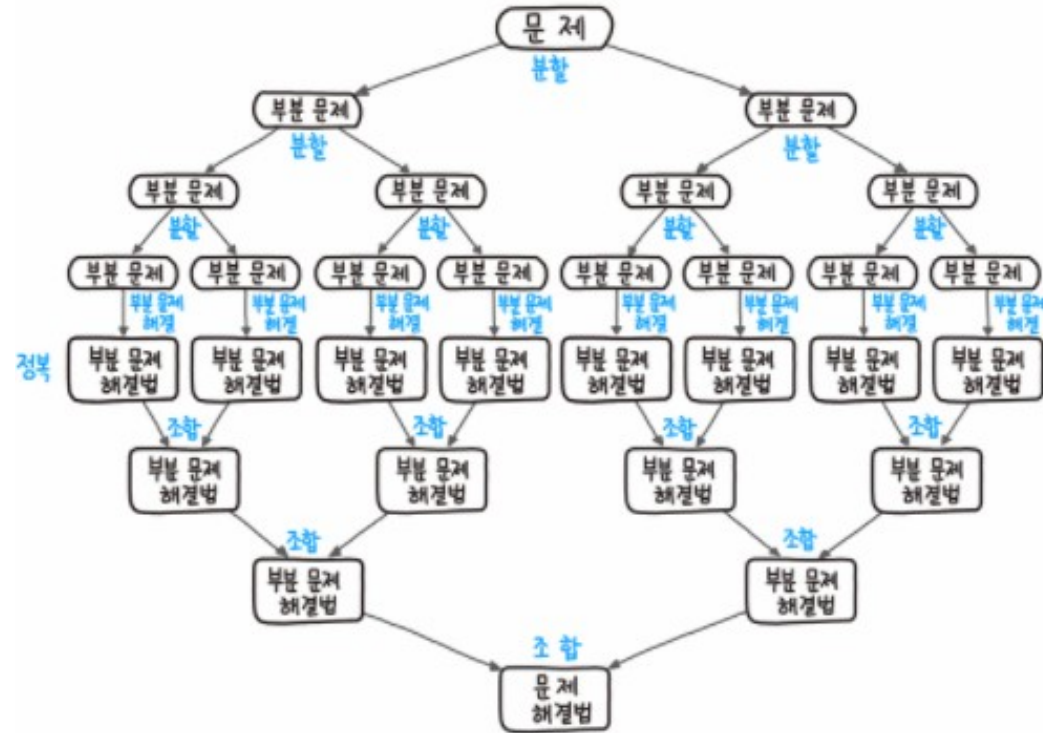


# 청년 AI 아카데미 23기 알고리즘 실습

## 재귀(Recursion)

# 재귀 (Recursion)



Bottom up!!

1. 문제를 굉장히 간단한 수준까지 하위 문제로 **분할**한다.
2. 하위 문제를 **해결**한다.
3. 하위 문제에 대한 결과를 원래 문제에 대한 결과로 **정복(합병)**한다.

c.f. Divide and Conquer  
기법이라고도 부른다.

# 실습 목표

- 재귀 문제를 해결하기 위한 방법을 배우고 다양한 재귀 문제들을 실습해보자.
  1. 피보나치 수열의  $n$ 번째 항 구하기
  2. 하노이 탑
  3. 이진 탐색

# 재귀 함수 구현 Tip 3가지

## 1. 재귀 함수가 무엇을 하는 함수인지 **명확하게 정의합니다.**

- 함수 이름이 의미가 있을수록 좋습니다.
- 함수가 지니는 매개변수들, 함수가 무엇을 반환하는지 등이 정의 안에 전부 녹아 있어야 합니다.  
Ex) 양의 정수  $n$ 이 주어졌을 때 1부터  $n$ 까지 합한 수를 계산하여라.

```
def Sum (a, b): # 자연수 a부터 b까지의 합을 반환하는 함수
```

Ex)  $\text{Sum}(4,9) = 39$

# 재귀 함수 구현 Tip 3가지

2. 재귀 함수가 함수 내에서 호출이 된다면, 그것은 **올바른 답을 준다고 가정**하고, 하위 문제 정답을 통해 원래 문제의 정답을 구합니다.

- 예를 들어 현재 구현 중인  $\text{Sum}(a,b)$  함수를 봅시다.
- $\text{Sum}(a,b-1)$ 은  $a$ 부터  $b-1$ 까지의 합을 반환하므로,  $\text{Sum}(a,b) = b + \text{Sum}(a,b-1)$ 입니다.
- 아직  $\text{Sum}(a,b-1)$ 를 구현하지 않았지만, **구현했다고 가정**합니다.

```
def Sum (a, b): # 자연수 a부터 b까지의 합을 반환하는 함수
```

```
    return Sum(a,b-1)+b
```



“ $\text{Sum}(a,b-1)$  은 자연수  $a$ 부터  $b-1$ 까지의 합을 반환해 줄 것이다” 라는 믿음

# 재귀 함수 구현 Tip 3가지

3. 재귀 함수엔 **명확한 종료** 조건이 반드시 있다.

- 재귀 함수는 자기 자신을 부르는 함수이므로, **종료 조건이 없다면 무한히 수행**됩니다.
- 따라서 함수 내에 **종료 조건**이 존재하고, 이는 **가장 단순한 경우**입니다.

Ex) Sum(a,b) 에서 a가 b보다 크다면? a와 b가 같다면?

4 + 5 + 6 + 7 + 8 + 9

```
def Sum(a,b): # 자연수 a부터 b까지의 합을 반환하는 함수
    if a==b:
        return a
    else:
        return Sum(a,b-1)+b
```

→ Sum(4,9) = Sum(4,8) + 9 = 39  
→ Sum(4,8) = Sum(4,7) + 8 = 30  
→ Sum(4,7) = Sum(4,6) + 7 = 22  
→ Sum(4,6) = Sum(4,5) + 6 = 15  
→ Sum(4,5) = Sum(4,4) + 5 = 9

→ Sum(4,4) = Sum(4,3) + 4  
자연수 a부터 a까지의 합은 a이다.

자연수 4부터 3까지의 합???

# 01. 피보나치 수열의 n번째 항 구하기

피보나치 수열의 n번째 항을 계산합니다.

$$F(1) = 1$$

$$F(2) = 1$$

$$F(3) = F(1) + F(2) = 1 + 1 = 2$$

...

$$F(N) = F(N-1) + F(N-2)$$

재귀 함수(Recursive Function)를 이용하여 구현하여 봅시다.



## 02. 하노이 탑

3개의 기둥 A, B, C가 있고, A에 있는 n개의 무게가 다른 원판을 C로 옮기고 싶습니다. 아래와 같은 규칙에 따라 원판을 이동시킬 수 있습니다.

- 원판은 한 번에 한 개씩만 제일 위에 있는 원판만 이동할 수 있습니다.
- 원판은 항상 무거운 것이 아래에 있어야 합니다. (시작 상태에서도 마찬가지)

A에서 C로 최소 횟수로 이동시킬 때, 이동하는 방식을 출력하는 프로그램을 작성합니다.

