

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

**자료구조:** 동적 배열

**임의 접근**: (O(1))

**끝 추가/삭제**: 배열의 끝에서 이루어질 때 빠름 (O(1))

**중간 추가/삭제**: 비효율적 (O(n))

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

**자료구조:** 이중 연결 리스트

**순차 접근**: (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

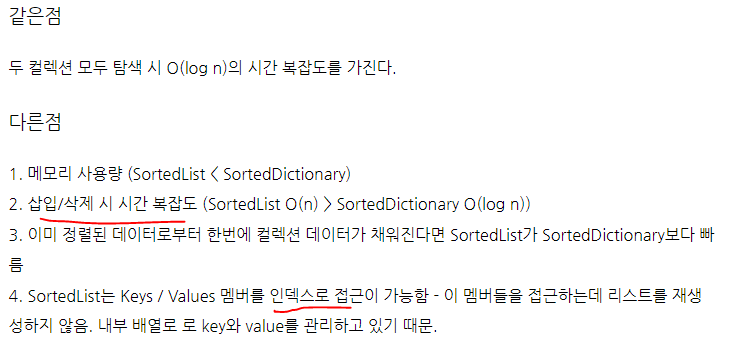
**중복 배제**: 중복을 허용하지 않습니다.

**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)

**값 타임 vs 참조 타입**

<https://hundredjunny.tistory.com/16>

메모리의 구조: 코드 영역, 데이터 영역, 스택 영역, 힙 영역(스택과 힙은 같은 공간을 사용함, 서로 견제하는 느낌)

값 타입은 스택 메모리에 저장돼요.

별도로 변경됨, 간단한 데이터, 데이터 불변성이 중요한 경우

Ex) **int, float, char, bool, 구조체**

참조 타입은 힙 메모리에 저장돼요.

같이 변경됨, 복잡한 데이터로 메모리 절약, 데이터 불변성이 상관 없는 경우

Ex) **클래스, 배열, 문자열**

**Awake vs Start**

**공통점**: 오브젝트 자체가 비활성화 돼있으면 작동 안함

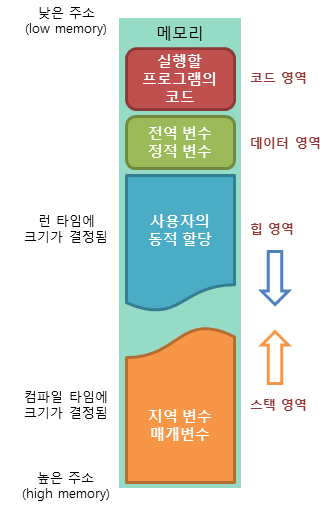
**Awake**는 스크립트만 비활성화 돼있을 때도 작동함, **start**는 작동 안함

**메모리에서 스택과 힙의 차이점**

[**https://junghyun100.github.io/%ED%9E%99-%EC%8A%A4%ED%83%9D%EC%B0%A8%EC%9D%B4%EC%A0%90**](https://junghyun100.github.io/%ED%9E%99-%EC%8A%A4%ED%83%9D%EC%B0%A8%EC%9D%B4%EC%A0%90)

**스택 영역:** 매개 변수, 지역 변수, 자동 해제됨, 메모리의 높은 주소에서 낮은 주소로 할당

**힙 영역:** 사용자가 관리하는 영역, 메모리의 낮은 주소에서 높은 주소로 할당



**Const와 ReadOnly의 차이**

<https://holjjack.tistory.com/95>

**const**

컴파일 타입의 상수이다.

메모리 할당 위치는 **Stack Memory**이다.

**readonly**

런타임 상수이다.

메모리 할당 위치는 **Heap Memory**이다.

**왜 const 보다 readonly가 더 좋은가?**

**const**는 **Stack Memory**에 저장 되기 때문에, 접근이 빠르다는 장점이 있지만,

**const**는 **컴파일 상수**이기 때문에, **const** 변수 값이 바뀌는 경우 해당 프로젝트 뿐만 아니라,

참조 받거나 영향을 받는 프로젝트 모두 재 컴파일을 해야 하는 단점이 있다.

**Delegate**: C++의 함수 포인터와 유사

**Event**: 이벤트를 호출 할 수 있는 것은 해당 이벤트를 가진 클래스, **Delegate**와 유사

비선점: FIFO, SJF, HRN(대기시간 + 서비스시간) / 서비스시간)

선점: RR, SRT

그 외: **기아**, **aging**