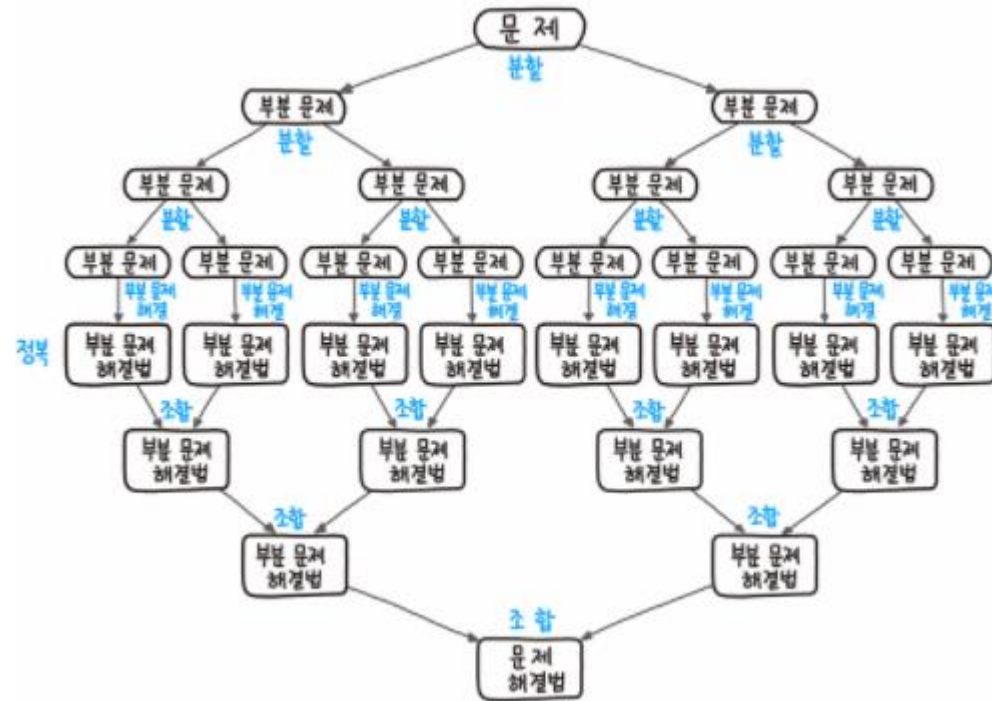


대경 HuStar아카데미 알고리즘 실습

분할 정복

분할 정복의 철학



Bottom up!!

1. 문제를 굉장히 간단한 수준까지 하위 문제로 나눈다.
2. 하위 문제를 해결하여 정복한다.
3. 하위 문제에 대한 결과를 원래 문제에 대한 결과로 조합한다.

분할 정복의 예시

자연수 a, b 가 주어지면, $a \sim b$ 까지의 합을 더하는 프로그램

1. 해결할 문제를 함수로 정의

$\text{Sum}(a, b)$: $a \sim b$ 까지의 합

2. 분할(재귀함수)

$a \sim b$ 를 절반으로 나눔

→ 재귀함수로 부분문제를 해결

3. 정복(분할에 의한 결과들 합치기)

S_a 와 S_b 로 $\text{Sum}(a, b)$ 를 구함

→ $S_a + S_b$

4. 종료 조건(초기값)

a 와 b 가 같을 때 $\text{Sum}(a, b) = a$

```
def Sum(a, b):
```

```
    if a == b:
        return a
```

```
    mid = (a + b) // 2
    Sa = Sum(a, mid)
    Sb = Sum(mid + 1, b)
```

```
    return Sa + Sb
```

01. $n^k \bmod m$ 효율적으로 계산하기

n^k 를 12345678로 나눈 나머지를 계산합니다.

Ex) $2^5 \bmod 12345678 = 32$

$10^7 \bmod 12345678 = 10000000$

$10^8 \bmod 12345678 = 1234576$

$1^{89195} \bmod 12345678 = 1$

$8949156^0 \bmod 12345678 = 1$

$0^{123456} \bmod 12345678 = 0$

Python에서는 int 범위 이상의 큰 수를 자동으로 다루지만, **자리수가 많아지면 오래 걸리기 때문에 위의 식을 활용해야 합니다!** (cf. C++: Overflow)

Hint1: $ab \bmod x = ((a \bmod x) * (b \bmod x)) \bmod x$

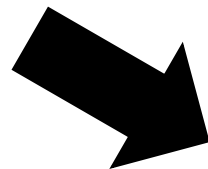
9*9을 5로 나눈 나머지 = 1 = (9를 5으로 나눈 나머지)*(9을 5으로 나눈 나머지) 를 5으로 나눈 나머지

Hint2: $X^a X^b = X^{(a+b)}$

Hint3: $X^{2b} = X^b * X^b$ & $X^{2b+1} = X^b * X^b * X$

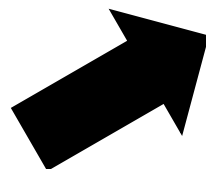
01. $n^k \bmod m$ 효율적으로 계산하기

$$n^k = \begin{cases} 1 & k = 0 \\ n & k = 1 \\ (n^{\lfloor \frac{k}{2} \rfloor})^2 \cdot n & k \text{ is odd} \\ (n^{\lfloor \frac{k}{2} \rfloor})^2 & k \text{ is even} \end{cases}$$



Pow(n, k, m)

$$n^k \bmod m = \begin{cases} 1 & k = 0 \\ n \bmod m & k = 1 \\ (n^{\lfloor \frac{k}{2} \rfloor})^2 \cdot n \bmod m & k \text{ is odd} \\ (n^{\lfloor \frac{k}{2} \rfloor})^2 \bmod m & k \text{ is even} \end{cases}$$



$$= \begin{cases} 1 & \text{if } k == 0 \\ n \% m & \text{if } k == 1 \\ (X * X * n) \% m & \text{if } k \% 2 == 1 \\ (X * X) \% m & \text{if } k \% 2 == 0 \end{cases}$$

