**<기본적인 자료구조들>**

1) 선형 구조 - 배열 - 연결 리스트 - 스택 - 큐

2) 비선형 구조 - 트리(Tree) - 그래프(Graph)

**시간복잡도: 알고리즘에 사용되는 연산 횟수**

**공간복잡도: 알고리즘에 사용되는 메모리 양**

효율적인 알고리즘을 사용한다고 가정했을 때 일반적으로 시간과 공간은 반비례 관계

**<시간 복잡도>**

**시간 복잡도는 Big-O 표기법을 사용함**

최악의 경우를 나타냄

* **O(𝑛) 시간 복잡도**

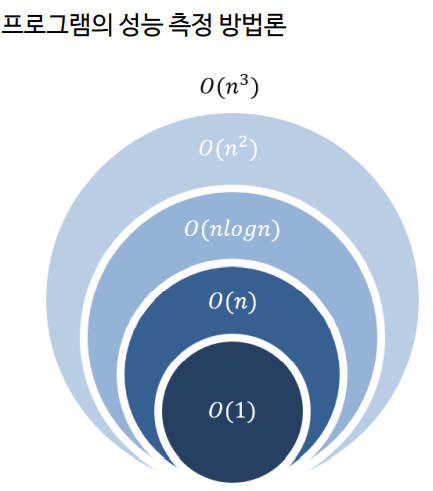
텍스트이(가) 표시된 사진

자동 생성된 설명

**O(n\*\*2)의 시간 복잡도 -> 이중 for문**

텍스트이(가) 표시된 사진

자동 생성된 설명

 텍스트이(가) 표시된 사진

자동 생성된 설명

왠만하면 10억번 이상이 아니어야함.

시간 복잡도를 표기할 때는 항상 큰 항과 계수만 표

𝑂(3𝑛^2 + 𝑛) = 𝑂(𝑛^2 )

현실의 다양한 문제에서는 시간 제한이 1초정도

<공간 복잡도>

공간 복잡도를 표기할 때는 일반적으로 MB 단위로 표기함.

텍스트이(가) 표시된 사진

자동 생성된 설명

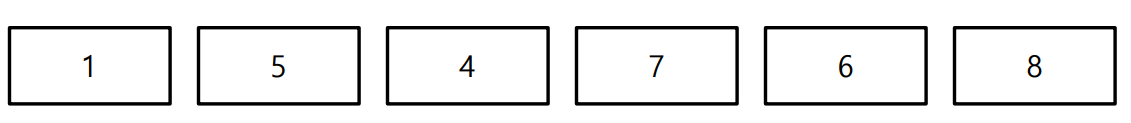
<연결 리스트>

연결 리스트의 필요성

1) 일반적으로 배열을 사용하여 데이터를 순차적으로 저장하고, 나열할 수 있음.

2) 배열을 사용하는 경우 메모리 공간이 불필요하게 낭비될 수 있음.

<배열 기반의 리스트>



1) 배열로 만들었으므로 특정한 위치의 원소에 즉시 접근할 수 있다는 장점이 있습니다.

2) 데이터가 들어갈 공간을 미리 메모리에 할당해야 한다는 단점이 있습니다.

3) 원하는 위치로의 삽입이나 삭제가 비효율적입니다.

연결 리스트

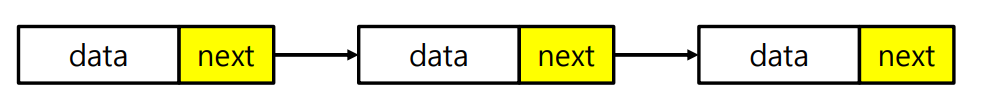
1) **일반적으로 연결 리스트는 구조체와 포인터를 함께 사용하여 구현**합니다.

