# Tree 형태 Solution Space 탐색 2

## 트리 탐색 과정과 유의사항에 익숙해 지기

01. 문제에서 사용하는 정의 이해: 약수의 합과 amicable chain

02. 첫 번째 알고리즘

03. 두 번째 알고리즘

04. 정리 문제 풀이

05. 실습 문제 풀이 & 질의 응답

수업자료에 첨부된 파일 중 Divisor.py를 미리 열어 두세요. 수업 중 실행해 보며 진행합니다.

lms에서 수정된 [수업자료] 파일이 올라가는 경우 파일 뒤에 "-[숫자]" 형식으로 번호가 추가되니, 수정되었다면 다시 받아 두세요.

<왜 Tree에 대해 배우는가?>

- Tree는 **여러 단계를 거쳐 하나의 해를 선택**하는 상황에서 **선택할 수 있는 모든 가능성**을 나타내는데 자연스러운 구조임
- 여러 가능한 해 중 **최적의 해를 선택**하기 위해서는 **Tree를 탐색**해야 함
- 이번 주에는 **효율적으로 Tree를 탐색하기 위해 고려할 것들**을 다시 한 번 연습해 보겠음

# 문제 정의: amicable chains

이 문제는 주어진 정수 a ≥ 0가 있을 때 **자기 자신을 제외한 약수의 합 s(a)**에 대한 문제이다.

예를 들어 a = 8인 경우 자기 자신을 제외한 약수는 {1, 2, 4, ~~8~~}이며, 따라서 s(8) = 1 + 2 + 4 = 7 이다.

단 s(a)의 초깃값 s(0) = 0, s(1) = 0이라 하자.

| | |
|---|---|
| 임의의 수 $a_1$에서 시작해서 | 예를 들어, s(12496) = 14288 |
| $s(a_1) = a_2$ | s(14288) = 15472 |
| $s(a_2) = a_3$ | s(15472) = 14536 |
| ... | s(14536) = 14264 |
| $s(a_i) = a_1$ | s(14264) = 12496 |
| 을 만족할 때, {$a_1$, $a_2$, $a_3$, ... $a_i$}를 amicable chain이라 한다. (※ amicable = friendly의 뜻) | 이며, 따라서 {12496, 14288, 15472, 14536, 14264}는 amicable chain이다. |

자연수 입력 N이 주어졌을 때, **모든 원소가 1≤ 이고 ≤ N인 가장 긴 amicable chain**을 찾으시오.

# 문제 풀이 과정

① **<u>이번 시간 문제를 잘 이해</u>**하기 위해 간단한 경우부터 시작해 문제 이해하고 답 이끌어내 보기

**문제 명확히 이해하는데 도움**

**이 과정에서 문제에 대한 해법도 생각해 낼 수 있음**

② 이번 시간 문제에 대한 **<u>풀이 방법 생각</u>**

③ 생각한 풀이 방법을 보다 **<u>효율적으로 개선</u>**

이 문제는 주어진 정수 a ≥ 0가 있을 때 **자기 자신을 제외한 약수의 합 s(a)**에 대한 문제이다.

예를 들어 a = 8인 경우 자기 자신을 제외한 약수는 {1, 2, 4}이며, 따라서 **s**(8) = 1 + 2 + 4 = 7 이다.

단 **s**(a)의 초깃값 **s**(0) = 0, **s**(1) = 0이라 하자.

(0) s(a)의 정의에 따르면 s(1) = 0이 맞는지 확인해 보시오. s(0)에 대해서도 생각해 보시오.

이 문제는 주어진 정수 a ≥ 0가 있을 때 **자기 자신을 제외한 약수의 합 s(a)**에 대한 문제이다.

예를 들어 a = 8인 경우 자기 자신을 제외한 약수는 {1, 2, 4}이며, 따라서 **s**(8) = 1 + 2 + 4 = 7 이다.

단 **s**(a)의 초깃값 **s**(0) = 0, **s**(1) = 0이라 하자.

(1) 작은 a 값에서 시작해 s(a)를 구하는 과정부터 이해해 보자. s(6)은 무엇인가?

(2) s(7)는 무엇인가?

(3) s(9)는 무엇인가?

| | |
|---|---|
| 임의의 수 $a_1$에서 시작해서<br>$s(a_1) = a_2$<br>$s(a_2) = a_3$<br>...<br>$s(a_i) = a_1$<br>을 만족할 때, {$a_1$, $a_2$, $a_3$, ... $a_i$}를 amicable chain 이라 한다. (※ amicable = friendly의 뜻) | 예를 들어, $s(12496) = 14288$<br>$s(14288) = 15472$<br>$s(15472) = 14536$<br>$s(14536) = 14264$<br>$s(14264) = 12496$<br>이며, 따라서 {12496, 14288, 15472, 14536, 14264}는 amicable chain이다. |

(4) 6은 amicable chain을 구성하는가?