$X = \text{Pow}(n, k // 2, m)$

01. $n^k \bmod m$ 효율적으로 계산하기

```
def Power(n, k, m):    #Power(n, k, m): n^k를 m으로 나눈 나머지를 반환하는 함수
    if k == 0:
        return 1
    if k == 1:
        return n

    half = Power(n, k//2, m)

    if k % 2 == 0:
        return (half*half)%m
    else:
        return (half*half*n)%m

t = int(input())
for _ in range(t):
    n, k, m = map(int, input().split())
    print(Power(n, k, m))
```

01. $n^k \bmod m$ 효율적으로 계산하기

```
1 t=int(input())
2 for _ in range(t):
3     n,k,mod = map(int,input().split())
4     ans = pow(n,k,mod)
5     print(ans)
```

02. 이진 탐색 1

(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

02. 이진 탐색 1

(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: []

02. 이진 탐색 1

(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: []

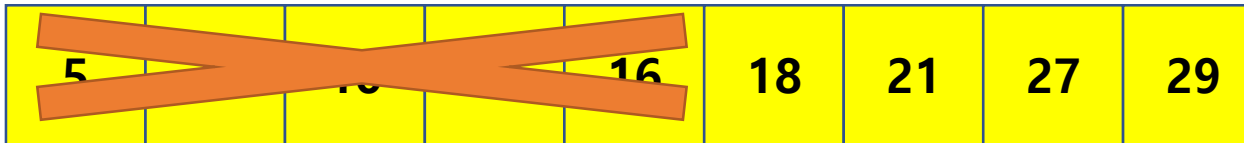
02. 이진 탐색 1

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

기존 Start=0

새 Start=5

End=8



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

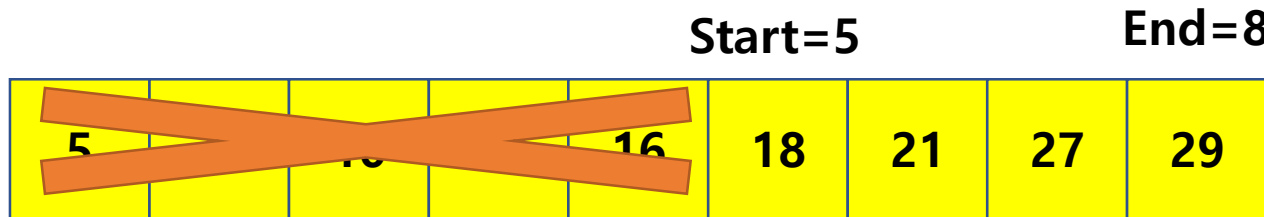
리스트: [21 11 18]

출력: []

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

02. 이진 탐색 1

(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

02. 이진 탐색 1

(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]

02. 이진 탐색 1

(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!

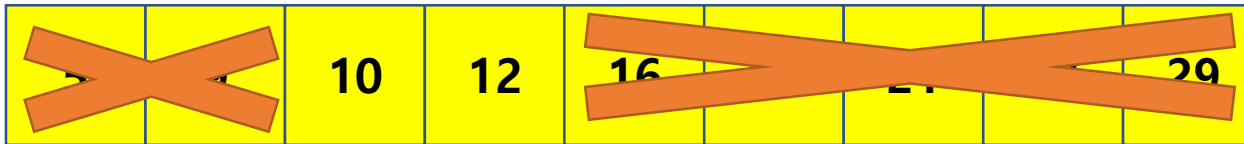
리스트: [21 11 18]

출력: [6]

02. 이진 탐색 1

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

Start=2 End=3



Median=2

오른쪽 탐색! $10 < 11$

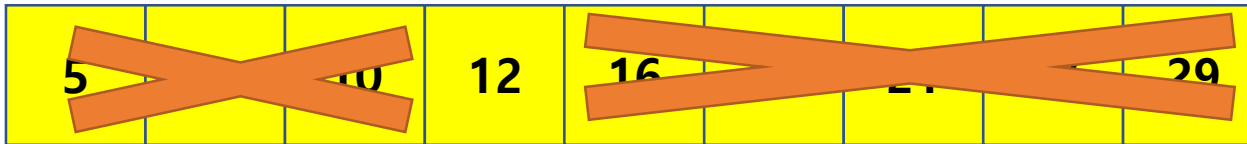
리스트: [21 11 18]

출력: [6]

02. 이진 탐색 1

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

Start=End=3



Median=3

왼쪽 탐색! $12 > 11$

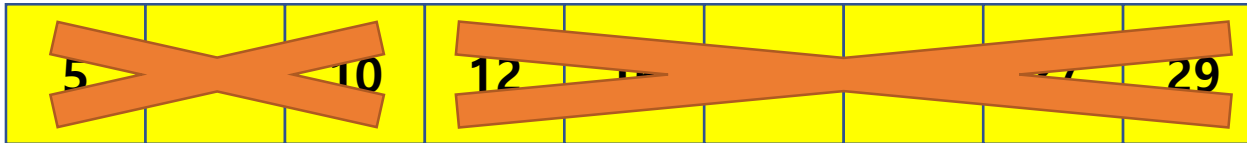
리스트: [21 11 18]

출력: [6]

02. 이진 탐색 1

(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]

02. 이진 탐색 1

(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]

02. 이진 탐색 1

Note: 각 테스트 케이스마다, **첫번째 줄**로 주어지는 리스트는 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!