입력 예시

2
2
3

출력 예시

A->B
A->C
B->C
A->C
A->B
C->B
A->C
B->A
B->C
A->C



B

C





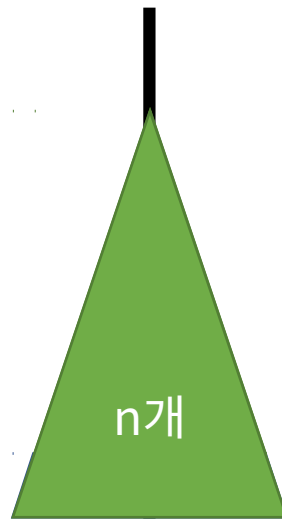
## 02. 하노이 탑

### 1. 재귀 함수의 정의

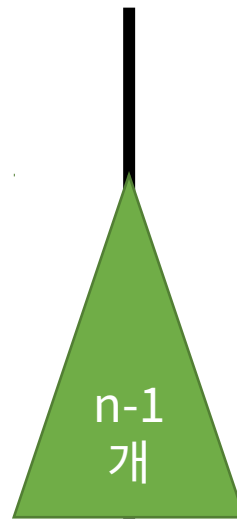
Hanoi(start,mid,end,n) #n개의 원판을 start에서 mid를 거쳐 end로 이동하는 방식을 출력하는 함수

### 2. 재귀 호출을 어떤 식으로 할 수 있을까요?

Hanoi(A,C,B,n-1)



A



B



C

```
Hanoi(A,C,B,n-1)
print(A->C)
Hanoi(B,A,C,n-1)
```

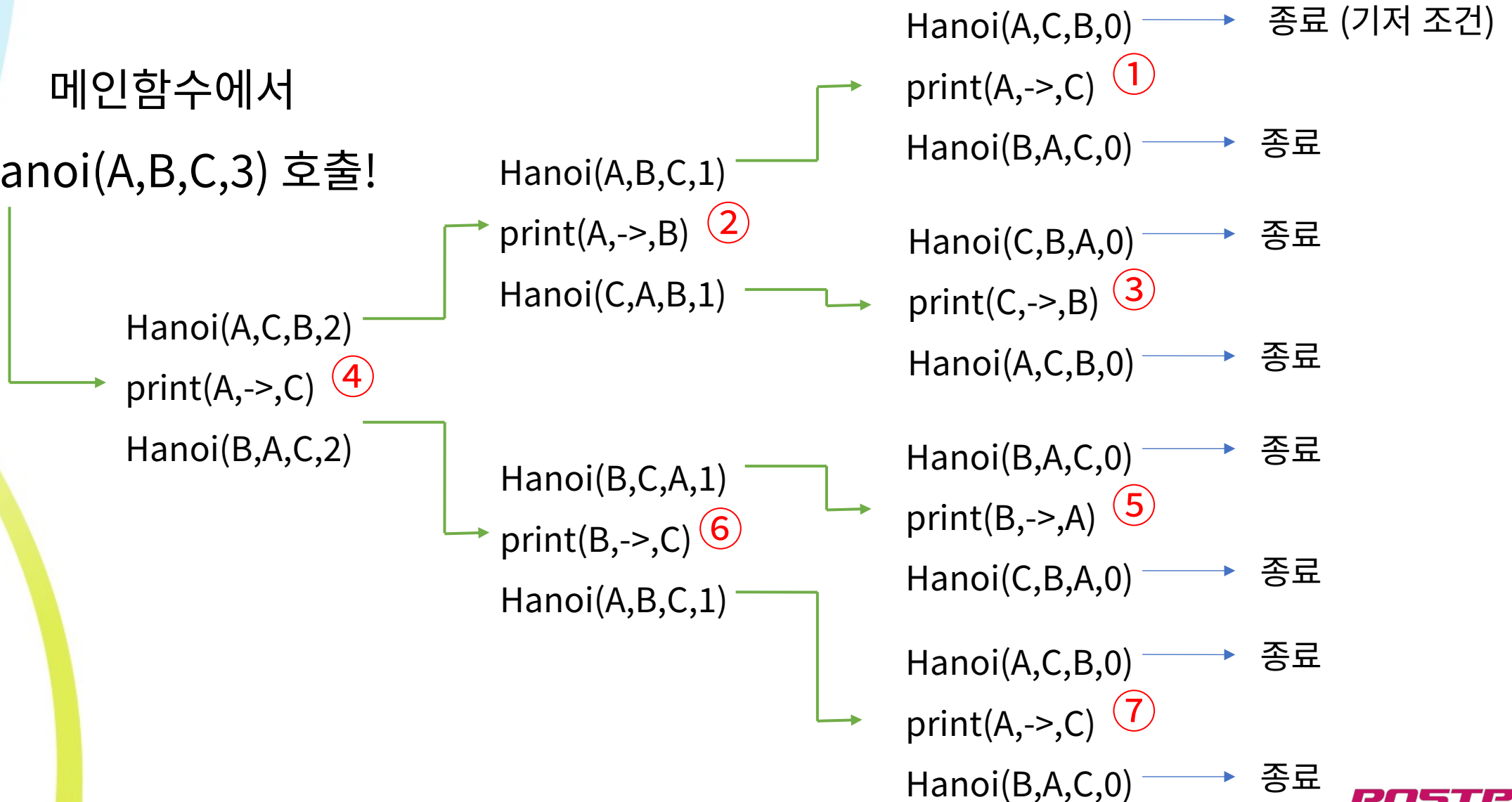
A,B,C의 위치에 주의!!

### 3. 종료 조건: 가장 기본적인 수행

n이 0이라면, 어떠한 실행도 하지 않고 함수를 종료한다.

## 02. 하노이 탑

메인함수에서  
Hanoi(A,B,C,3) 호출!



## 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.

Start=0 (리스트 인덱스)

End=8

5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----

리스트: [ 21 11 18 ]

출력: [ ]

주어진 리스트에 21이 들어 있는지 확인하는 방법?

1. 리스트의 앞부터 차례대로 탐색  $O(n)$
2. 이진탐색  $O(\log n)$  [단, 리스트가 정렬되어 있을 때!]  
-> 굉장히 유용 ex: 검색시스템

## 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.

Start=0 (리스트 인덱스)

End=8

5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----

리스트: [ 21 11 18 ]

출력: [ ]

## 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.

Start=0 (리스트 인덱스)

End=8

5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----



Median(중앙값) =  $(0+8)/2 = 4$

16 < 21

오른쪽을 탐색!

리스트: [ 21 11 18 ]

출력: [ ]

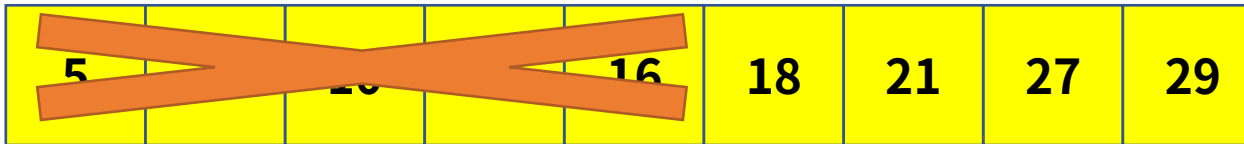
## 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.

기존 Start=0

새 Start=5

End=8



5	11	16	18	21	27	29
---	----	----	----	----	----	----

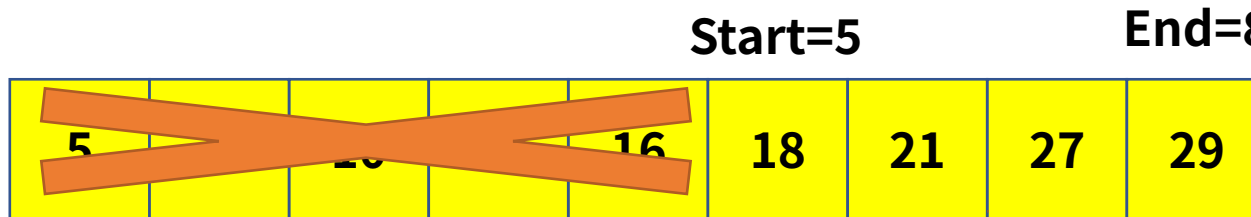
리스트: [ 21 11 18 ]

출력: [ ]

(새 Start)=5 index가 5보다 작은 원소들은 탐색하지 않아도 됨!

