Longest Decreasing Sequence

최장 감소 수열

LDS?

- 가장 긴 감소하는 부분 수열

LDS?

- 가장 긴 감소하는 부분 수열

Q. arr에서 최장 감소 수열을 구하면?

arr	30	40	5	15	10	20

LDS?

- 가장 긴 감소하는 부분 수열

Q. arr에서 최장 감소 수열을 구하면?

arr	30	40	5	15	10	20

arr	30	40	5	15	10	20

LDS?

- 가장 긴 감소하는 부분 수열

Q. arr에서 최장 감소 수열을 구하면?

arr	30	40	5	15	10	20

arr	30	40	5	15	10	20

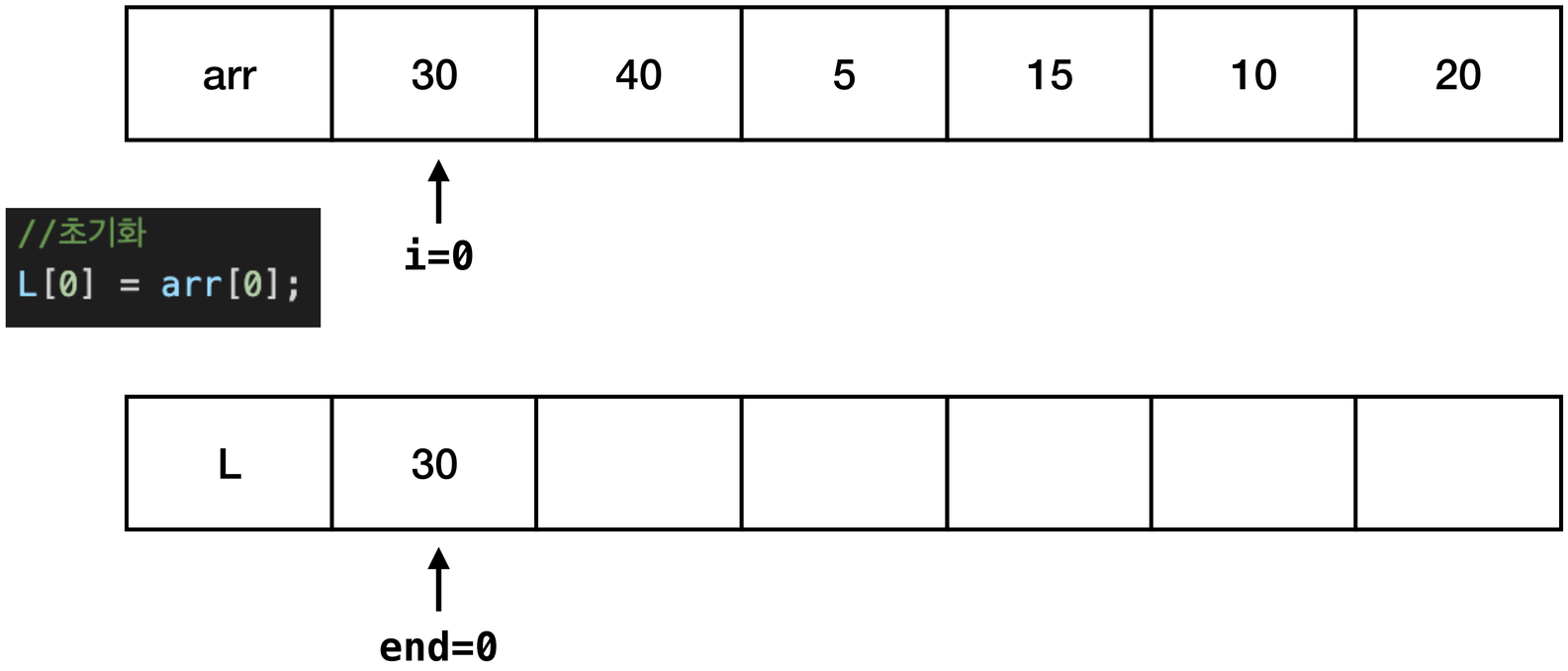
코드를 어떻게 짜느냐에 따라 {40,15,10} 일수도, {30,15,10} 일수도 있음.
이번엔 {40,15,10} 경우를 볼게요

여러분은 똑똑하니까 O(n^2)짜리 알고리즘 따윈 LIS에서 만족하고 바로 이분탐색 넘어갈게요

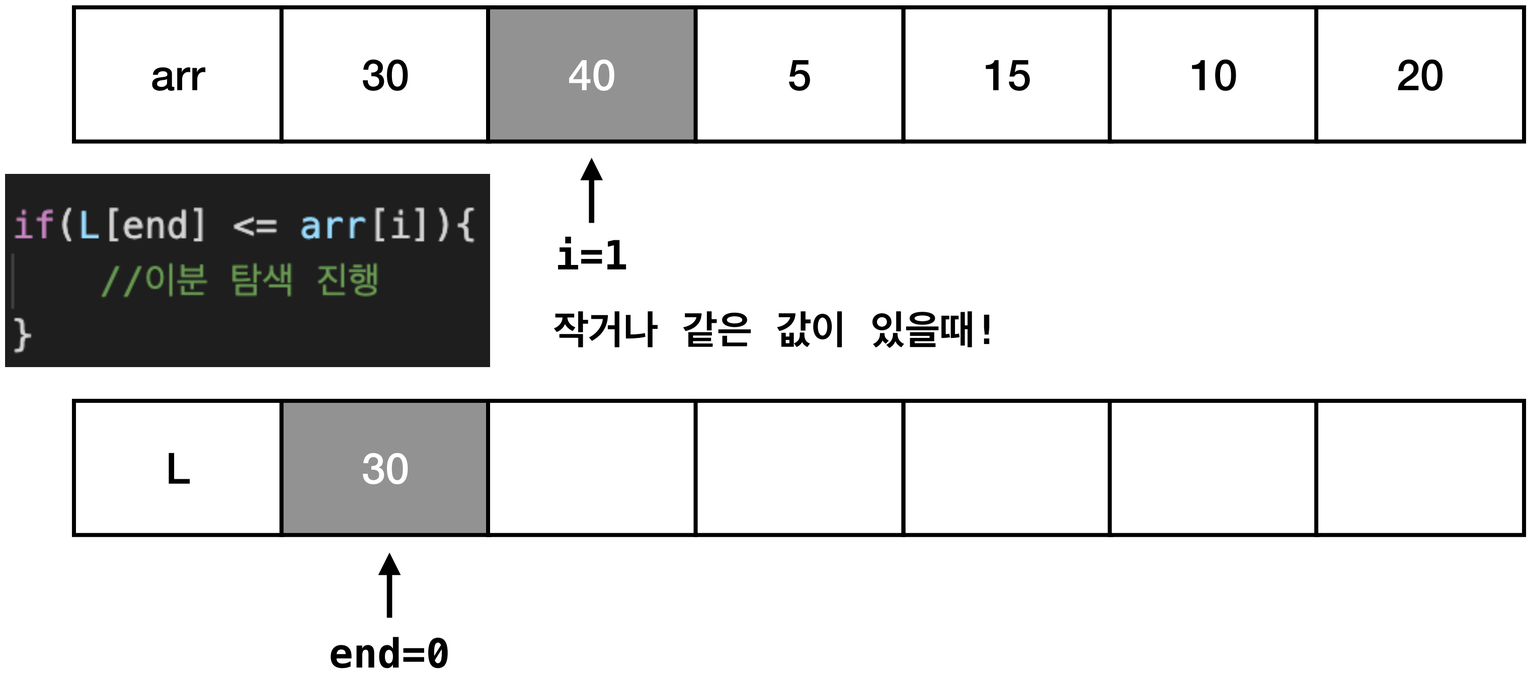
LDS?

- `LDS[0] == arr[0]`
- 작거나 같은 값이 있으면 이분탐색 후 해당 위치에 값을 갱신
- 들어오는 값이 가장 작은 경우에는 배열의 맨 뒤에 저장