02. 이진 탐색 1

Note: 각 테스트 케이스마다, **첫번째 줄**로 주어지는 리스트는 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

02. 이진 탐색 1 [재귀 함수]

```
def binary(data, left, right, q): #data의 left:right 범위 내에서 q의 위치를 반환하는 함수
    if left > right: # 없는 경우의 중간
        return -1
    mid = (left + right) // 2 # 중간 위치의 값 설정
    if data[mid] == q: # 현재 중앙에 있는 값이 맞는 경우의 조건
        return mid
    # 문제분할: q와 data[mid]의 관계에 따라 부분문제를 호출
    if data[mid] > q:
        subproblem = binary(data, left, mid-1, q)
    elif data[mid] < q:
        subproblem = binary(data, mid+1, right, q)
    # 분할된 문제들을 합쳐서 따로 행하는 것이 없음
    return subproblem
```

```
t = int(input())
for _ in range(t):
    data = list(map(int, input().split()))
    query = list(map(int, input().split()))
    answer = []

    for q in query:
        answer.append(binary(data, 0, len(data)-1, q))

    print(*answer)
```

A=[1,2,3]
print(A) → [1,2,3] 출력
print(*A) → 1 2 3 출력

02. 이진 탐색 1 [반복문]

```
t = int(input())
for _ in range(t):
    data = list(map(int, input().split()))
    query = list(map(int, input().split()))
    answer = []

    for q in query:
        left = 0
        right = len(data)-1
        while left <= right:
            mid = (left + right) // 2
            if data[mid] == q:
                break
            elif data[mid] > q:
                right = mid - 1
            elif data[mid] < q:
                left = mid + 1
        if left > right:
            answer.append(-1)
        else:
            answer.append(mid)

    print(*answer)
```

02. 이진 탐색 1 [라이브러리]

```
import bisect

def find(l,x):
    i = bisect.bisect_left(l,x)
    if i != len(l) and l[i] == x:
        return i
    return -1

t = int(input())
for _ in range(t):
    data = list(map(int,input().split()))
    query = list(map(int,input().split()))
    answer = [find(data,x) for x in query]
    print(*ans)
```

03. 최대 합 부분 연속 수열

리스트가 주어지면, 적당한 구간을 잡습니다.

그 구간들 전부 중에서, 구간 내 원소들의 합이 가장 큰 것을 찾는 프로그램을 작성합니다.

1	-2	3	-4	5	-3	8	-9	22
---	----	---	----	---	----	---	----	----

$$\text{합: } 5 + (-3) + 8 + (-9) = 23$$



03. 최대 합 부분 연속 수열

1. 리스트의 크기가 1인 경우에는 원소가 하나 뿐이므로 그 원소의 값이 최대 합
2. 리스트의 크기가 2 이상인 경우, 리스트를 왼쪽 반과 오른쪽 반 두 개로 나누어 각각의 최대 합을 구한다
3. 리스트의 왼쪽과 오른쪽을 걸친 부분 연속 수열의 최대 합을 구한다.

1, 2는 재귀 함수를 통해 구현

3번은 리스트를 가운데에서 왼쪽/가운데에서 오른쪽으로 하나씩 탐색하며 구한다.

1	-2	3	-4	5	-3	8	1	2	-5	5	-9	22
1	-2	3	-4	5	-3	8	1	2	-5	5	-9	22
1	-2	3	-4	5	-3	8	1	2	-5	5	-9	22
1	-2	3	-4	5	-3	8	1	2	-5	5	-9	22



03. 최대 합 부분 연속 수열

1	-2	3	-4	5	-3
---	----	---	----	---	----

리스트의 왼쪽과 오른쪽에 걸쳐있는 경우 가운데는 반드시 포함하게 됨
lsum과 lmax 값을 유지하며 리스트를 탐색
lsum: 가운데에서 현재 인덱스까지의 합
lmax: 현재까지 확인한 lsum 값 중 최대값

03. 최대 합 부분 연속 수열

1	-2	3	-4	5	-3
---	----	---	----	---	----



lsum: -3
lmax: -3

03. 최대 합 부분 연속 수열

1	-2	3	-4	5	-3
---	----	---	----	---	----



lsum: $-3+5=2$

lmax: $\max(-3, \text{lsum})=2$

03. 최대 합 부분 연속 수열

1	-2	3	-4	5	-3
---	----	---	----	---	----



lsum: $2 + (-4) = -2$

lmax: $\max(2, \text{lsum}) = 2$

03. 최대 합 부분 연속 수열

1	-2	3	-4	5	-3
---	----	---	----	---	----



lsum: $-2+3=1$

lmax: $\max(2, \text{lsum})=2$

03. 최대 합 부분 연속 수열

1	-2	3	-4	5	-3
---	----	---	----	---	----



lsum: $1 + (-2) = -1$

lmax: $\max(2, \text{lsum}) = 2$

03. 최대 합 부분 연속 수열

1	-2	3	-4	5	-3
---	----	---	----	---	----



lsum: $-1+1=0$

lmax: $\max(2, \text{lsum})=2$

03. 최대 합 부분 연속 수열

```
t=int(input())  
  
for _ in range(t):  
    l = list(map(int,input().split()))  
    print(mls(l))
```

03. 최대 합 부분 연속 수열

```
1 def mls(l):
2     n = len(l)
3     #만약 원소가 하나라면, 정답은 그 원소.
4     if n==1:
5         return l[0]
6     #전체 리스트를 절반으로 나누어 각각에 대해서 부분 문제를 해결
7     left = mls(l[:n//2])
8     right = mls(l[n//2:])
9     #left는 왼쪽 절반 리스트에서의 최대합이고,
10    #right는 오른쪽 절반 리스트에서의 최대합.
11
12    #왼쪽 리스트와 오른쪽 리스트에 동시에 걸쳐있는 정답을 고려하여야함.
13    #양쪽에 걸쳐있는 것은 반드시 리스트의 중앙을 지나게 됨.
14    lindex=n//2-1 #왼쪽 리스트에 걸쳐 있는 부분은 반드시 포함해야 하는 원소
15    lmax=l[lindex] #현재까지 확인한 합중 최대
16    lsum=l[lindex] #중간값 부터 lindex 까지의 합
17    for i in range(lindex-1,-1,-1): # 1씩 감소하는 루프, lindex-1, lindex-2, ..., 0
18        lsum+=l[i] #합 변경
19        if lsum>lmax: #합의 최대값 갱신
20            lmax=lsum
21    #위와 같은 내용을 오른쪽 리스트에 대해서 확인
22    rindex=n//2
23    rmax=l[rindex]
24    rsum=l[rindex]
25    for i in range(rindex+1,n):
26        rsum+=l[i]
27        if rsum>rmax:
28            rmax=rsum
29    #세가지 후보 중 가장 큰 값이 정답.
30    return max(lmax+rmax, left, right)
```