## 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.



리스트: [ 21 11 18 ]

출력: [ 6 ]

$$\text{Median(중앙값)} = (5+8)//2 = 6$$

왼쪽을 탐색!

List[6] = 21



## 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.

Start=0

End=8

5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----



Median=4

왼쪽을 탐색!     $16 > 11$

리스트: [ 21 11 18 ]

출력: [ 6 ]

## 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.

Start=0

End=3

5	9	10	12	16	21	29
---	---	----	----	----	----	----



Median=1

9 < 11    오른쪽을 탐색!

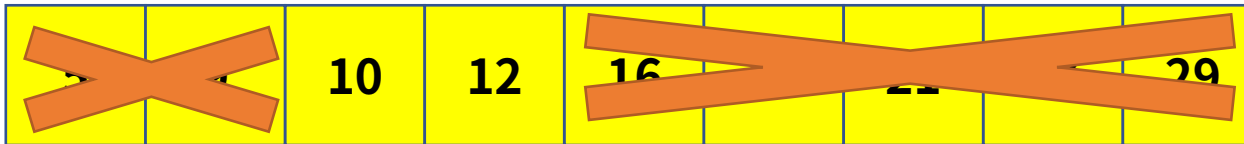
리스트: [ 21 11 18 ]

출력: [ 6 ]

## 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.

Start=2 End=3



Median=2

오른쪽 탐색!  $10 < 11$

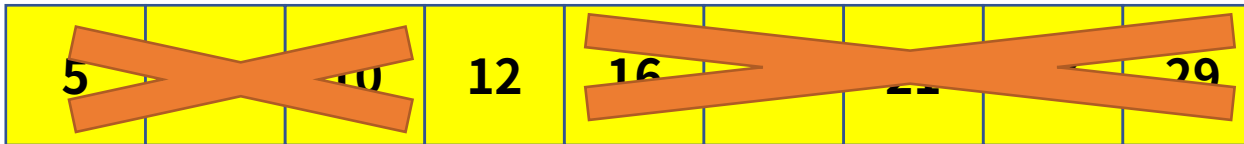
리스트: [ 21 11 18 ]

출력: [ 6 ]

### 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.

Start=End=3



Median=3

왼쪽 탐색!

$12 > 11$

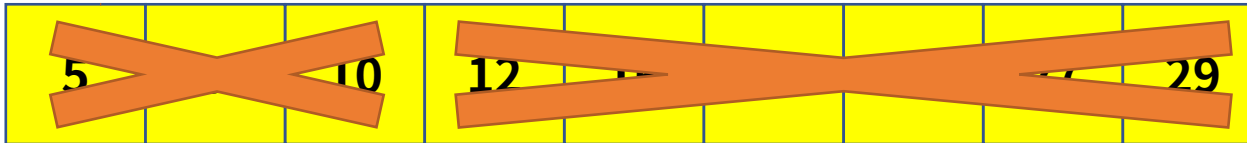
리스트: [ 21 11 18 ]

출력: [ 6 ]

### 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.

End=2 Start=3



존재하지 않음. -1 리턴

리스트: [ 21 11 18 ]

출력: [ 6 -1 ]

## 03. 이진 탐색

(1) 정렬된 리스트와, (2) 찾고자 하는 원소들을 담은 리스트가 주어집니다.

5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----

리스트: [ 21 11 18 ]

최종 결과

출력: [ 6 -1 5 ]

## 03. 이진 탐색

**Note**스트는 항상 오름차순으로 정렬돼 있음

5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----

(O)

5	10	9	12	16	18	21	27	29
---	----	---	----	----	----	----	----	----

(X) 이런 경우는 발생하지 않음

그렇기에, 중앙값을 기준으로 왼쪽 부분이나 오른쪽 부분 하나만 재귀적으로 조사하면 됨!



## 03. 이진 탐색

**Note**스트는 항상 오름차순으로 정렬돼 있음

5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----

(O)

5	10	9	12	16	18	21	27	29
---	----	---	----	----	----	----	----	----

(X) 이런 경우는 발생하지 않음

하지만, 두번째 줄로 주어지는 리스트는 그렇지 않을 수도 있음!

리스트: [ 9 21 18 ] (O)

리스트: [ 9 18 21 ] (O) 문제 없음



# ADD01. 이진 탐색 2

이전 문제인 이진 탐색과 비슷, 하지만 가장 가까운 원소를 찾아야 함!

Start=0

End=8

5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----



Mid(중간) =  $(0+8) // 2 = 4$

리스트: [ 21 22 38 ]

출력: [ ]

```
if List[mid]==target:
    output.append(target)
elif List[mid]<target:
    start=mid+1
elif List[mid]>target:
    end=mid
```

List[Start-1] 왼쪽의 원소들은 정답이 될 수  
**없지만**, List[Start-1]는 정답이 될 수 있음

List[End] 오른쪽의 원소들은 정답이 될 수  
**없지만**, List[End]는 정답이 될 수 있음

# ADD01. 이진 탐색 2

이전 문제인 이진 탐색과 비슷, 하지만 가장 가까운 원소를 찾아야 함!

Start=0

End=8

5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----



Mid=4

16 < 21

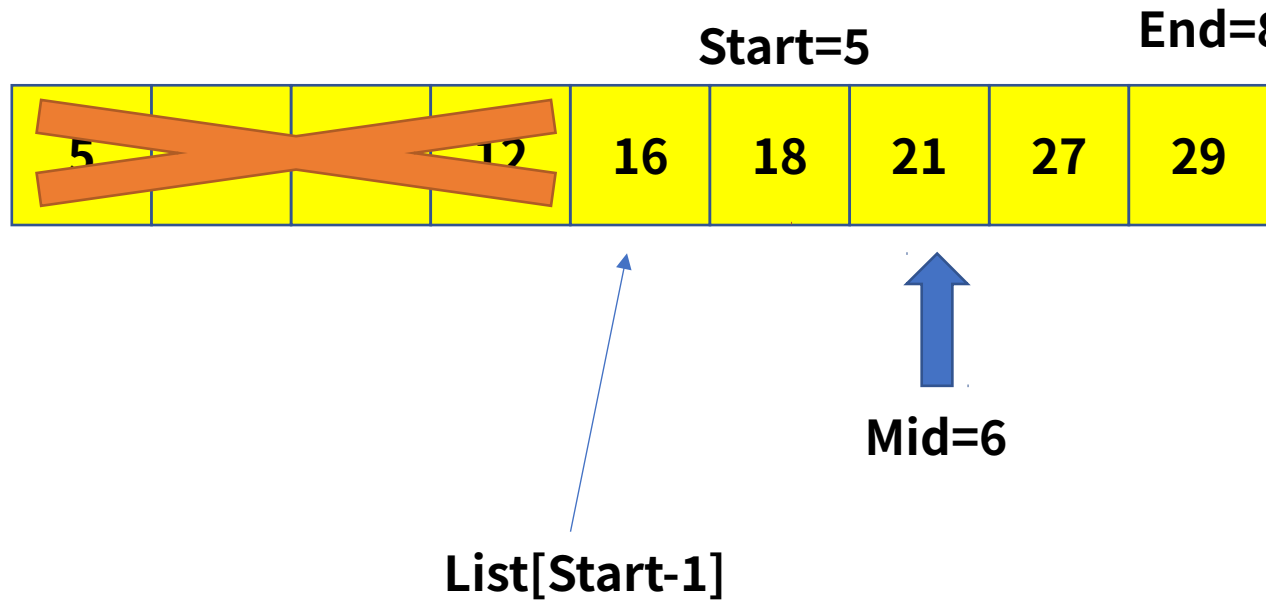
오른쪽을 탐색!