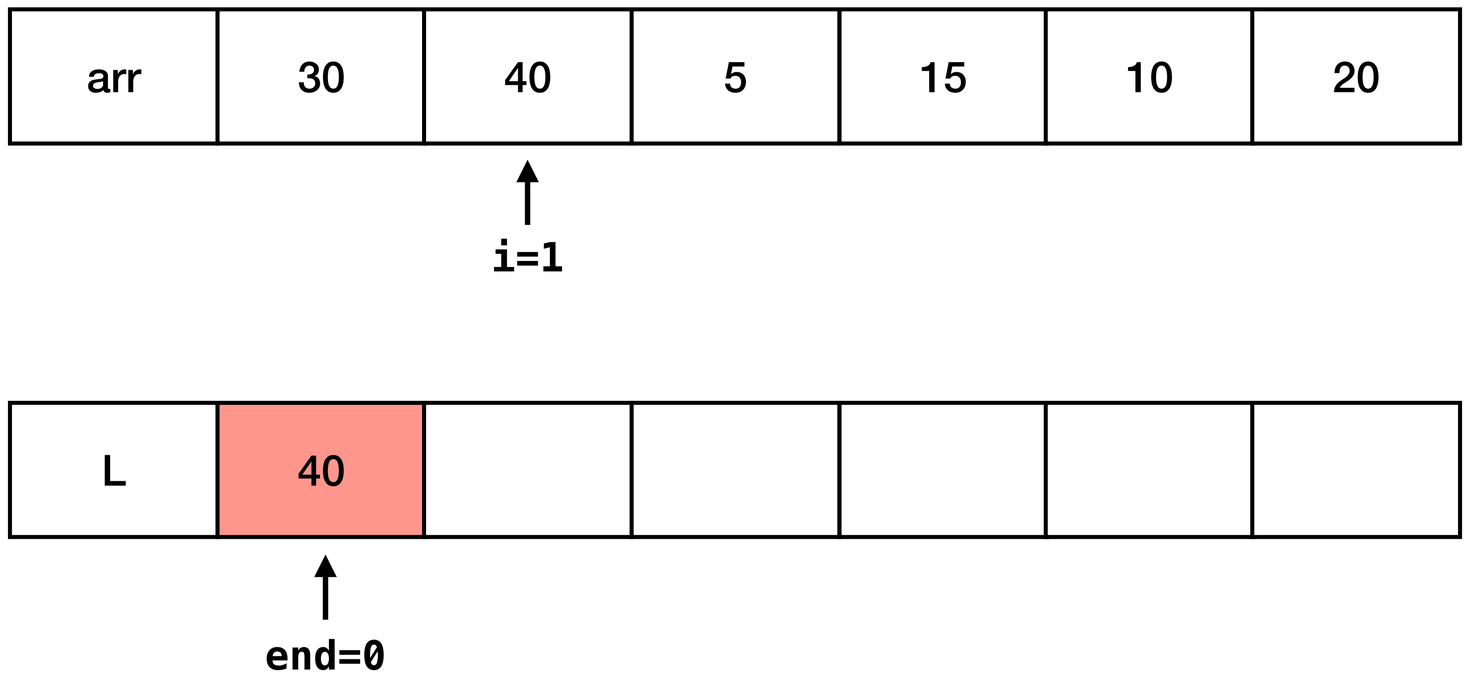
이분탐색으로 LDS 구현하기



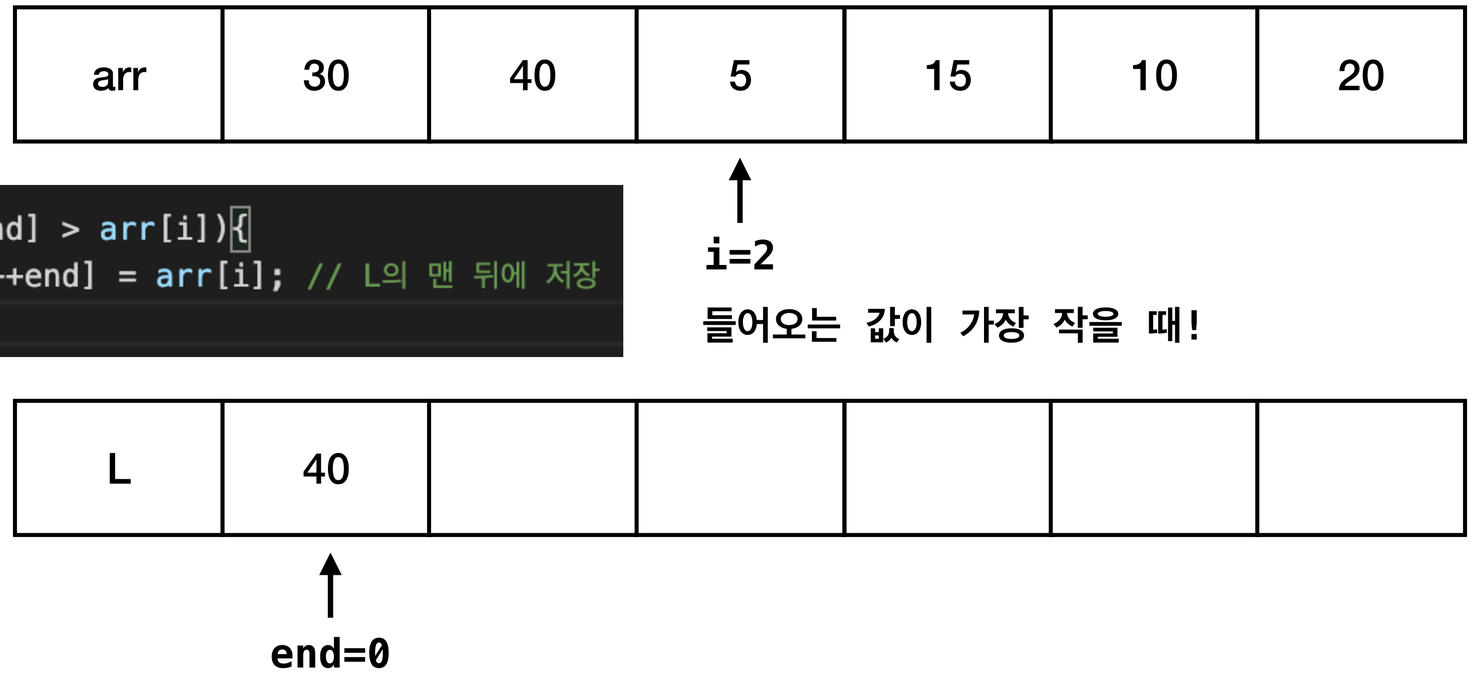
이분탐색으로 LDS 구현하기



이분탐색으로 LDS 구현하기

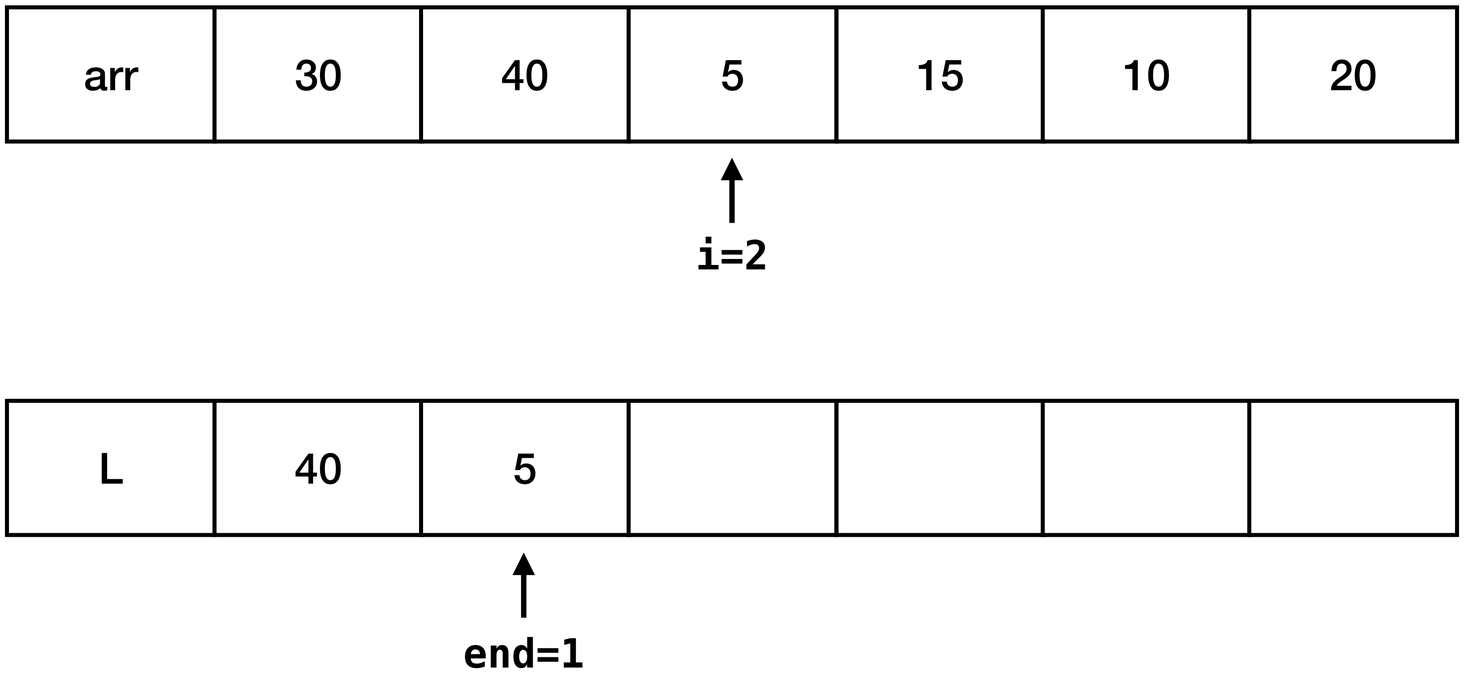


이분탐색으로 LDS 구현하기

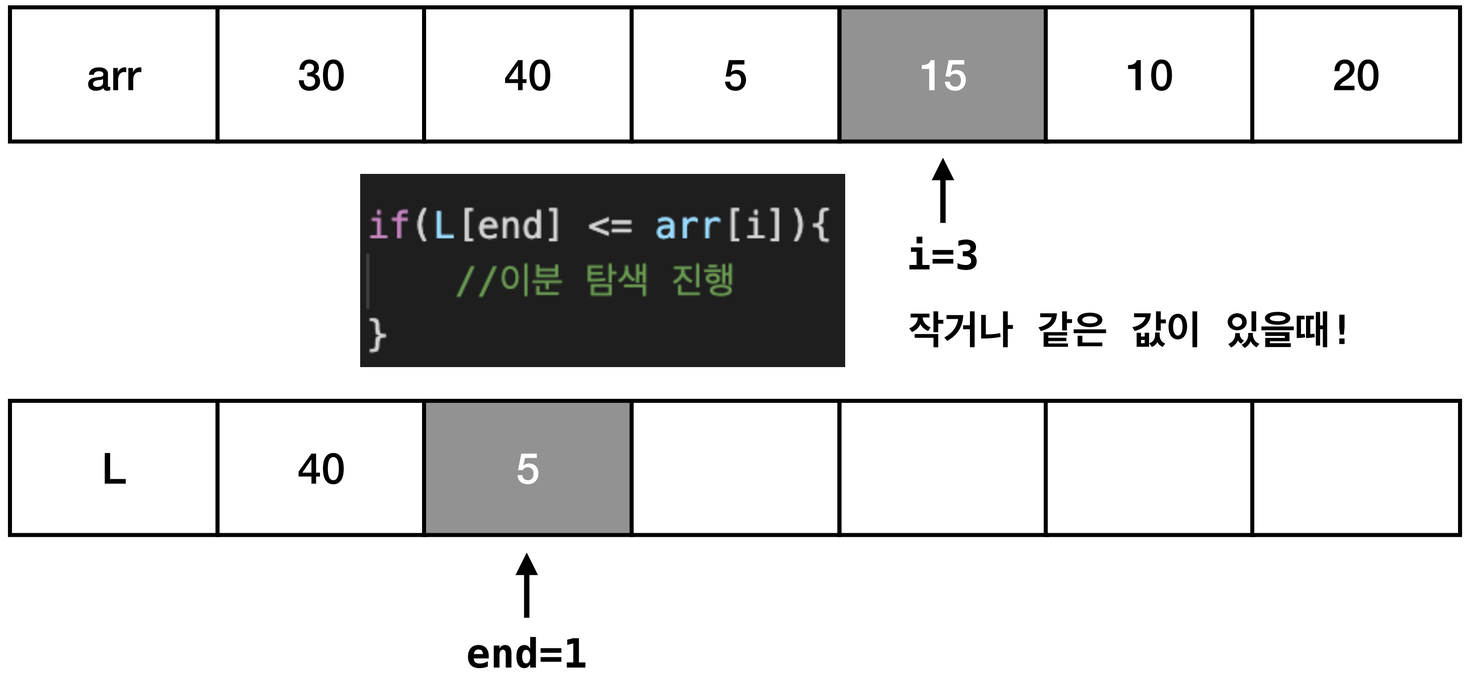



```
if(L[end] > arr[i]){
    L[++end] = arr[i]; // L의 맨 뒤에 저장
}
```

