, **C#**: SortedDictionary<TKey, TValue>

**자료구조:** 레드-블랙 트리

**구조**:둘 다 이진 탐색 트리를 사용합니다.

**성능**: O(log n) 시간 복잡도로 비슷한 성능을 제공합니다.

**키의 순서**: 둘 다 키를 정렬된 상태로 유지합니다.

**순차적 접근**: 1,5, 3순으로 삽입해도, 저장할 땐 1, 3, 5 순으로 저장

**중복 키 방지**: 중복 삽입 시 오류

다른 **키 값**일때, Hash Function을 통해 같은 값을 가지게 되는 경우 해시 충돌(저장되는 곳에서 충돌)

해결방법: 체이닝(Chaining)

같은 주소로 해싱 될 때, 연결 리스트(Linked List)로 연결하는 방식->O(n)

**컨테이너: C++:** unordered\_set, **C#:** HashSet<T>

**자료구조:** 해시 테이블

**중복 배제**

**키의 순서**: 키의 순서를 유지하지 않습니다.

**성능**: 평균적으로 O(1) 시간 복잡도로 빠른 검색, 삽입, 삭제를 지원합니다.

최악의 경우 O(n)의 시간이 걸림(해싱 충돌)

**컨테이너: C++:** set, **C#:** SortedSet<T>

**자료구조:** 레드-블랙 트리

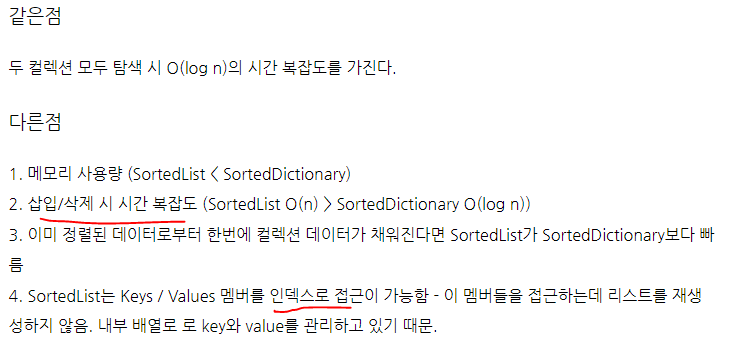
**중복 배제**: 중복을 허용하지 않습니다.

**성능**: O(log n) 시간 복잡도로 비슷한 성능을 제공합니다.

**자동 정렬**: 정렬된 상태로 유지합니다.

**컨테이너: C#**: SortedList<T> vs SortedDictionary<TKey, TValue>

**자료구조:** SortedList의배열



**스택**

**삽입**: O(1)

**삭제**: O(1)

**탐색**: O(n)

**큐**

**삽입**: O(1)

**삭제**: O(1)

**탐색**: O(n)