한글
판

정보



← 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
----	----	----	----	----	----	----	----

Project 01. 합병

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

초기상태

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

분할

Divide
Divide
Divide

#Pseudocode

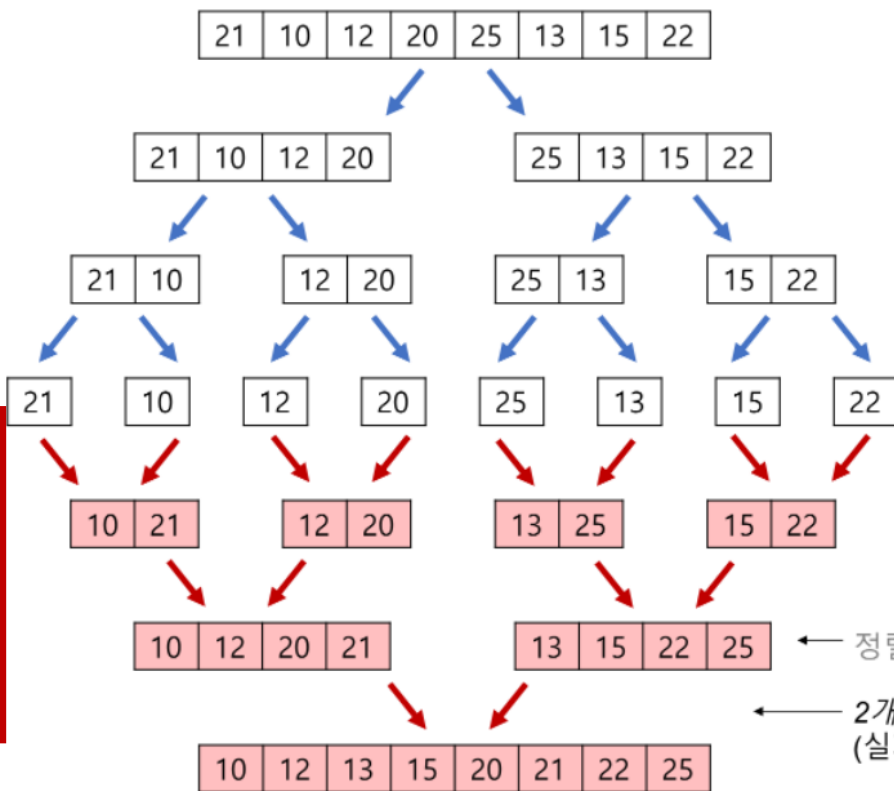
```
def merge_sort(input_list):    #input_list가 정렬된
    half=len(input_list)//2    #결과를 반환하는 함수
    if len(input_list)<=1:
        return input_list
    elif len(input_list)==2:
        sol=input_list
        if input_list[0]>input_list[1]:
            sol=[input_list[1],input_list[0]]
        return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

종료조건

분할

Conquer
Combine
Conquer
Combine
Conquer
Combine

정복



← 정렬된 2개의 부분 리스트