리스트: [ 21 22 38 ]

출력: [ ]

## ADD01. 이진 탐색 2

이전 문제인 이진 탐색과 비슷, 하지만 가장 가까운 원소를 찾아야 함!

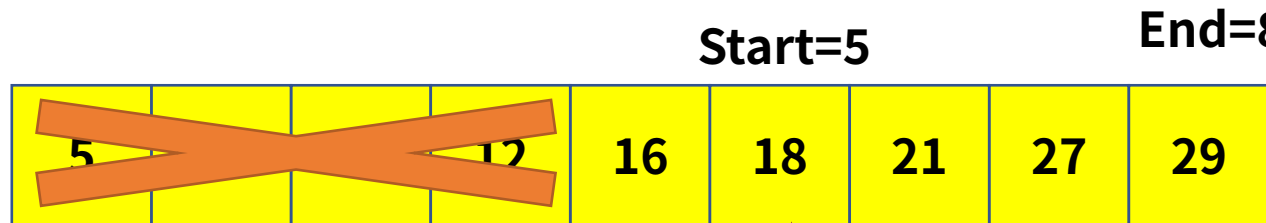


리스트: [ 21 22 38 ]

출력: [ ]

# ADD01. 이진 탐색 2

이전 문제인 이진 탐색과 비슷, 하지만 가장 가까운 원소를 찾아야 함!



Mid=6

탐색 완료!

List[mid] = 21

값을 그대로 리턴!

리스트: [ 21 22 38 ]

출력: [21]

# ADD01. 이진 탐색 2

이전 문제인 이진 탐색과 비슷, 하지만 **가장 가까운 원소**를 찾아야 함!

Start=0

End=8

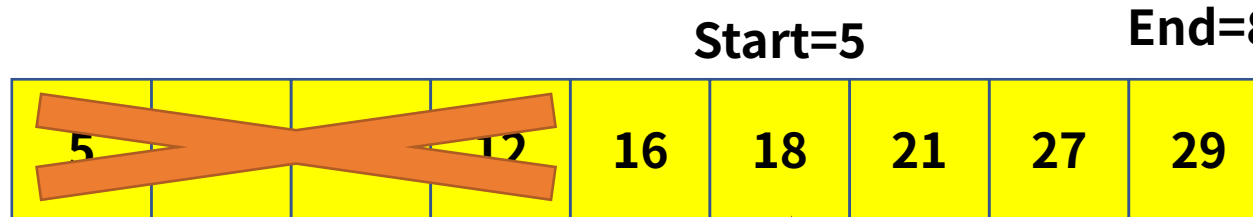
5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----

리스트: [ 21 **22** 38 ]

출력: [ 21 ]

# ADD01. 이진 탐색 2

이전 문제인 이진 탐색과 비슷, 하지만 가장 가까운 원소를 찾아야 함!



Mid=6

21 < 22    **오른쪽을 탐색!**

리스트: [ 21 22 38 ]

출력: [ 21 ]

직전의 21과 비슷하게, 22를 찾는 과정이 진행됨. 여기서...



## ADD01. 이진 탐색 2

이전 문제인 이진 탐색과 비슷, 하지만 가장 가까운 원소를 찾아야 함!

Start=7 End=8



List[Start-1]

Mid=7

27 > 22    왼쪽을 탐색!

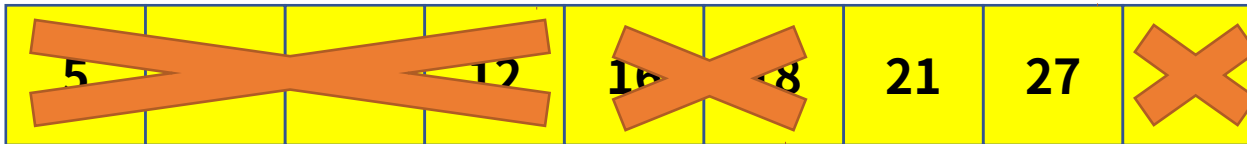
리스트: [ 21 22 38 ]

출력: [ 21 ]

## ADD01. 이진 탐색 2

이전 문제인 이진 탐색과 비슷, 하지만 가장 가까운 원소를 찾아야 함!

Start=End=7



Mid=7

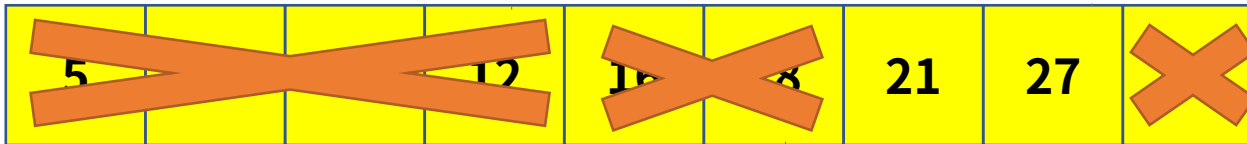
리스트: [ 21 22 38 ]

출력: [ 21 ]

## ADD01. 이진 탐색 2

이전 문제인 이진 탐색과 비슷, 하지만 가장 가까운 원소를 찾아야 함!

Start=End=7



Mid=7

리스트: [ 21 22 38 ]

출력: [ 21 21 ]

남은 선택지는 2개밖에 없음.(21과 27)

22-21 < 27-22이므로,  
정답은 21

# ADD01. 이진 탐색 2

이전 문제인 이진 탐색과 비슷, 하지만 **가장 가까운 원소**를 찾아야 함!

5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----

리스트: [ 21 22 **38** ]

출력: [ 21 21 **29** ]

**38**은 가장 큰 원소인 29보다도 크므로,  
정답은 29

# ADD01. 이진 탐색 2 [ 힌트 3: 라이브러리 사용 ]

```
import bisect
...
i = bisect_left(l,x)
```

위의 구문을 통해 정렬된 리스트 l에서 x라는 값이 들어갈 위치를 찾을 수 있습니다.  
이때 bisect\_left 함수는 x와 같은 값들이 있을 때 그 중 가장 작은 인덱스를 반환하며  
bisect\_right 함수는 가장 큰 인덱스를 반환합니다.

bisect\_left:  $l[i-1] < x \leq l[i]$  if  $i \neq 0$  ,    bisect\_right:  $l[i] \leq x < l[i+1]$  if  $i \neq \text{len}(l)-1$

# ADD01. 이진 탐색 2 [ 힌트 3: 라이브러리 사용 ]

```
import bisect
...
i = bisect_left(l,x)
```

위의 구문을 통해 정렬된 리스트 l에서 x라는 값이 들어갈 위치를 찾을 수 있습니다.  
이때 bisect\_left 함수는 x와 같은 값들이 있을 때 그 중 가장 작은 인덱스를 반환하며  
bisect\_right 함수는 가장 큰 인덱스를 반환합니다.