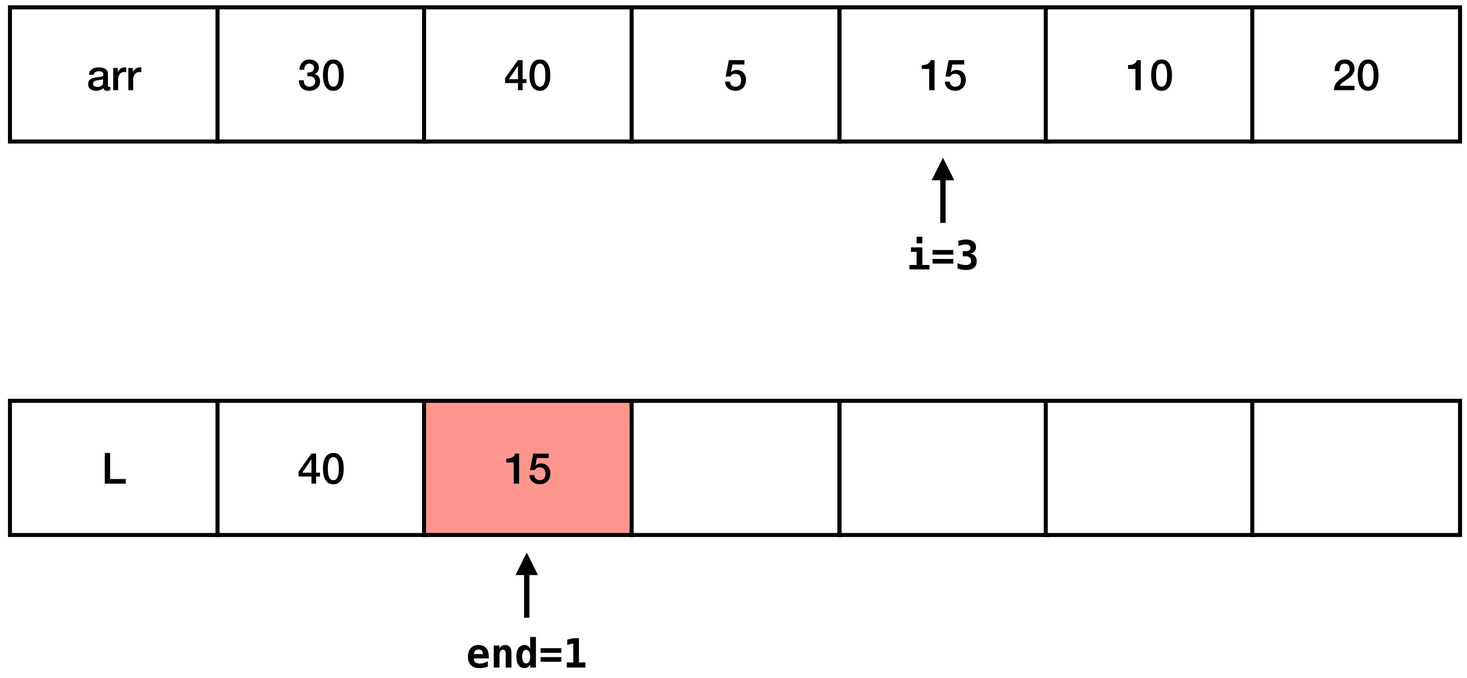
이분탐색으로 LDS 구현하기



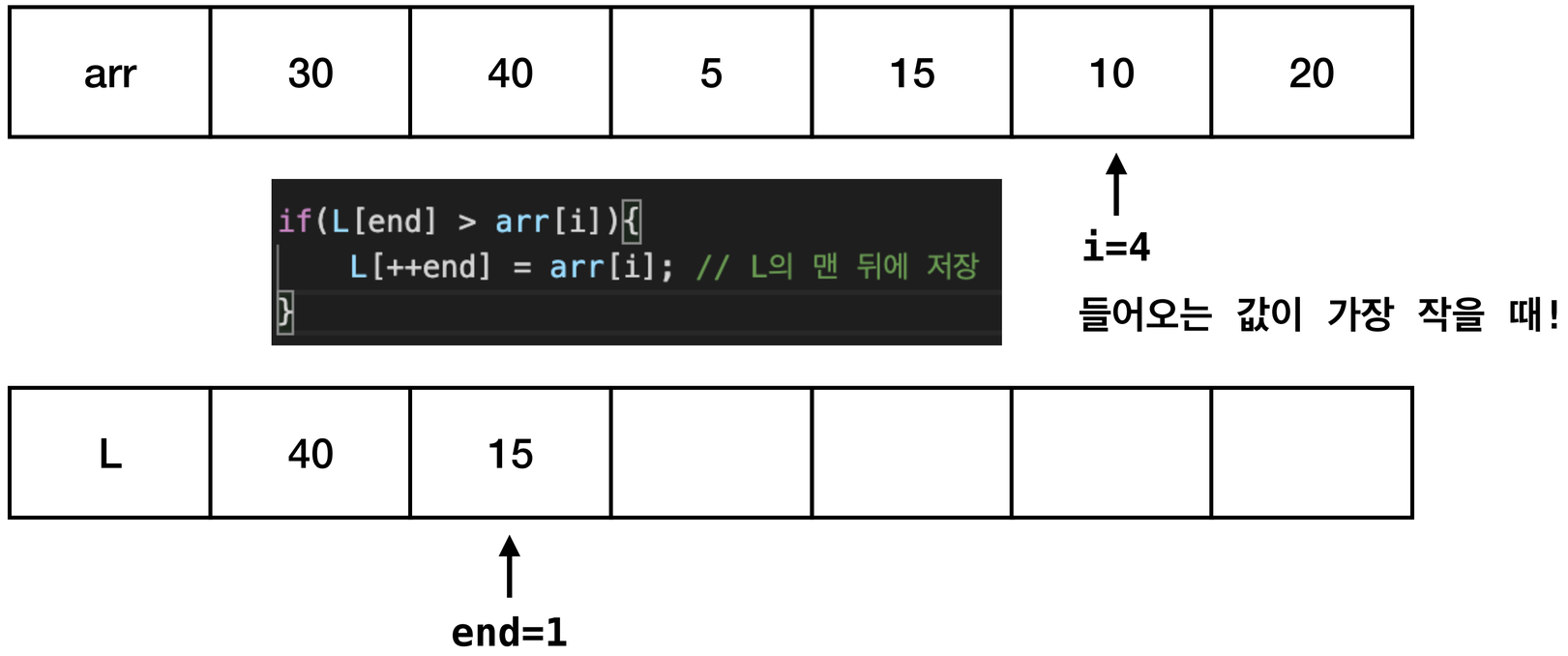
이분탐색으로 LDS 구현하기



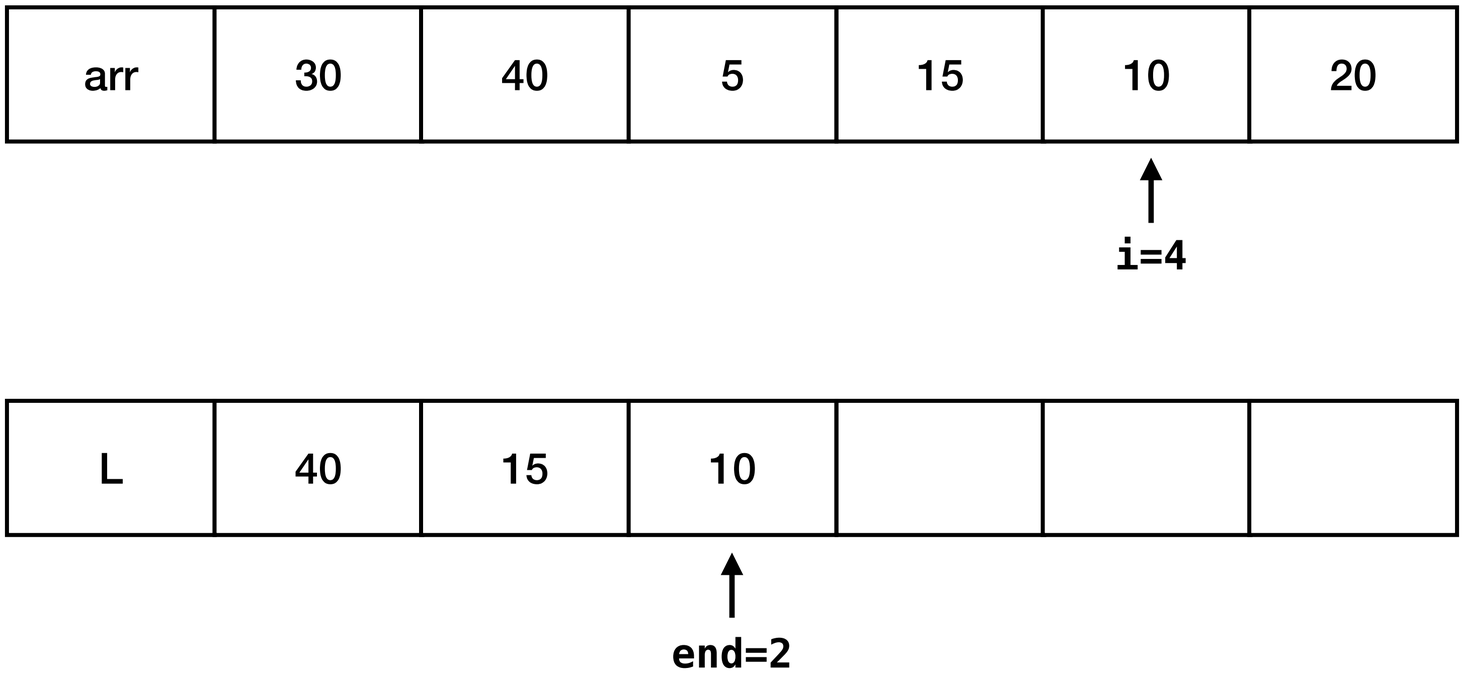
이분탐색으로 LDS 구현하기