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

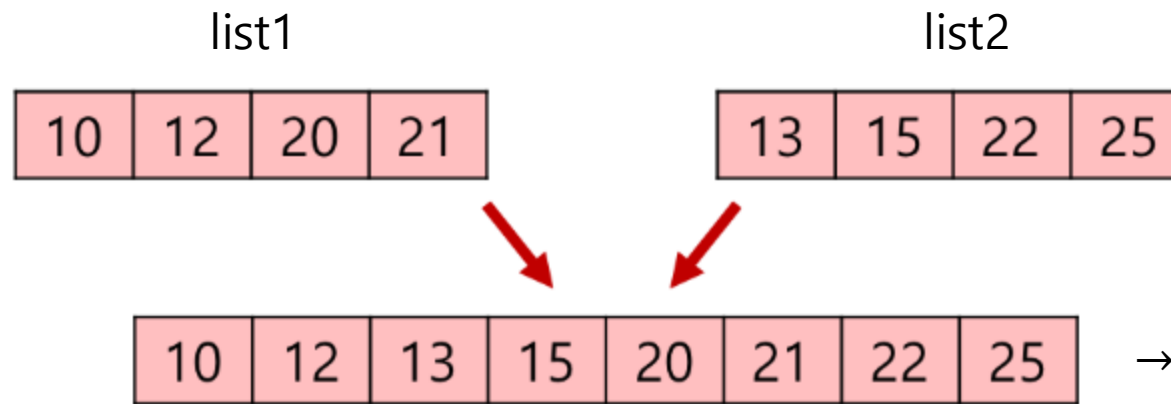
정복(조합)

오름차순
완성상태

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

Project 01. 합병

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



→merge(list1,list2) 의 return 값

Project 01. 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
---	---	---	----	----

Project 01. 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

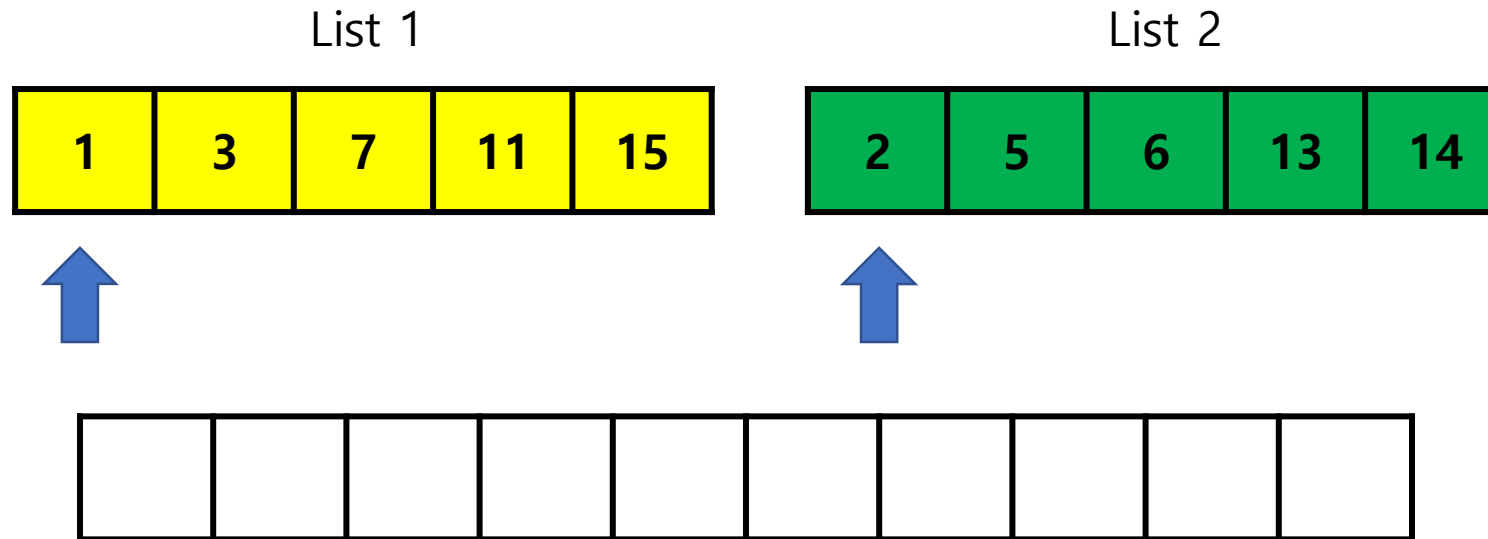
1	3	7	11	15
---	---	---	----	----

List 2

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

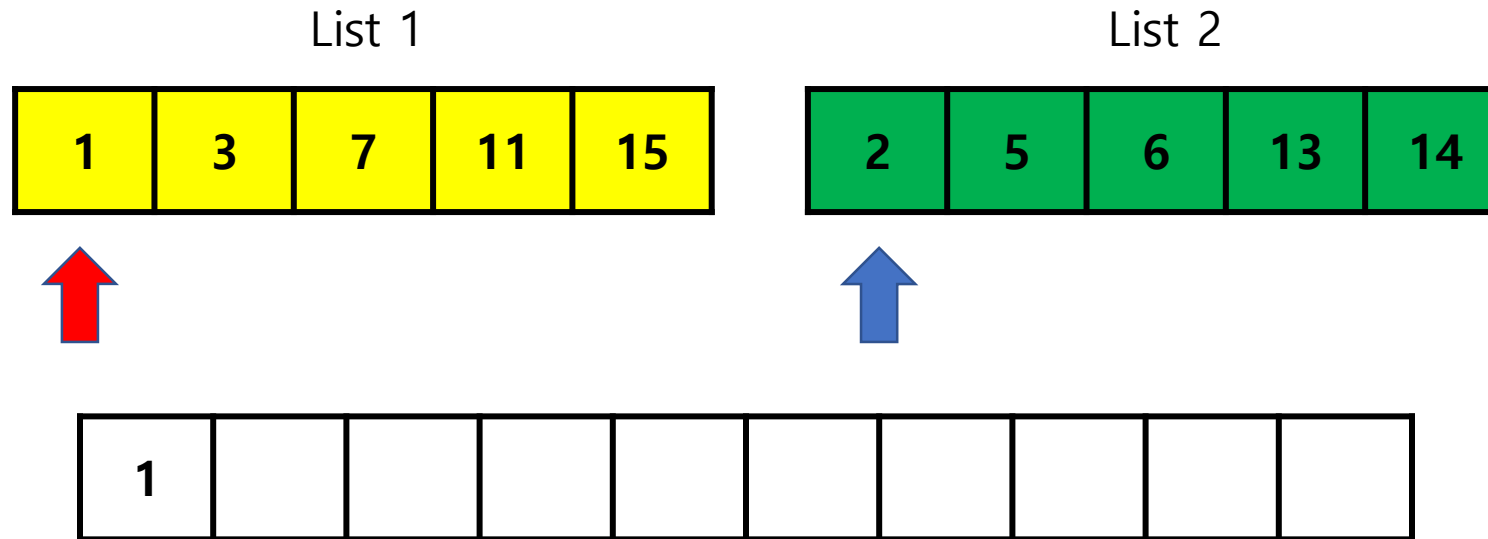
Project 01. 합병

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



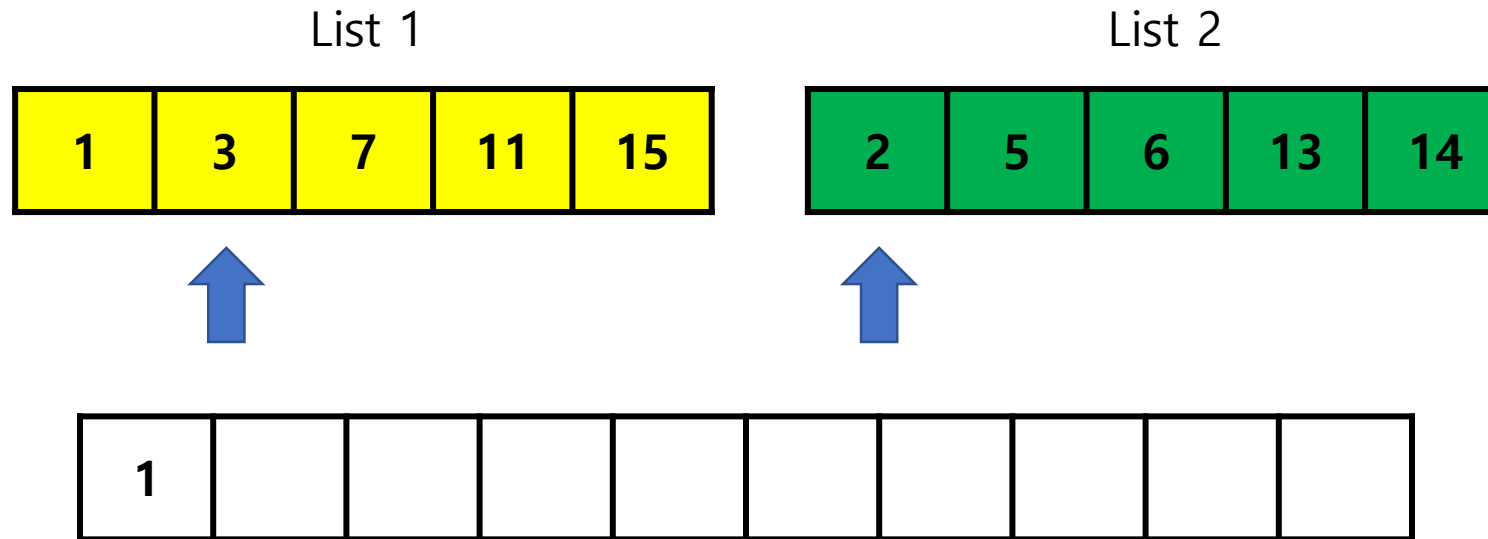
Project 01. 합병