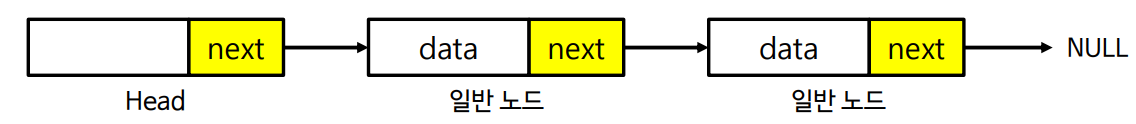
2) 연결 리스트는 **리스트의 중간 지점에 노드를 추가하거나 삭제할 수 있어야 합니다.**

3) **필요할 때마다 메모리 공간을 할당 받습니다. 동적 할당**

**<단일 연결리스트>**

포인터를 이용해 단방향적으로 다음 노드를 가리킴

****

****

1) 단일 연결 리스트는 다음과 같은 형태로 나타낼 수 있습니다.

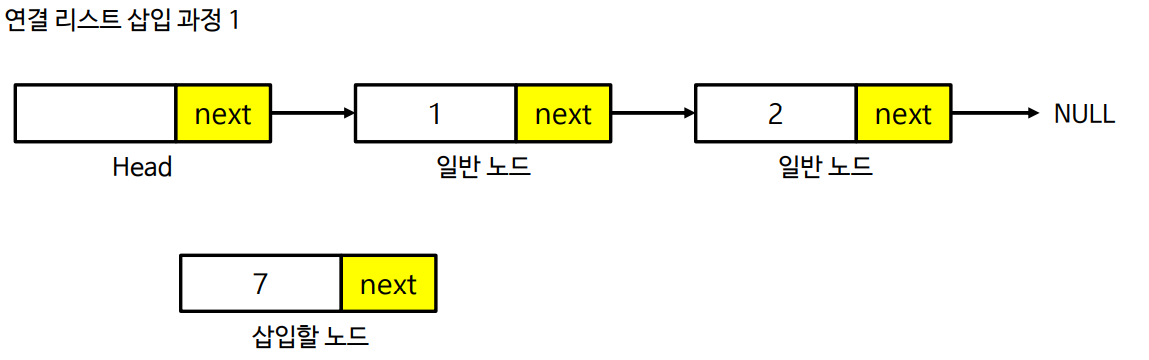
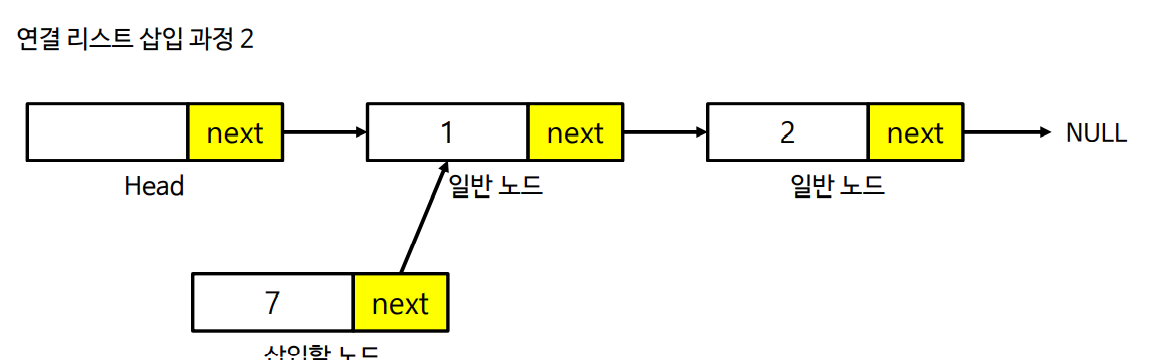
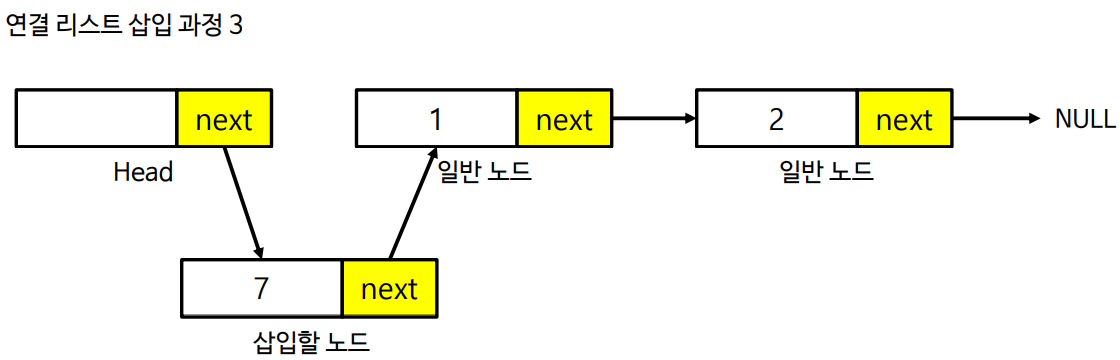
2) 포인터를 이용해 단방향적으로 다음 노드를 가리킵니다.