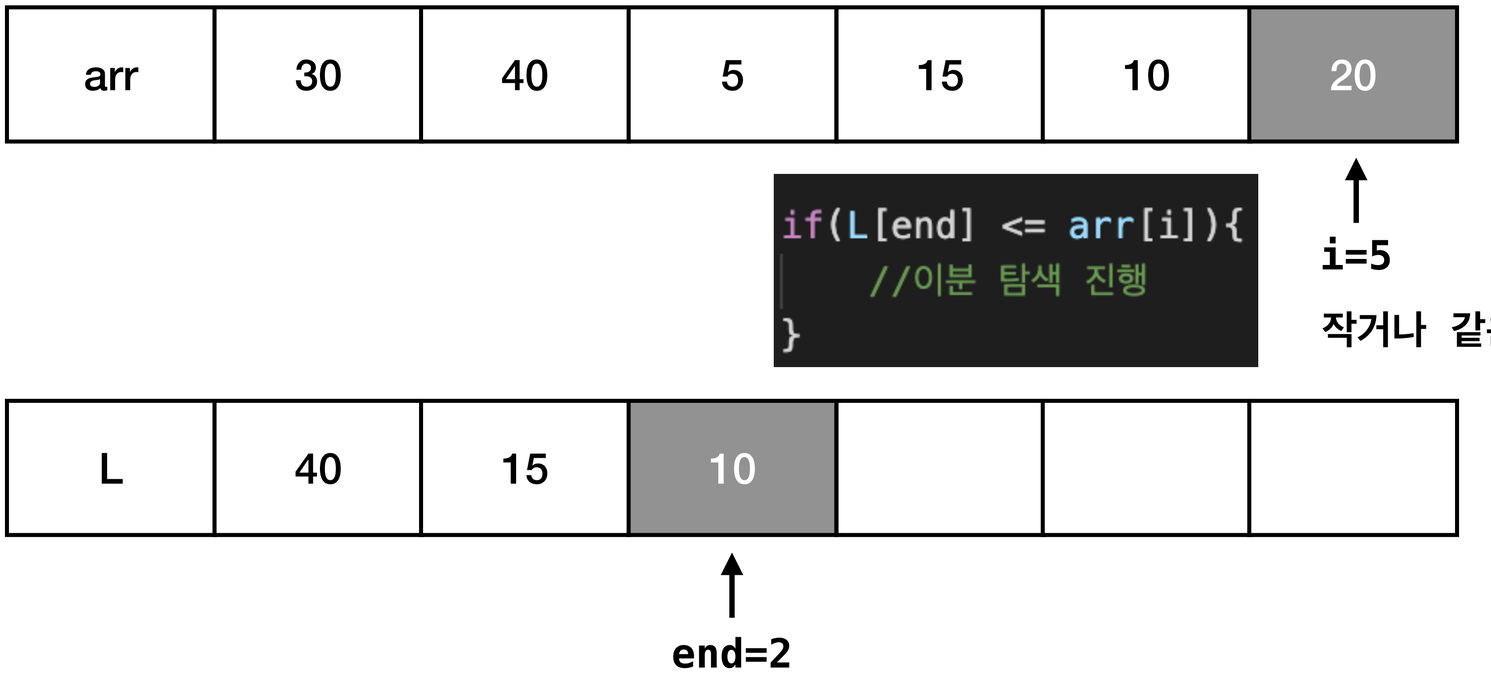
이분탐색으로 LDS 구현하기



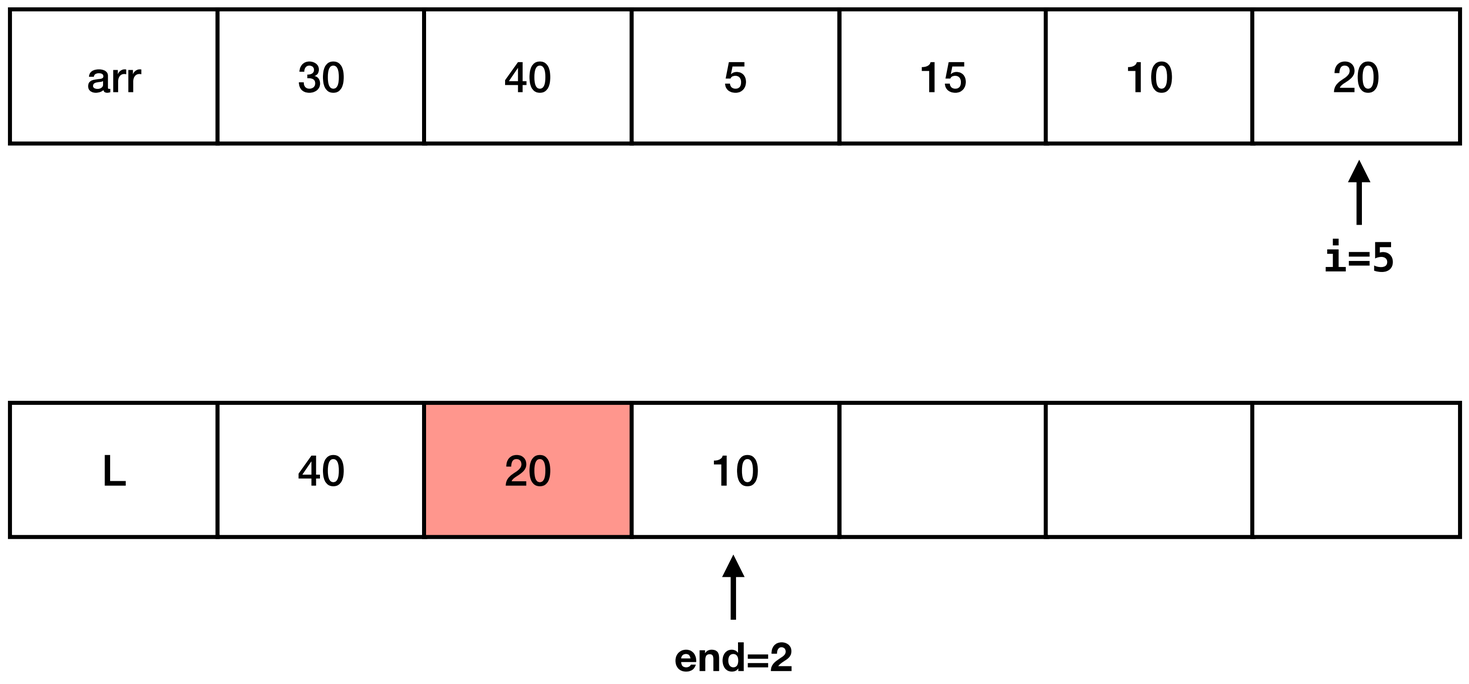
이분탐색으로 LDS 구현하기



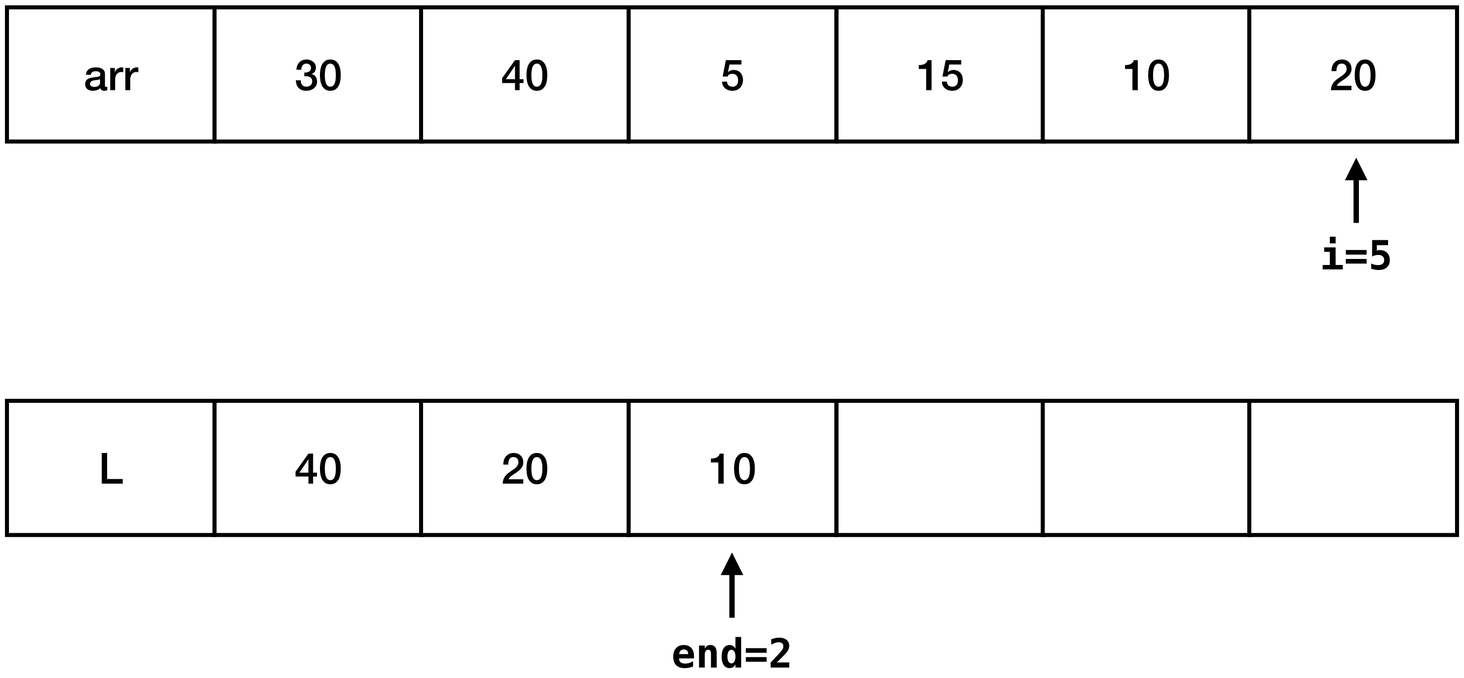
이분탐색으로 LDS 구현하기