bisect\_left:  $l[i-1] < x \leq l[i]$  if  $i \neq 0$  ,    bisect\_right:  $l[i] \leq x < l[i+1]$  if  $i \neq \text{len}(l)-1$



1	3	5	7	9	11	13	15	17	19
---	---	---	---	---	----	----	----	----	----

$x=8$  or  $9$

`bisect_left(l,x)==4`

# ADD02. 합병

합병 정렬(Merge Sort) -> Divide and Conquer를 이용하여 정렬하는 알고리즘

초기상태 

21	10	12	20	25	13	15	22
----	----	----	----	----	----	----	----

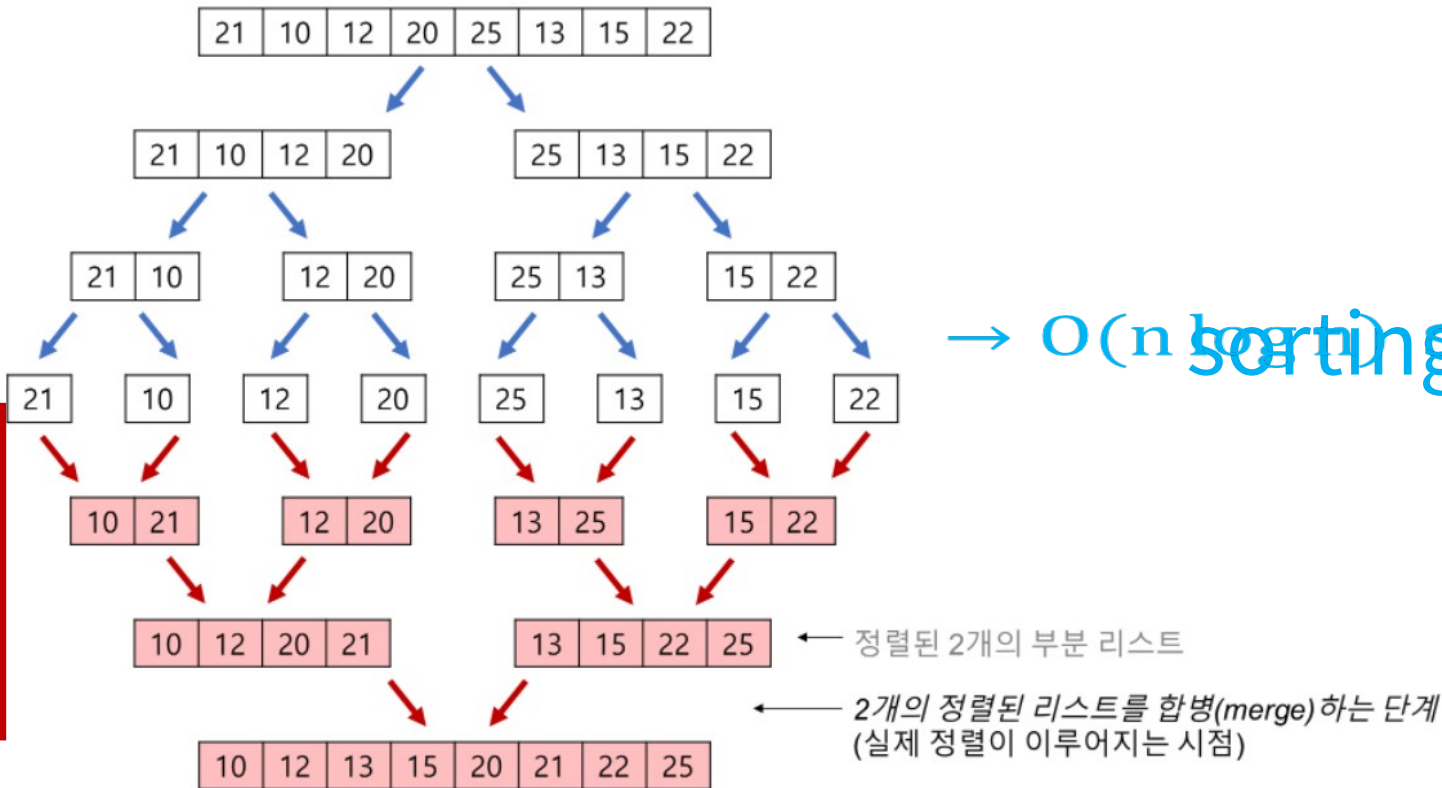
분할

Divide  
Divide  
Divide

→  $O(n \log n)$  sorting algorithm!!

정복

Conquer  
Combine  
Conquer  
Combine  
Conquer  
Combine



오름차순  
완성상태

10	12	13	15	20	21	22	25
----	----	----	----	----	----	----	----



# ADD02. 합병

합병 정렬(Merge Sort) -> Divide and Conquer를 이용하여 정렬하는 알고리즘

초기상태 21 10 12 20 25 13 15 22

분할

Divide

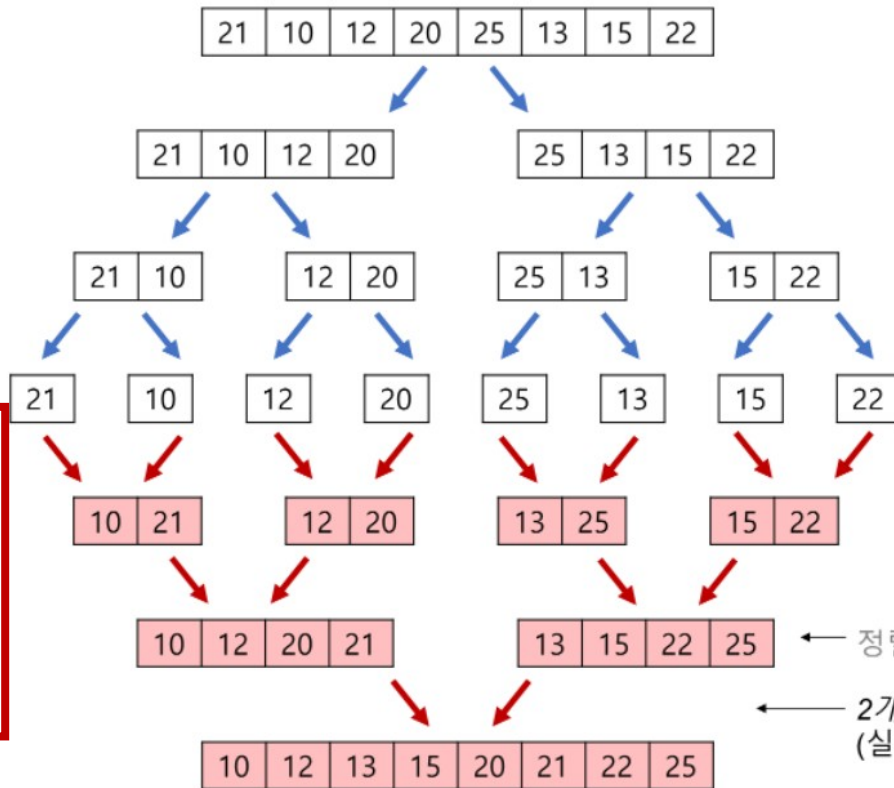
Divide

Divide

Conquer  
Combine

Conquer  
Combine

Conquer  
Combine



← 정렬된 2개의 부분 리스트

← 2개의 정렬된 리스트를 합병(merge) 하는 단계  
(실제 정렬이 이루어지는 시점)