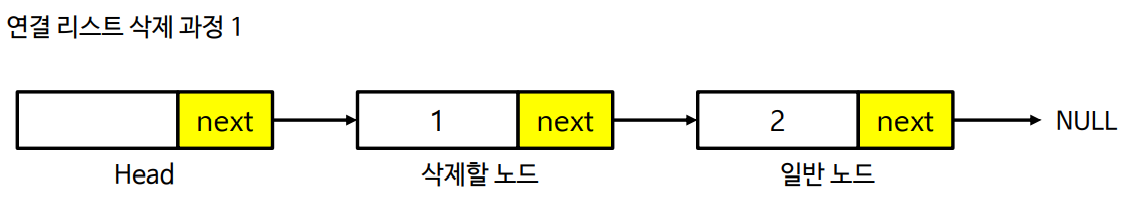
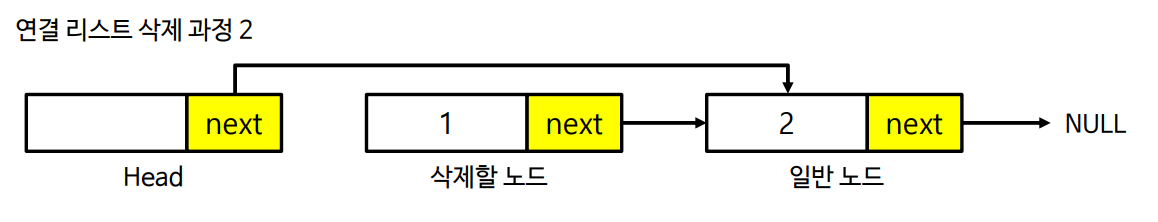
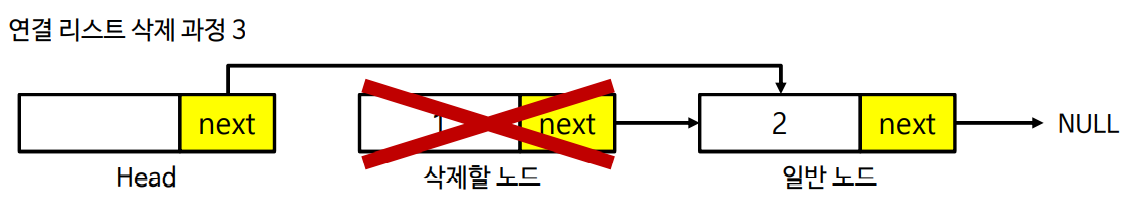
3) 일반적으로 연결 리스트의 **시작 노드를 헤드(Head)**라고 하며 별도로 관리합니다.

4) 다음 노드가 없는 **끝 노드의 다음 위치 값으로는 NULL을 넣습니다.**

단일 연결리스트에는 하나의 구조체 안에 두 개의 변수가 들어감.

하나는 데이터, 하나는 다음 데이터를 가르킴.

<연결 리스트 구현에 있어서 주의할 점>

1) 위 소스코드에 덧붙여서 삽입 및 삭제 기능에서의 예외 사항을 처리할 필요가 있음

2) 삭제할 원소가 없는데 삭제하는 경우, 머리(Head) 노드 자체를 잘못 넣은 경우 등을 체크

<특징>

1) 삽입과 삭제가 배열에 비해서 간단하다는 장점이 있음

2) 배열과 다르게 특정 인덱스로 즉시 접근하지 못하며, 원소를 차례대로 검색해야 함.

3) 추가적인 포인터 변수가 사용되므로 메모리 공간이 낭비됨.