이분탐색으로 LDS 구현하기

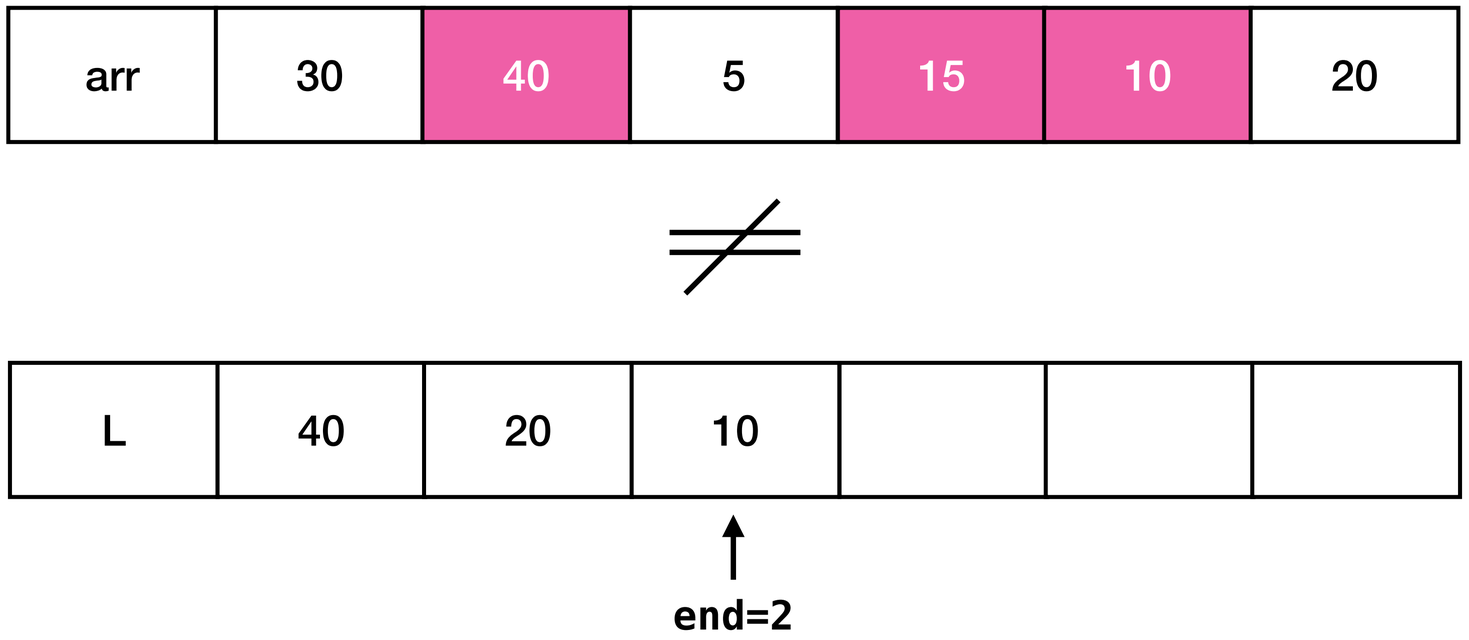


이분탐색으로 LDS 구현하기



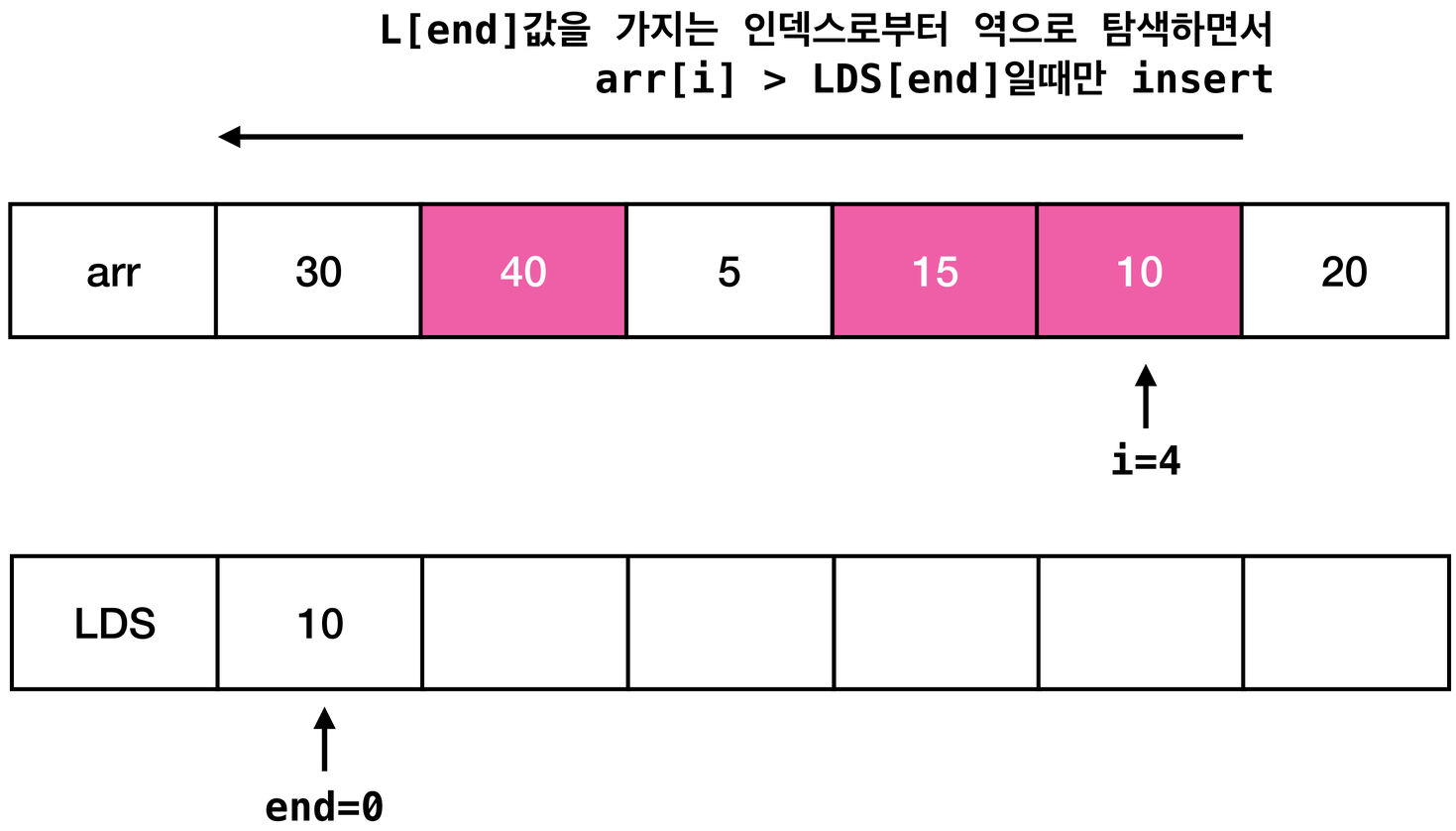
이렇게 하여 최장 감소 수열의 길이는 3이 됩니다.
(참고 : end + 1 = 3)