(5) 8은 amicable chain을 구성하는가?

| | |
|---|---|
| 임의의 수 $a_1$에서 시작해서 | 예를 들어, s(12496) = 14288 |
| $s(a_1) = a_2$ | s(14288) = 15472 |
| $s(a_2) = a_3$ | s(15472) = 14536 |
| ... | s(14536) = 14264 |
| $s(a_i) = a_1$ | s(14264) = 12496 |
| 을 만족할 때, {$a_1$, $a_2$, $a_3$, ... $a_i$}를 amicable chain | 이며, 따라서 {12496, 14288, 15472, 14536, 14264} |
| 이라 한다. (※ amicable = friendly의 뜻) | 는 amicable chain이다. |

(6) 10은 amicable chain을 구성하는가?

**이 문제를 풀면서 amicable chain을 찾을 때 동적 프로그래밍을 활용하면 더 효율적임을 이해 하시오.**

(7) 220은 amicable chain을 구성하는가?

(8) 1184는 amicable chain을 구성하는가?

※ 주어진 수 a에 대한 모든 factor를 찾기 위해
이번 시간 수업 자료에 첨부된 Divisor.py의 findDivisors() 함수를 활용 하시오.
(웹에도 인수분해를 해주는 온라인 사이트가 많으니 활용하세요)

자연수 입력 N이 주어졌을 때 **모든 원소가 1≤ 이고 ≤ N인 가장 긴 amicable chain**을 찾으시오.

**amicable chain의 길이**는 amicable chain을 구성하는 서로 다른 **원소의 수**이다.

지금까지 보았던 amicable chain들의 길이는 무엇인가?

자연수 입력 N이 주어졌을 때 **모든 원소가 1≤ 이고 ≤ N인 가장 긴 amicable chain**을 찾으시오.

**amicable chain의 길이**는 amicable chain을 구성하는 서로 다른 **원소의 수**이다.

**지금까지 본 amicable chain들이 2,000 이하에서 볼 수 있는 모든 chain이라고 가정**하자.

(9) 입력 **N=100**일 때, **모든 원소가≤N인 가장 긴 amicable chain**은 무엇인가?

자연수 입력 N이 주어졌을 때 **모든 원소가 1≤ 이고 ≤ N인 가장 긴 amicable chain**을 찾으시오.

**amicable chain의 길이**는 amicable chain을 구성하는 서로 다른 **원소의 수**이다.

**지금까지 본 amicable chain들이 2,000 이하에서 볼 수 있는 모든 chain이라고 가정**하자.

(10) 입력 **N=2,000**일 때, 모든 원소가≤**N**인 가장 긴 amicable chain은 무엇인가?

※ **길이가 같은 chain이 2개 이상이라면 각 chain의 최소 원소를 비교하여 더 작은 쪽을 출력한다.**

자연수 입력 N이 주어졌을 때 **모든 원소가 1≤ 이고 ≤ N인 가장 긴 amicable chain**을 찾으시오.

**amicable chain의 길이**는 amicable chain을 구성하는 서로 다른 **원소의 수**이다.

**길이가 같은 chain이 2개 이상이라면 각 chain의 최소 원소를 비교하여 더 작은 쪽을 출력한다.**

※ **지금까지 본 amicable chain들이 2,000 이하에서 볼 수 있는 모든 chain이라고 가정**하자.

(11) 입력 N=250일 때 답은 무엇인가?

(12) 입력 N=1,200일 때 답은 무엇인가?

**이 문제를 풀면서,
임의의 자연수 입력 N이
주어졌을 때, 모든 원소≤N인
가장 긴 amicable chain은
어떻게 구하면 될지 생각해 보자.**

(13) 입력 N=30이 주어졌다. **모든 원소가 1≤ 이고 ≤ N**인 가장 긴 amicable chain을 찾아야 한다.
아래 표와 같이 s(0)~s(27)을 구했다고 가정하고 답을 찾아보자.
답을 찾아보면서 **<u>임의의 N이 주어졌을 때 일반적으로 어떤 과정을 거쳐 답을 찾아야 할지 생각</u>**해 보자.