오름차순  
완성상태

10 12 13 15 20 21 22 25

#Pseudocode

```
def merge_sort(input_list):    #input_list가 정렬된
    half=len(input_list)//2    #결과를 반환하는 함수
    if len(input_list)<=1:
        return input_list
    elif len(input_list)==2:
        sol=input_list
        if input_list[0]>input_list[1]:
            sol=[input_list[1],input_list[0]]
        return sol
    else:
        left=merge_sort(input_list[:half])
        right=merge_sort(input_list[half:])
        return merge(left,right)
```

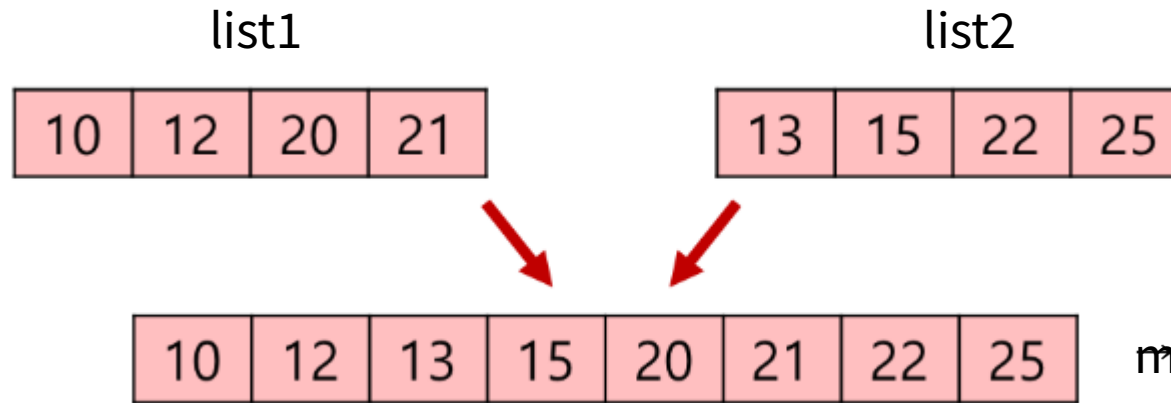
종료조건

분할

정복(합병)

# ADD02. 합병

합병 정렬(Merge Sort) -> Divide and Conquer를 이용하여 정렬하는 알고리즘



merge(list1, list2)의 return 값

## ADD02. 합병

정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.

List 1

1	3	7	11	15
---	---	---	----	----

List 2

2	5	6	13	14
---	---	---	----	----

## ADD02. 합병

정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.

List 1

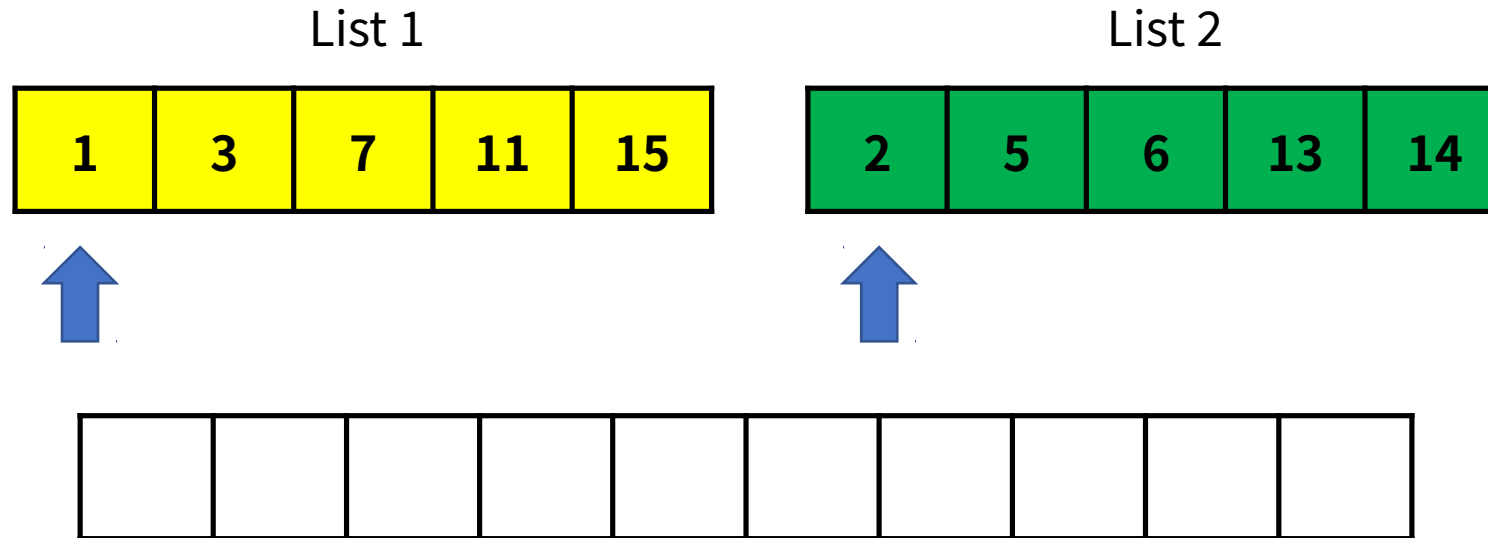
1	3	7	11	15
---	---	---	----	----

List 2

2	5	6	13	14
---	---	---	----	----

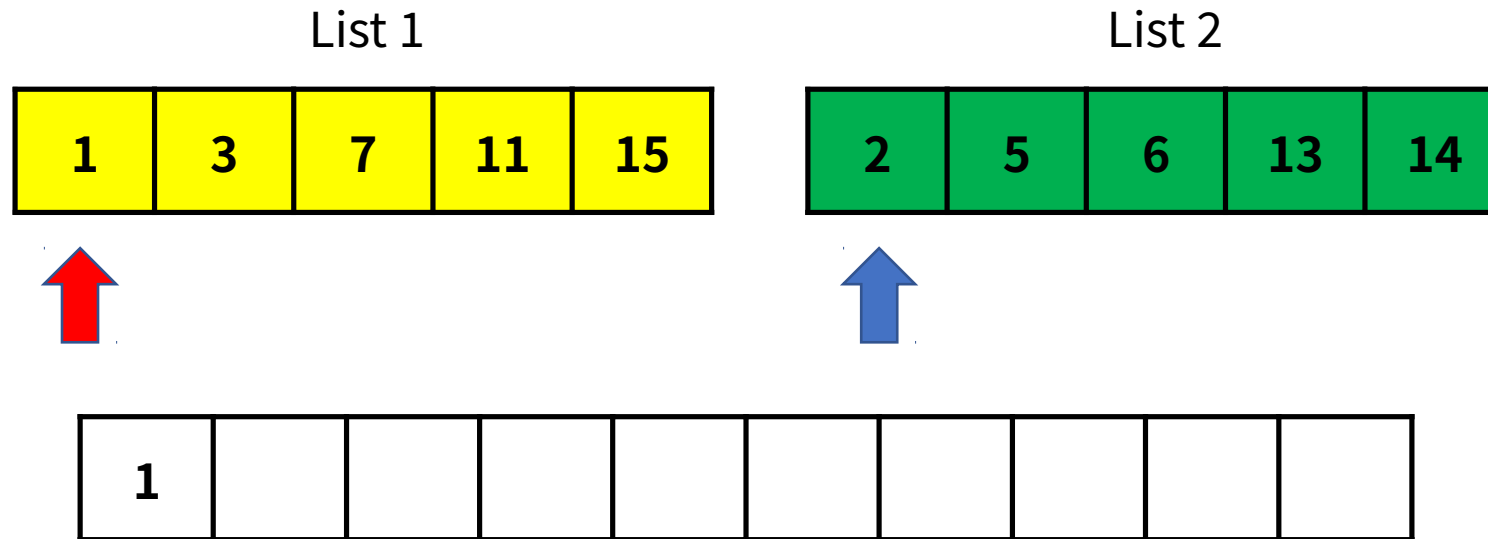
## ADD02. 합병

정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.