이분탐색으로 LDS 구현하기

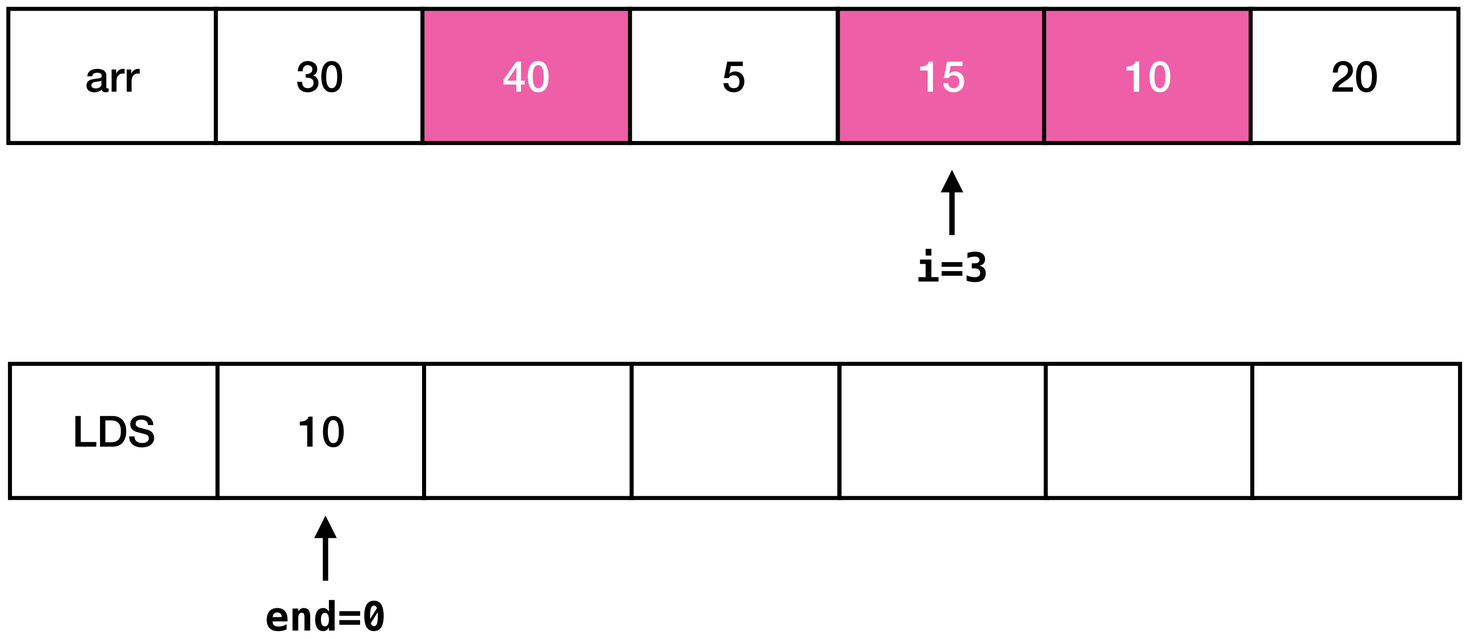


엥 근데 내가 찾는 LDS는 아닌데요?
L은 LDS의 길이를 알아보는 용도일뿐, LDS을 만족하는건 아님

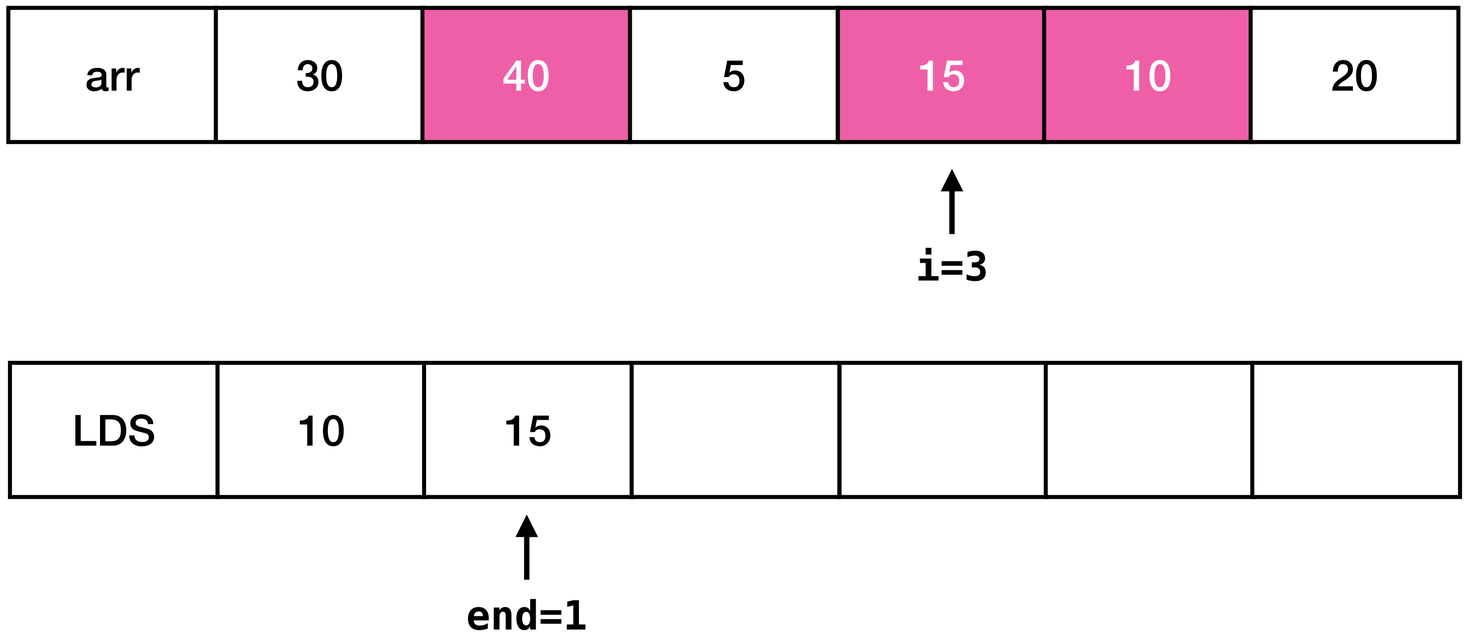
이분탐색으로 LDS 구현하기



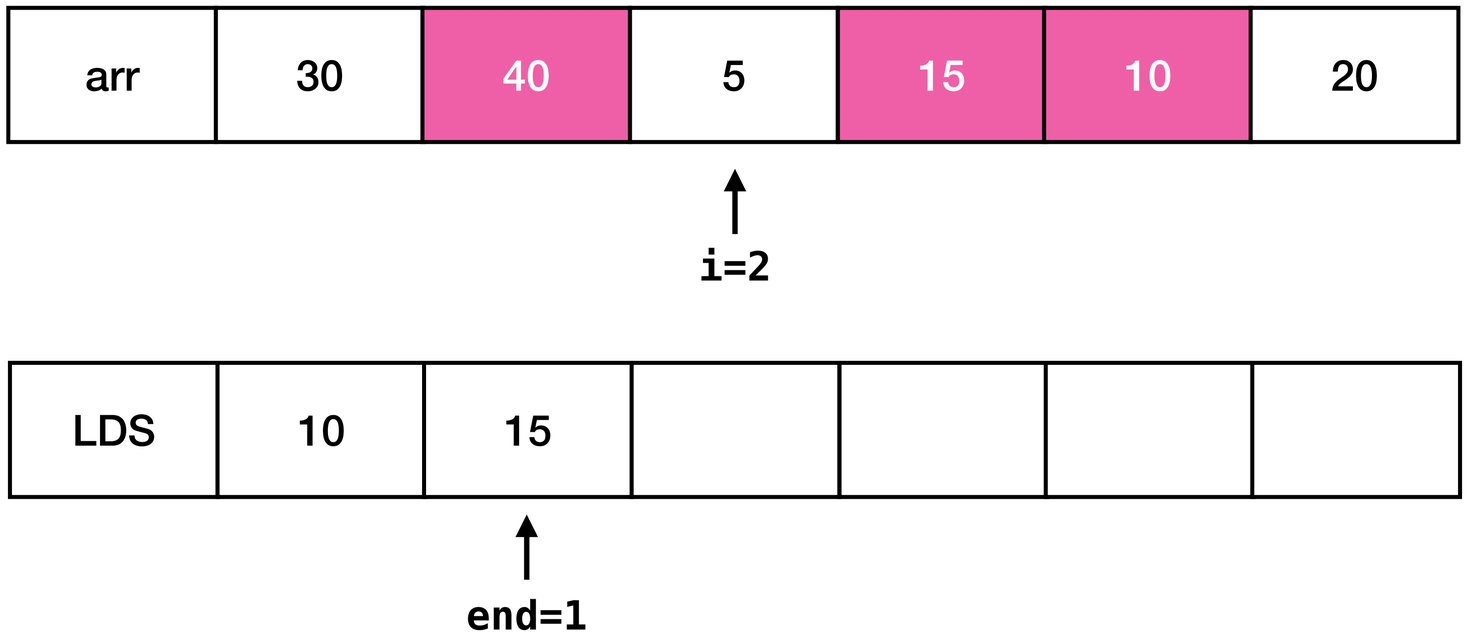
이분탐색으로 LDS 구현하기



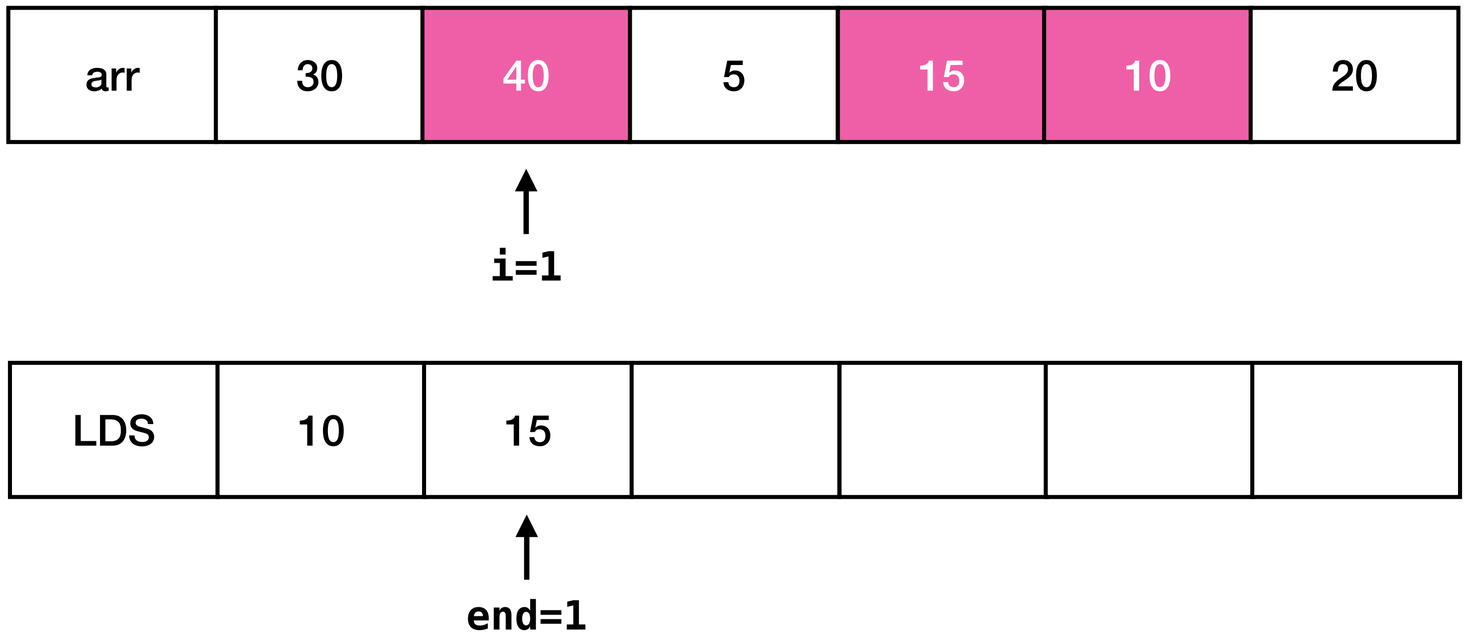
이분탐색으로 LDS 구현하기