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



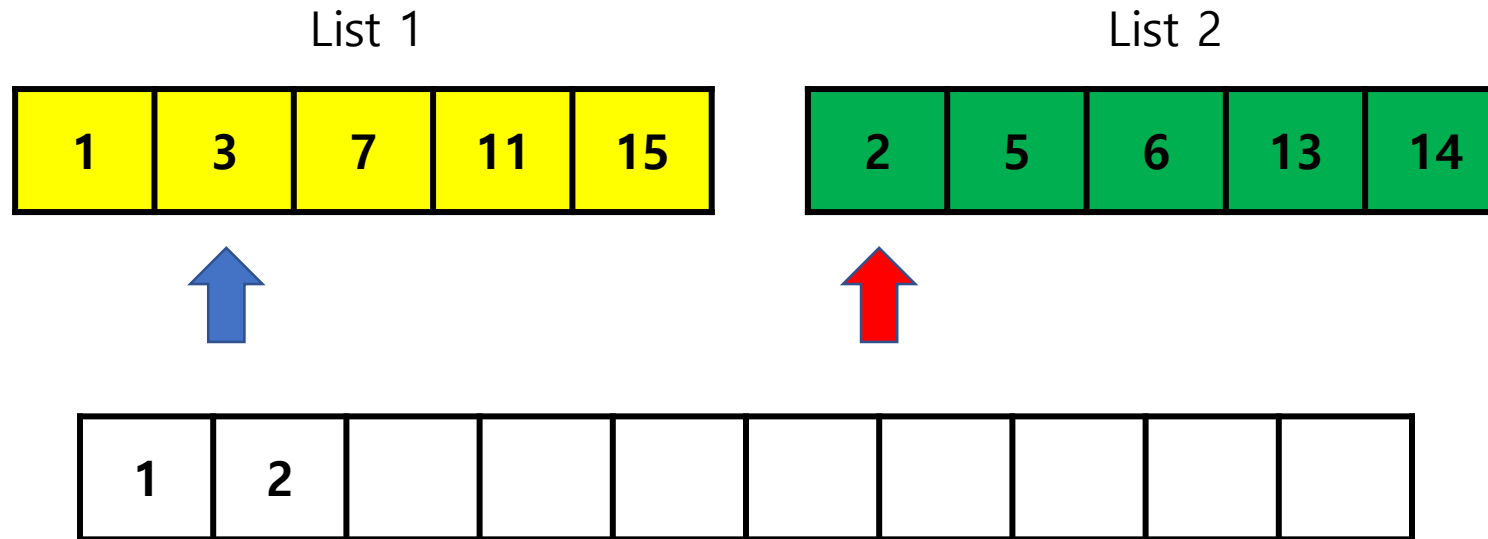
Project 01. 합병

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



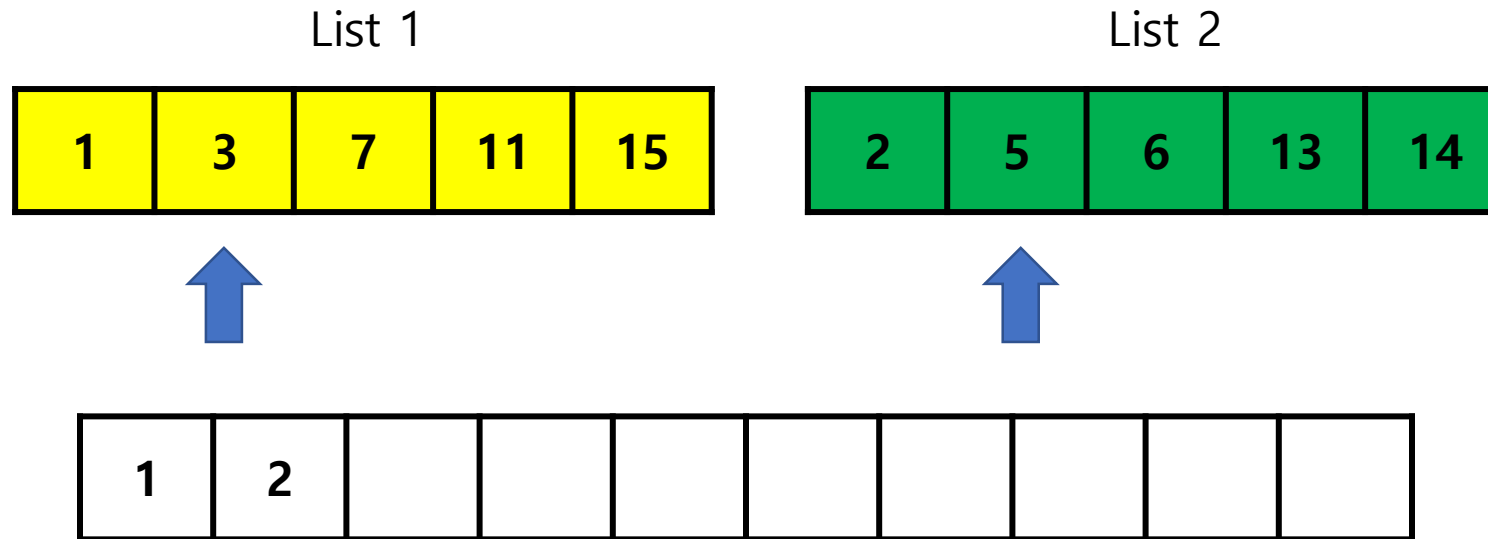
Project 01. 합병

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



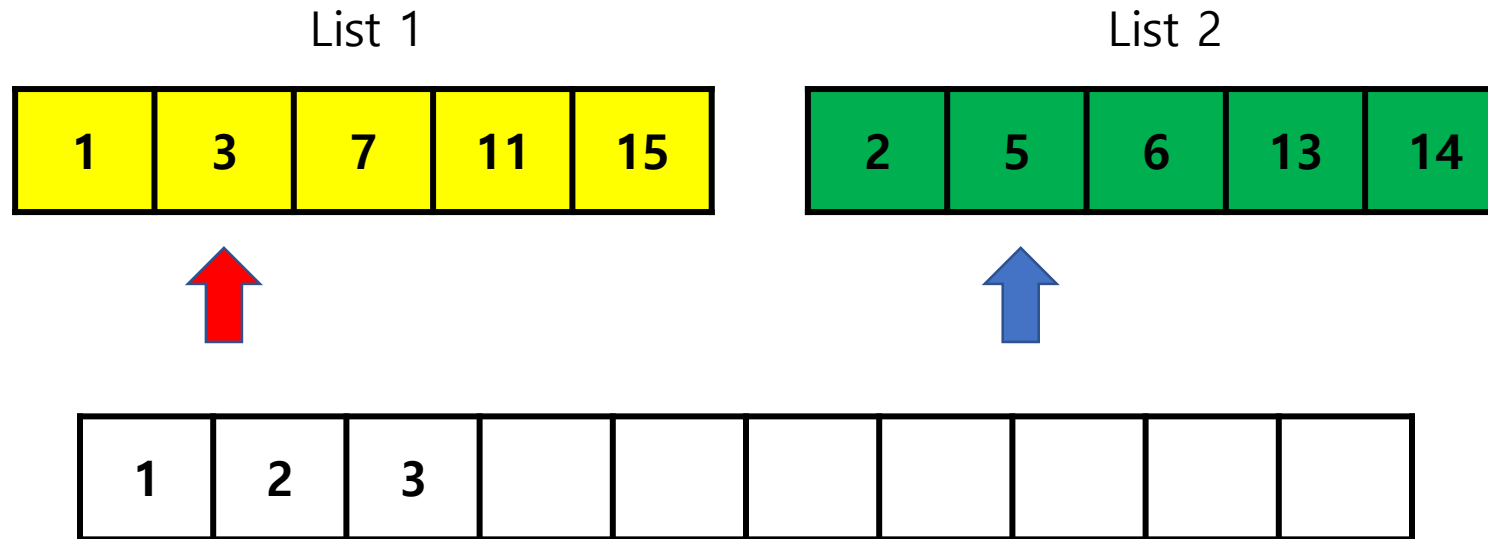
Project 01. 합병

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



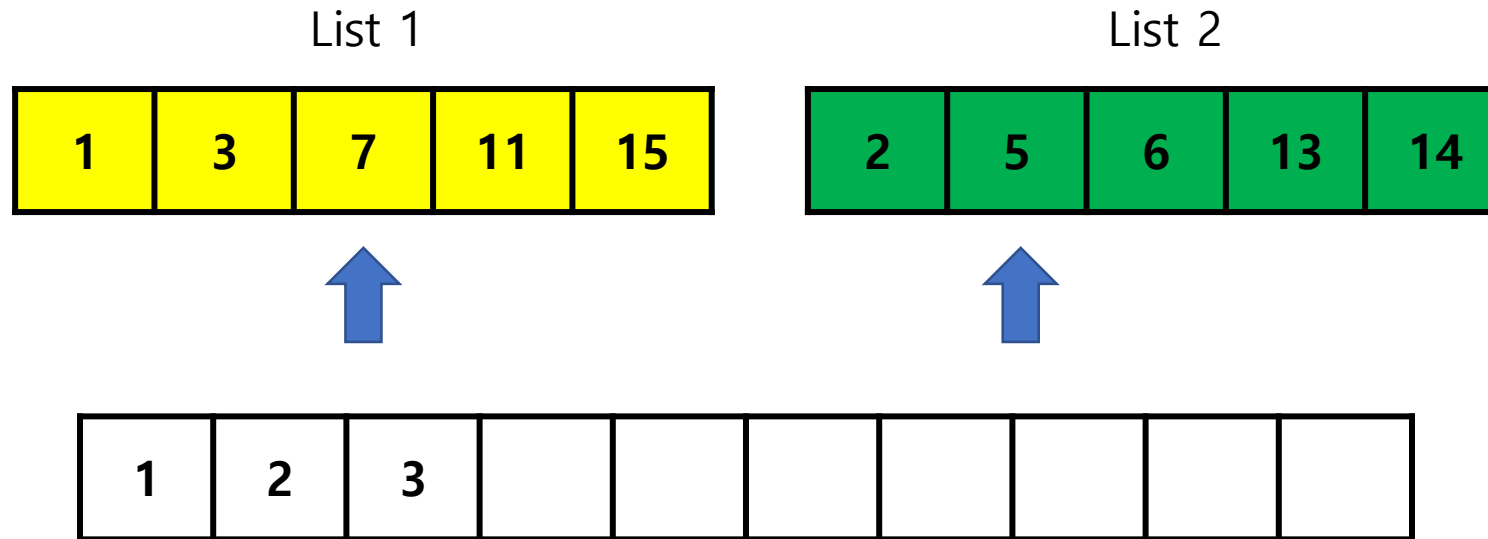
Project 01. 합병

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



Project 01. 합병

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



Project 01. 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
---	---	---	---	---	---	----	----	----	----

Project 01. 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