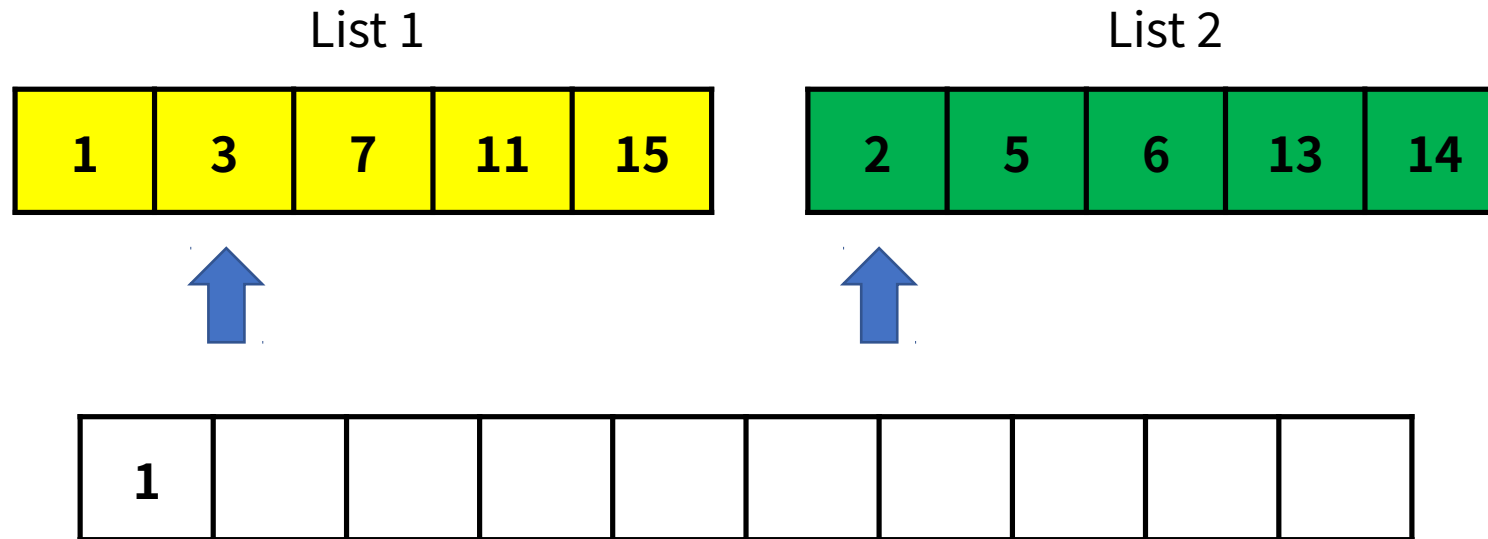
## ADD02. 합병

정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.



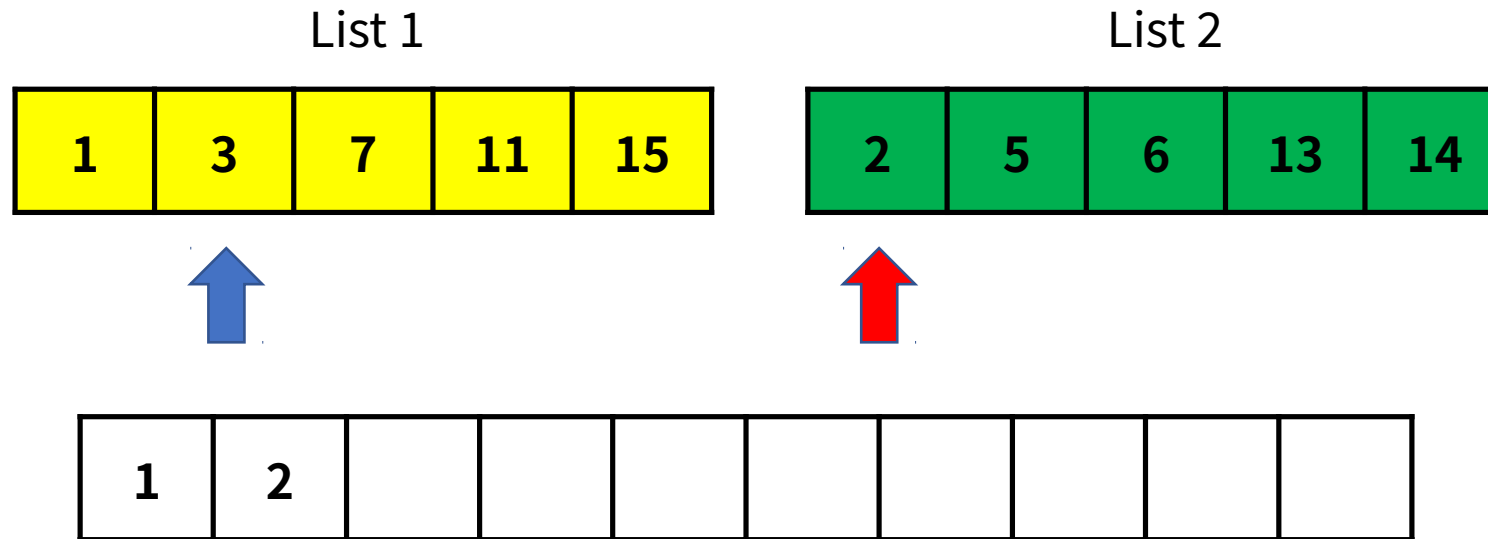
## ADD02. 합병

정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.



## ADD02. 합병

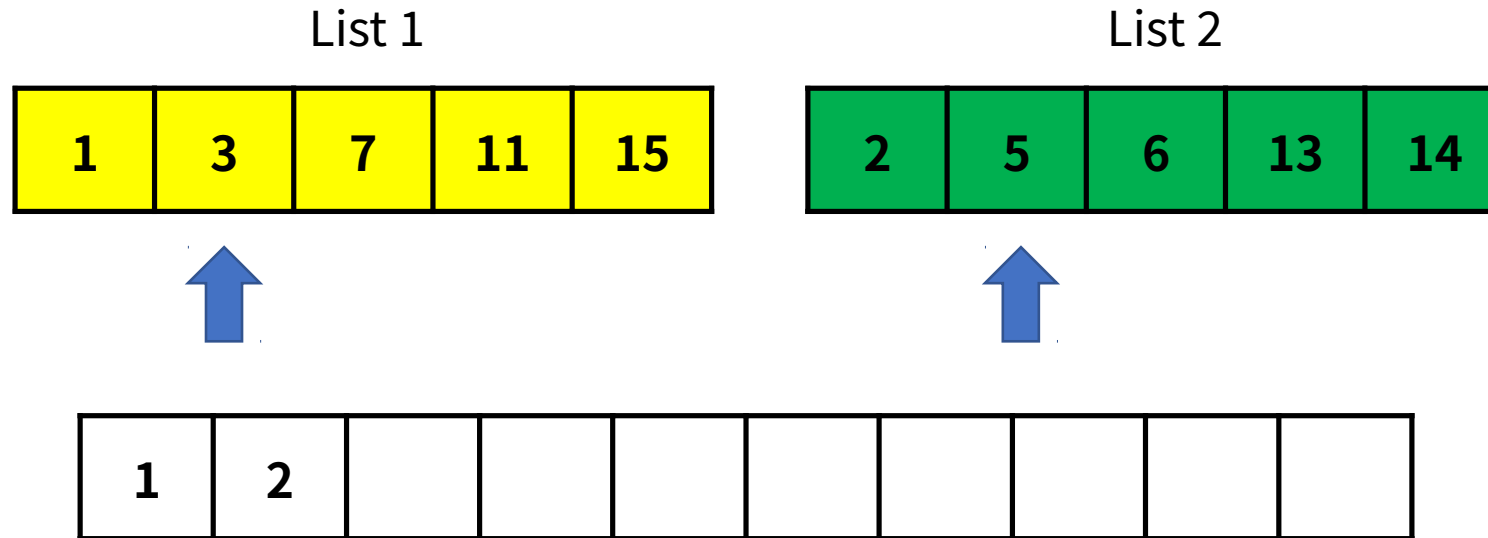
정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.