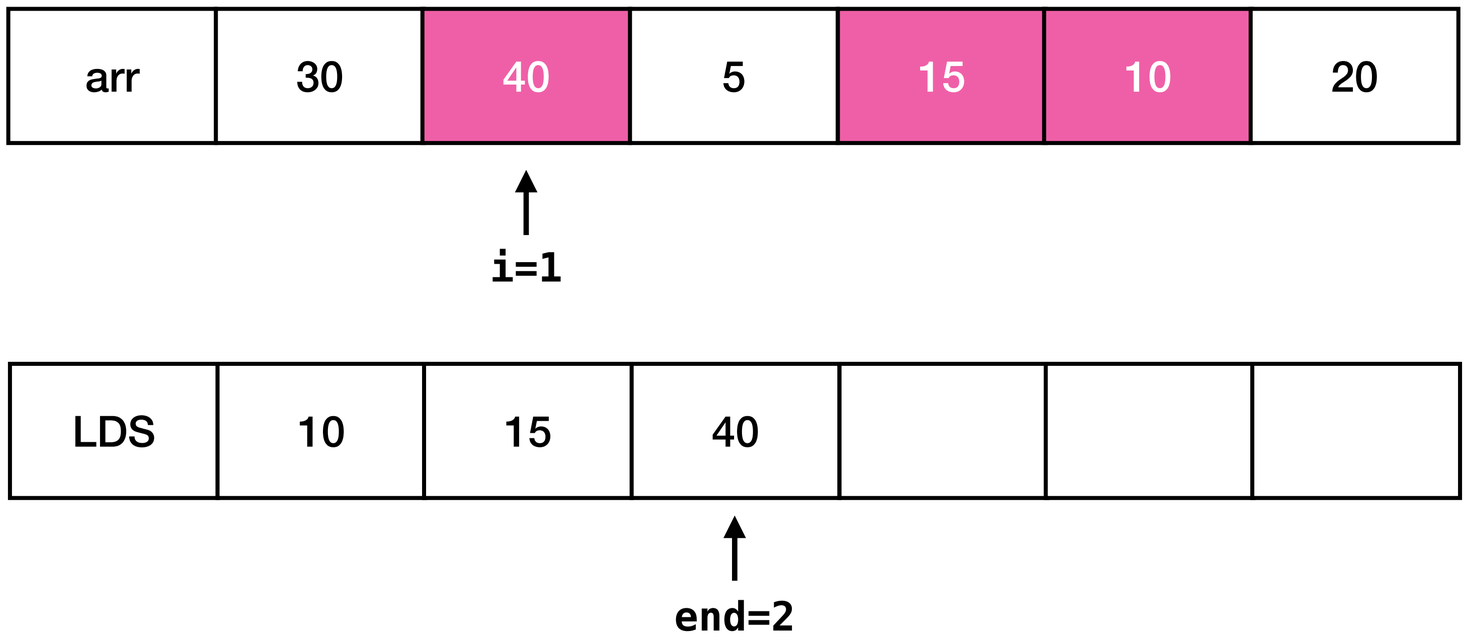
이분탐색으로 LDS 구현하기



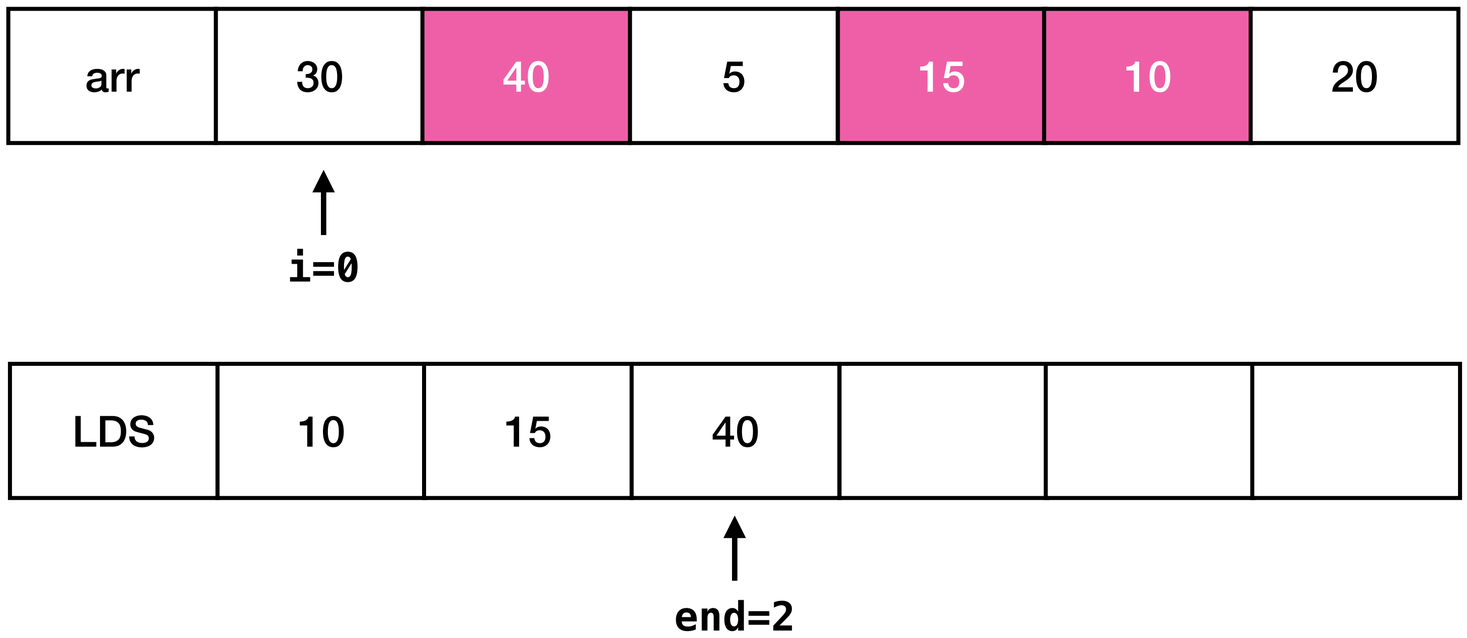
이분탐색으로 LDS 구현하기



이분탐색으로 LDS 구현하기



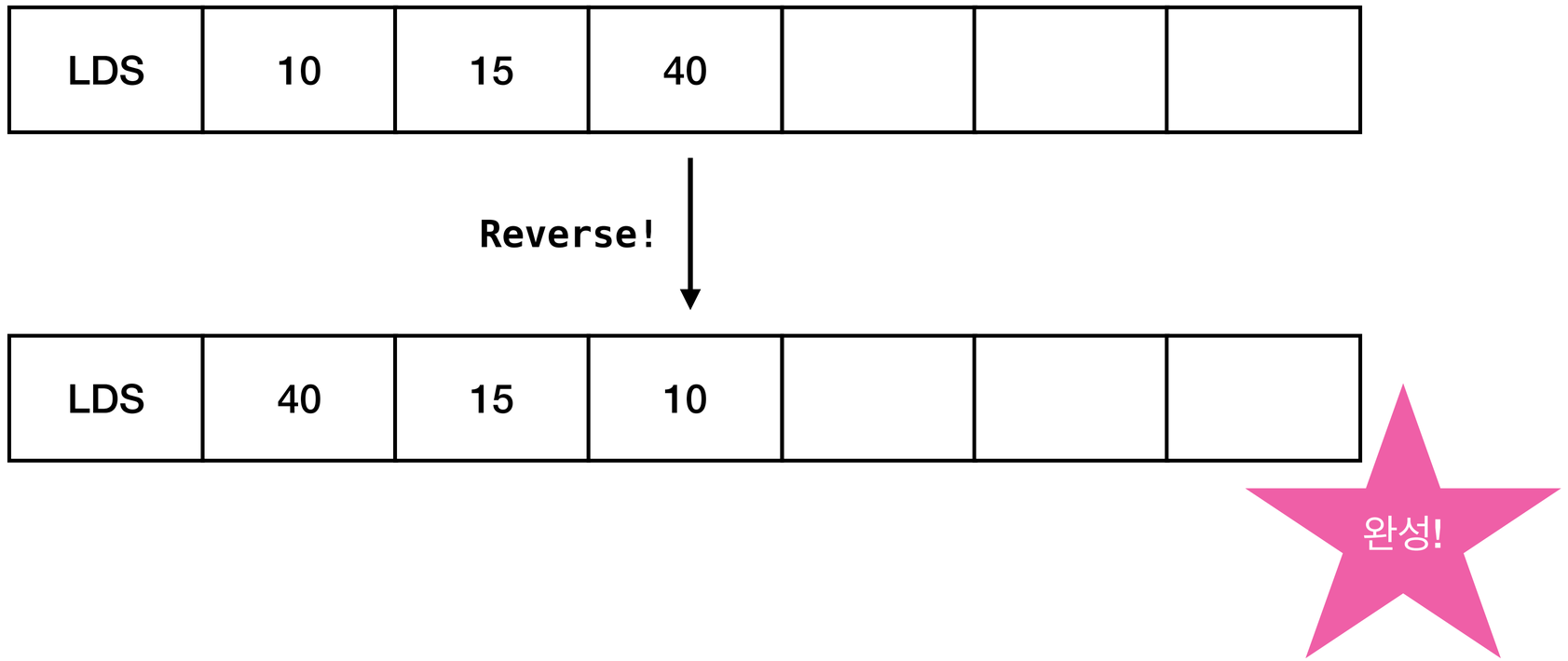
이분탐색으로 LDS 구현하기



이분탐색으로 LDS 구현하기

LDS	10	15	40			

이분탐색으로 LDS 구현하기

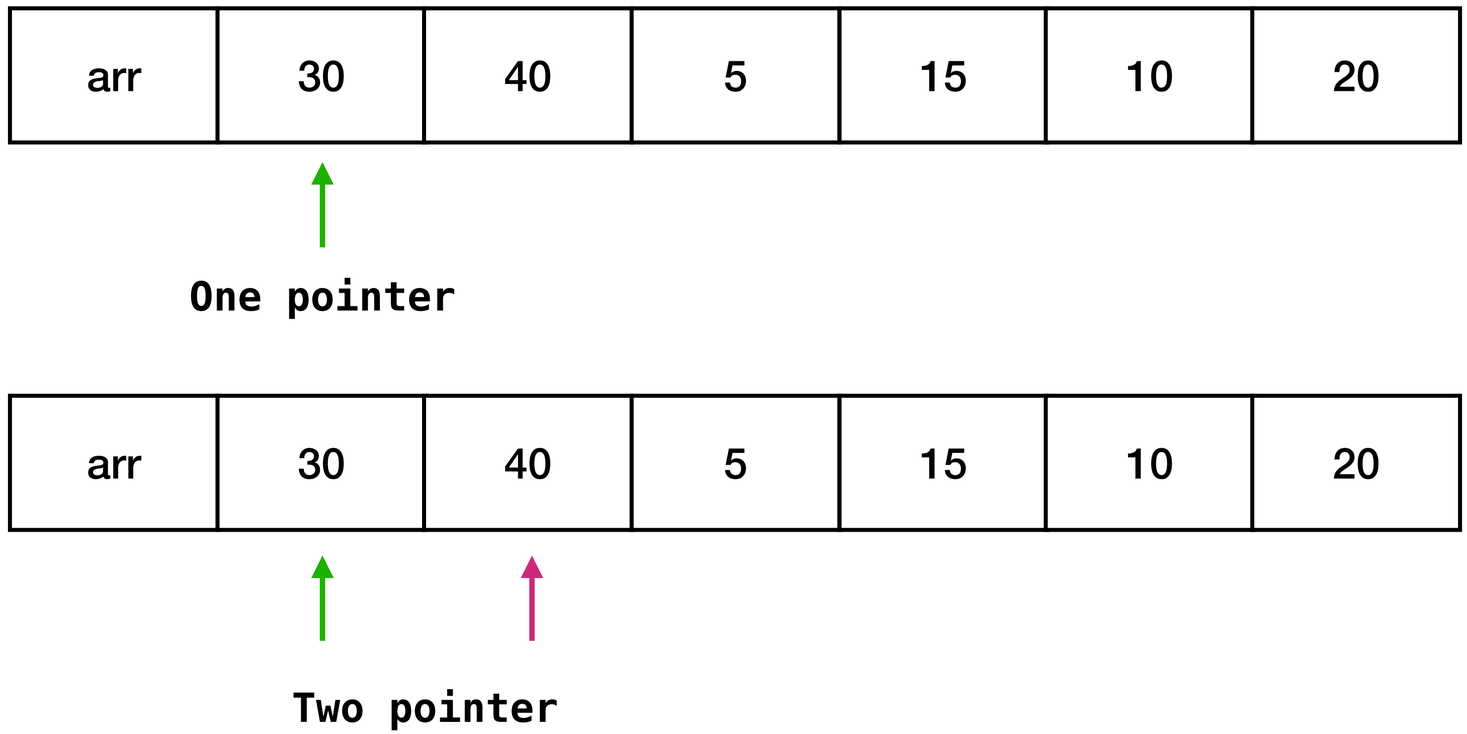


Two Pointer

투 포인터

Two pointer?