Project 02. 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
---	---	----	----	----	----	----	----	----



$$\text{Mid(중간)} = (0+8) // 2 = 4$$

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

Project 02. 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

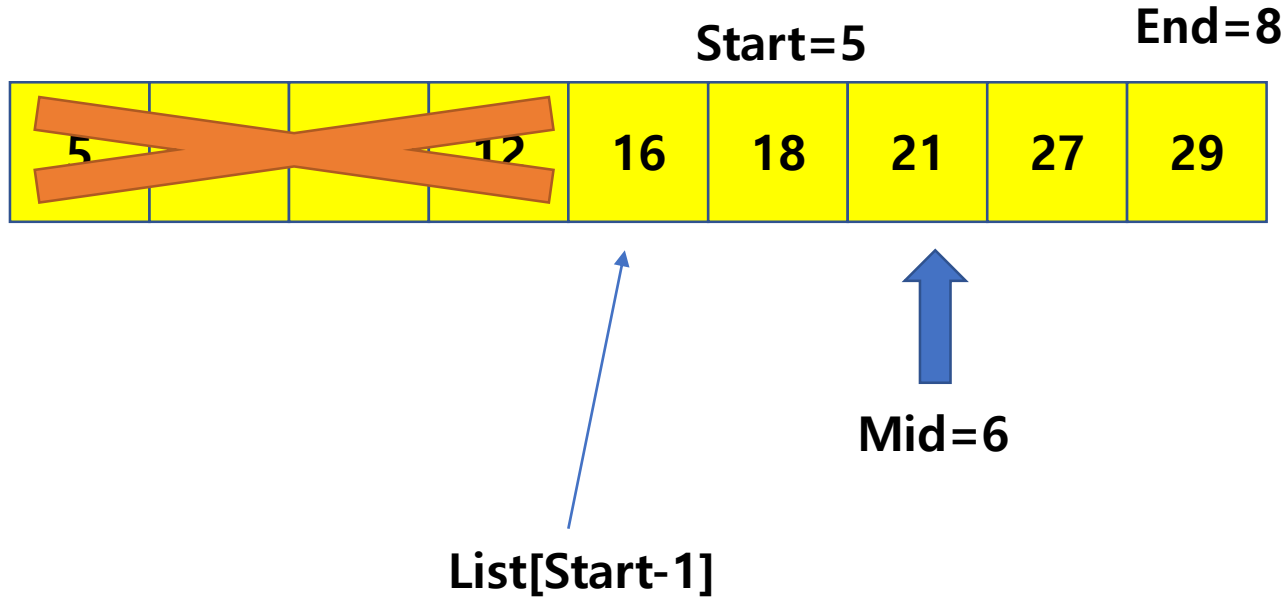
16 < 21 **오른쪽을 탐색!**

리스트: [21 22 38]

출력: []

Project 02. 이진 탐색 2

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

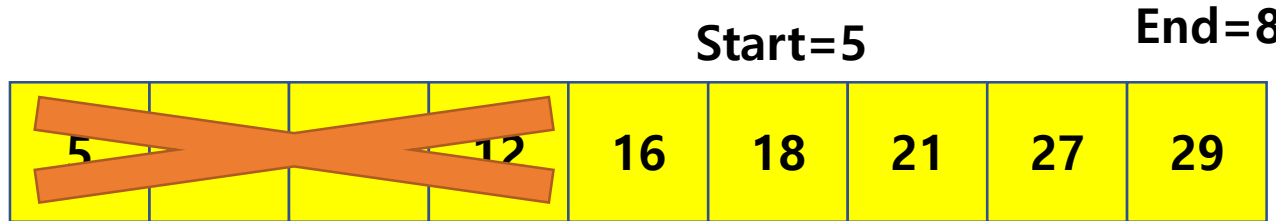


리스트: [21 22 38]

출력: []

Project 02. 이진 탐색 2

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



↑
Mid=6

리스트: [21 22 38]

출력: [21]

탐색 완료!

List[mid] = 21

값을 그대로 리턴!

Project 02. 이진 탐색 2

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

Start=0

End=8

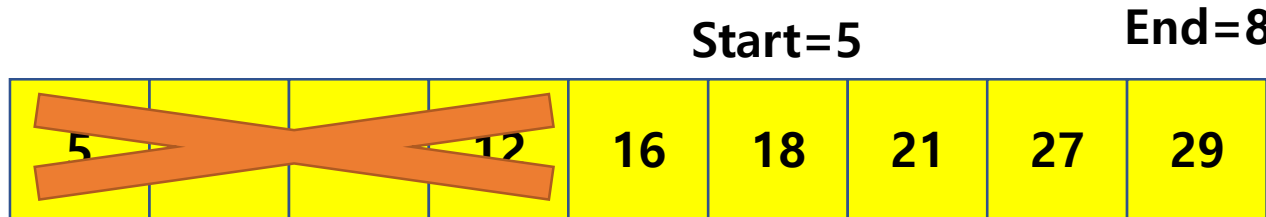