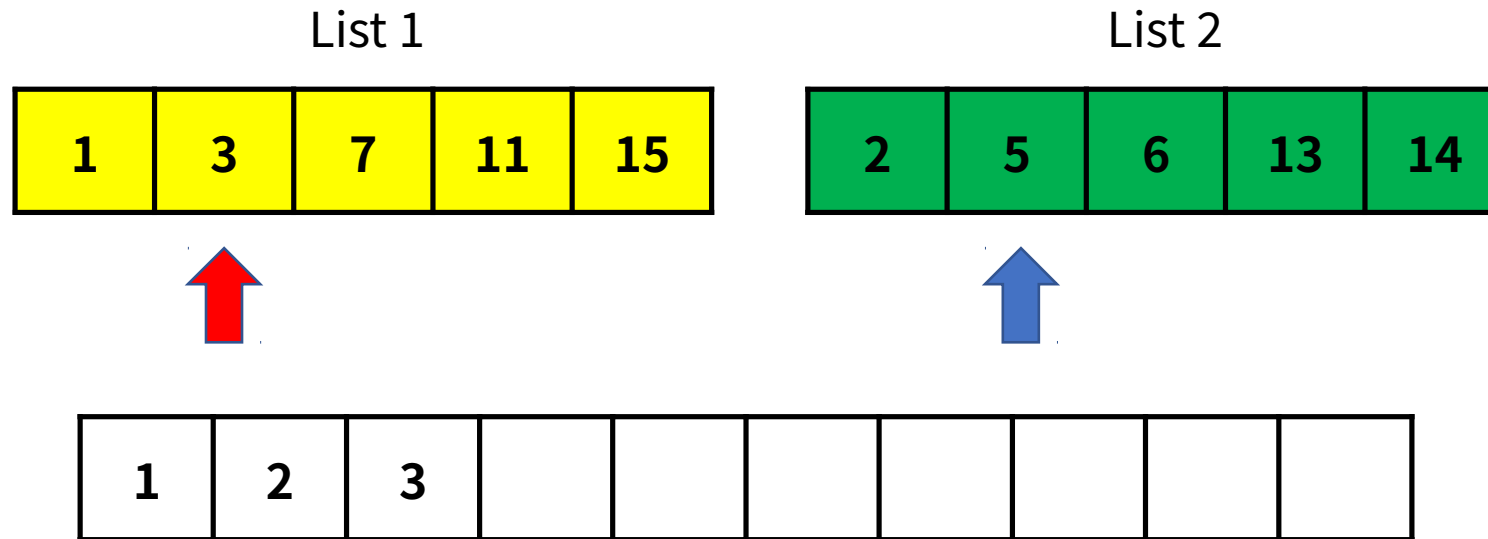
## ADD02. 합병

정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.



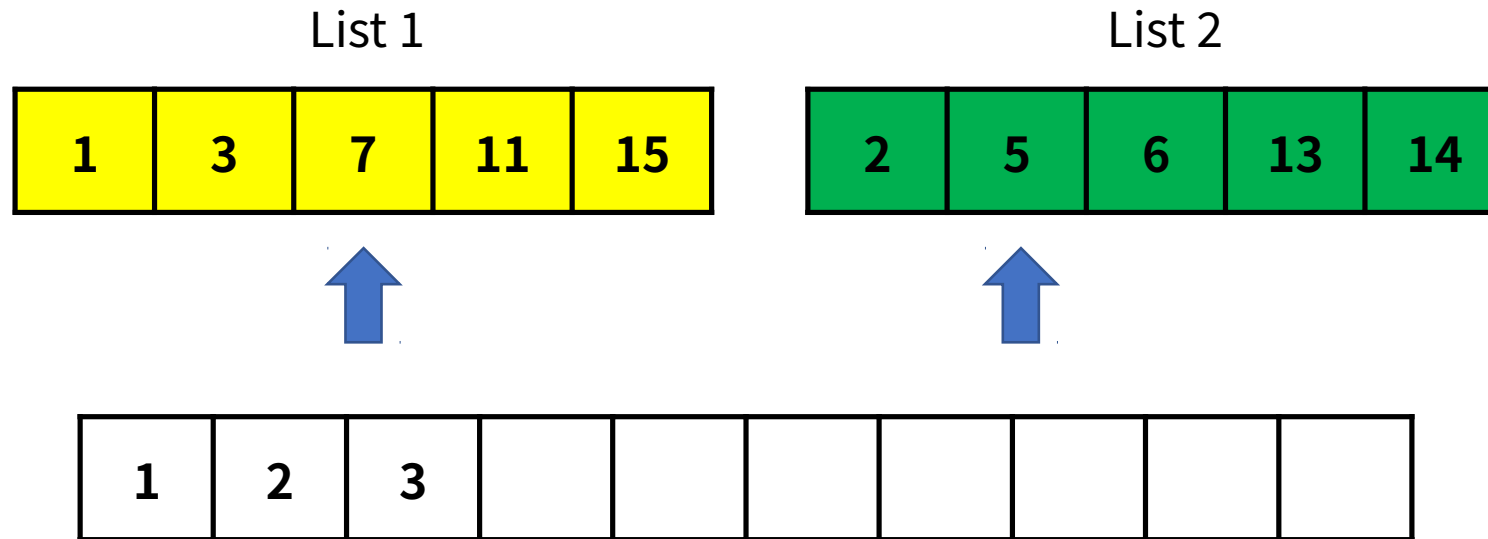
## ADD02. 합병

정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.



## ADD02. 합병

정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.



## ADD02. 합병

정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.

List 1

1	3	7	11	15
---	---	---	----	----

List 2

2	5	6	13	14
---	---	---	----	----

1	2	3	5	6	7	11	13	14	15
---	---	---	---	---	---	----	----	----	----

## ADD02. 합병

정렬된 두 개의 리스트가 주어졌을 때, 두 리스트를 합병하여 정렬한 결과의 각 원소가 두 리스트 중 어떤 리스트에서 가져온 것인지 계산하는 프로그램을 작성하여라.

List 1

1	3	7	11	15
---	---	---	----	----

List 2

2	5	6	13	14
---	---	---	----	----

1	2	3	5	6	7	11	13	14	15
---	---	---	---	---	---	----	----	----	----

출력: 1 2 1 2 2 1 1 2 2 1