- **정렬된 배열**에서 **두 개의 포인터**를 이용해 포인터를 조작하며 문제를 푸는 기법 (인덱스에 접근하는 iterator가 두개있다고 생각하면 될듯)



Two pointer?

Q1. 리스트 arr와 타켓 S가 주어질때, 두 수의 합이 S가 되는 순서쌍을 구하여라.

Q2. 리스트 arr이 주어질때 연속된 수들의 부분합 중 합이 S이상이 되는 것의 개수는?

등등… Q2는 문제로 낼거라 Q1살펴봅시다

A1 : 완전탐색

Q1. 리스트 arr와 타켓 S가 주어질때, 두 수의 합이 S가 되는 순서쌍을 구하라

S = 27

arr = [1, 3, 5, 6, 9, 11, 12, 16, 17, 19, 22, 25, 28]

A1 : 완전탐색

Q1. 리스트 arr와 타켓 S가 주어질때, 두 수의 합이 S가 되는 순서쌍을 구하라

S = 27

arr = [1, 3, 5, 6, 9, 11, 12, 16, 17, 19, 22, 25, 28]

```
for(int i=0; i<arr.size(); i++)
    for(int j=i+1; j<arr.size(); j++)
        if(arr[i] + arr[j] == S)
            cout<<arr[i]<<", "<<arr[j]<<" : S";
```

A1 : 완전탐색

Q1. 리스트 arr와 타켓 S가 주어질때, 두 수의 합이 S가 되는 순서쌍을 구하라

S = 27

arr = [1, 3, 5, 6, 9, 11, 12, 16, 17, 19, 22, 25, 28]

```
for(int i=0; i<arr.size(); i++)
    for(int j=i+1; j<arr.size(); j++)
        if(arr[i] + arr[j] == S)
            cout<<arr[i]<<", "<<arr[j]<<" : S";
```

O(N^2) => 효율성이 안좋음

A2 : 투포인터

Q1. 리스트 arr와 타켓 S가 주어질때, 두 수의 합이 S가 되는 순서쌍을 구하라

S = 27

arr = [1, 3, 5, 6, 9, 11, 12, 16, 17, 19, 22, 25, 28]

1. 리스트를 오름차순으로 정렬한다.
2. 포인터 `left`는 리스트의 시작점을, 포인터 `right`는 리스트의 끝점을 가리킨다.
3. `left`와 `right`가 만날 때 까지(즉, 모든 경우를 확인할 때 까지) 다음을 반복한다.
   3-1. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)과 같다면 조건을 만족하므로 출력하고, `left`를 1 증가, `right`를 1 감소시킨다.
   3-2. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 작다면 `left`를 1 증가시킨다.
   3-3. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 크다면 `right`를 1 감소시킨다.

A2 : 투포인터

```
S = 27
arr = [1,  3,  5,  6,  9,  11,  12,  16,  17,  19,  22,  25,  28]
```

1. 리스트를 오름차순으로 정렬한다.
2. 포인터 `left`는 리스트의 시작점을, 포인터 `right`는 리스트의 끝점을 가리킨다.
3. `left`와 `right`가 만날 때 까지(즉, 모든 경우를 확인할 때 까지) 다음을 반복한다.
   - 3-1. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)과 같다면 조건을 만족하므로 출력하고, `left`를 1 증가, `right`를 1 감소시킨다.
   - 3-2. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 작다면 `left`를 1 증가시킨다.
   - 3-3. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 크다면 `right`를 1 감소시킨다.

A2 : 투포인터

```
S = 27
arr = [1,  3,  5,  6,  9,  11,  12,  16,  17,  19,  22,  25,  28]
       ↑                                                   ↑
      Left                                               Right
```

1. 리스트를 오름차순으로 정렬한다.
2. 포인터 `left`는 리스트의 시작점을, 포인터 `right`는 리스트의 끝점을 가리킨다.
3. `left`와 `right`가 만날 때 까지(즉, 모든 경우를 확인할 때 까지) 다음을 반복한다.
   - 3-1. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)과 같다면 조건을 만족하므로 출력하고, `left`를 1 증가, `right`를 1 감소시킨다.
   - 3-2. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 작다면 `left`를 1 증가시킨다.
   - 3-3. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 크다면 `right`를 1 감소시킨다.

A2 : 투포인터

```
S = 27
arr = [1,  3,  5,  6,  9,  11,  12,  16,  17,  19,  22,  25,  28]
          ↑                                                  ↑
        Left                                              Right
```

1. 리스트를 오름차순으로 정렬한다.
2. 포인터 `left`는 리스트의 시작점을, 포인터 `right`는 리스트의 끝점을 가리킨다.
3. `left`와 `right`가 만날 때 까지(즉, 모든 경우를 확인할 때 까지) 다음을 반복한다.
   - 3-1. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)과 같다면 조건을 만족하므로 출력하고, `left`를 1 증가, `right`를 1 감소시킨다.
   - 3-2. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 작다면 `left`를 1 증가시킨다.
   - 3-3. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 크다면 `right`를 1 감소시킨다.

A2 : 투포인터

```
S = 27
arr = [1,  3,  5,  6,  9,  11,  12,  16,  17,  19,  22,  25,  28]
```

1. 리스트를 오름차순으로 정렬한다.
2. 포인터 `left`는 리스트의 시작점을, 포인터 `right`는 리스트의 끝점을 가리킨다.
3. `left`와 `right`가 만날 때 까지(즉, 모든 경우를 확인할 때 까지) 다음을 반복한다.
   - 3-1. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)과 같다면 조건을 만족하므로 출력하고, `left`를 1 증가, `right`를 1 감소시킨다.
   - 3-2. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 작다면 `left`를 1 증가시킨다.
   - 3-3. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 크다면 `right`를 1 감소시킨다.

A2 : 투포인터

```
S = 27
arr = [1,  3,  5,  6,  9,  11,  12,  16,  17,  19,  22,  25,  28]
```

Left

Right

1. 리스트를 오름차순으로 정렬한다.
2. 포인터 `left`는 리스트의 시작점을, 포인터 `right`는 리스트의 끝점을 가리킨다.
3. `left`와 `right`가 만날 때 까지(즉, 모든 경우를 확인할 때 까지) 다음을 반복한다.
   - 3-1. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)과 같다면 조건을 만족하므로 출력하고, `left`를 1 증가, `right`를 1 감소시킨다.
   - 3-2. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 작다면 `left`를 1 증가시킨다.
   - 3-3. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 크다면 `right`를 1 감소시킨다.

A2 : 투포인터

```
S = 27
arr = [1,  3,  5,  6,  9,  11,  12,  16,  17,  19,  22,  25,  28]
```

1. 리스트를 오름차순으로 정렬한다.
2. 포인터 `left`는 리스트의 시작점을, 포인터 `right`는 리스트의 끝점을 가리킨다.
3. `left`와 `right`가 만날 때 까지(즉, 모든 경우를 확인할 때 까지) 다음을 반복한다.
   - 3-1. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)과 같다면 조건을 만족하므로 출력하고, `left`를 1 증가, `right`를 1 감소시킨다.
   - 3-2. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 작다면 `left`를 1 증가시킨다.
   - 3-3. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 크다면 `right`를 1 감소시킨다.

A2 : 투포인터

```
S = 27
arr = [1,  3,  5,  6,  9,  11,  12,  16,  17,  19,  22,  25,  28]
                                  ↑
                                Left
                                  ↑
                                Right
```

1. 리스트를 오름차순으로 정렬한다.
2. 포인터 `left`는 리스트의 시작점을, 포인터 `right`는 리스트의 끝점을 가리킨다.
3. `left`와 `right`가 만날 때 까지(즉, 모든 경우를 확인할 때 까지) 다음을 반복한다.
   - 3-1. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)과 같다면 조건을 만족하므로 출력하고, `left`를 1 증가, `right`를 1 감소시킨다.
   - 3-2. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 작다면 `left`를 1 증가시킨다.
   - 3-3. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 크다면 `right`를 1 감소시킨다.

A2 : 투포인터

```
S = 27
arr = [1,  3,  5,  6,  9,  11,  12,  16,  17,  19,  22,  25,  28]
                              ↑    ↑
                           Right  Left
```

1. 리스트를 오름차순으로 정렬한다.
2. 포인터 `left`는 리스트의 시작점을, 포인터 `right`는 리스트의 끝점을 가리킨다.
3. `left`와 `right`가 만날 때 까지(즉, 모든 경우를 확인할 때 까지) 다음을 반복한다.
   - 3-1. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)과 같다면 조건을 만족하므로 출력하고, `left`를 1 증가, `right`를 1 감소시킨다.
   - 3-2. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 작다면 `left`를 1 증가시킨다.
   - 3-3. `arr[left]`와 `arr[right]`의 합이 타겟 값(S)보다 크다면 `right`를 1 감소시킨다.

Two pointer?

- 투 포인터를 활용하면 리스트를 한 번만 탐색
  - 리스트가 이미 정렬되어 있는 경우 O(n)
  - 정렬되어 있지 않더라도 O(n*log*n)

- 데이터양을 살펴보고 투포인터를 사용할 수 있다면 사용할 수 있는 기지가 필요
- 응용하여 Q2문제를 풀어보세요(백준 1802) 그럼 안녕!