5	9	10	12	16	18	21	27	29
---	---	----	----	----	----	----	----	----

리스트: [21 **22** 38]

출력: [21]

Project 02. 이진 탐색 2

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



↑
Mid=6

리스트: [21 22 38]

출력: [21]

21 < 22 오른쪽을 탐색!

직전의 21과 비슷하게, 22를 찾는 과정이 진행됨. 여기서...

Project 02. 이진 탐색 2

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

Start=7 End=8



List[Start-1]

Mid=7

리스트: [21 22 38]

출력: [21]

27 > 22 왼쪽을 탐색!

Project 02. 이진 탐색 2

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

Start=End=7



Mid=7

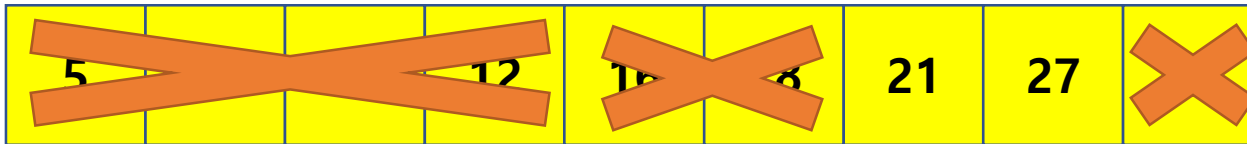
리스트: [21 22 38]

출력: [21]

Project 02. 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

Project 02. 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**]

38은 가장 큰 원소인 29보다도 크므로,
정답은 29