| a | s(a) |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |
| 4 | 3 |
| 5 | 1 |
| 6 | 6 |
| 7 | 1 |
| 8 | 7 |
| 9 | 4 |
| 10 | 8 |
| 11 | 1 |
| 12 | 16 |
| 13 | 1 |
| 14 | 10 |
| 15 | 9 |
| 16 | 15 |
| 17 | 1 |
| 18 | 21 |
| 19 | 1 |
| 20 | 22 |
| 21 | 11 |
| 22 | 14 |
| 23 | 1 |
| 24 | 36 |
| 25 | 6 |
| 26 | 16 |
| 27 | 13 |
| 28 | |
| 29 | |
| 30 | |

| a | s(a) |
|---|---|
| … | … |
| 220 | 284 |
| … | … |
| 284 | 220 |
| … | … |
| 1184 | 1210 |
| … | … |
| 1210 | 1184 |
| … | … |

**이 문제를 풀면서,
사용한 방법의 시간 복잡도도
N을 사용해 나타내 보자.
(s(a)는 이미 0~N까지
계산되어 있다고 가정)**

| a | s(a) |
|---|---|
| 12496 | 14288 |
| … | … |
| 14264 | 12496 |
| … | … |
| 14288 | 15472 |
| … | … |
| 14536 | 14264 |
| … | … |
| 15472 | 14536 |
| … | … |

# 첫 번째 알고리즘

```
# s[]: s(a) 값을 저장하는 배열

s[0]=0; s[1]=0;  // s[] 초기화
for a=2~N
   s(a) 구해 s[a]에 저장 (즉, a의 모든 약수 찾고 a 자신 제외한 약수 모두 더해 s[a]에 저장)

for a=2~N
    if 아직 s[a]에 대한 amical chain 검사를 하지 않았다면
        s[a]에서 시작해 s[a]로 돌아오는 chain 있는지 검사해서, 기존에 찾은 chain보다 긴 것이 있다면 기록
```

하나의 for loop으로 구현할 수도 있으나, 서로 다른 작업의 복잡도 비교가 편리하여 두 개의 for loop으로 나눔

두 번째 for loop에 대한 시간 복잡도를 N을 사용해 나타내 보자.

# 문제 풀이 과정

① **이번 시간 문제를 잘 이해**하기 위해 간단한 경우부터 시작해 문제 이해하고 답 이끌어내 보기

② 이번 시간 문제에 대한 **풀이 방법 생각**

③ 생각한 풀이 방법을 보다 **효율적으로 개선**

**기본 해법에서 시간이 가장 많이 걸리는 부분 파악하고 이를 개선해 보자.**

# 첫 번째 알고리즘

```
# s[]: s(a) 값을 저장하는 배열

s[0]=0; s[1]=0;  // s[] 초기화
for a=2~N
   s(a) 구해 s[a]에 저장 (즉, a의 모든 약수 찾고 a 자신 제외한 약수 모두 더해 s[a]에 저장)

for a=2~N
    if 아직 s[a]에 대한 amical chain 검사를 하지 않았다면
        s[a]에서 시작해 s[a]로 돌아오는 chain 있는지 검사해서, 기존에 찾은 chain보다 긴 것이 있다면 기록
```

(14) 위에 제시한 방법에서 **가장 시간이 많이 소요되는 작업(또는 연산)**은 무엇인가? 속도를 개선하기 위해서는 우선 이 작업을 보다 효율적으로 개선해야 할 것이다.

※ Hint: 모든 과정을 손으로 수행한다고 생각해 보자. 어떤 작업이 가장 오래 걸릴까? 작업 A와 B를 사람이 수행할 때 (상대적으로) A보다 B에 더 많은 시간이 소요된다면, 컴퓨터가 수행할 때도 일반적으로 A보다 B에 더 많은 시간이 소요된다.

# a = 2~N 각각에 대해 모든 약수 찾아 s[a] 구하는 기본 방법

```
s = [0 for _ in range(n+1)]   # s[0~n]을 모두 0으로 초기화
for a in range(2, n+1):    # a = 2~n 범위에 대해
    for i in range(1, a):    #  i = 1~(a-1) 범위 값에 대해 (즉 a 자신은 제외한 값을 탐색)
        if a % i == 0: s[a] += i    # a % i == 0 이라면 i가 a의 약수이므로 s[a]에 i를 더함
```

```
s = [0 for _ in range(n+1)]   # s[0~n]을 모두 0으로 초기화
for a in range(2, n+1):    # a = 2~n 범위에 대해
    for i in range(1, a):    #  i = 1~(a-1) 범위 값에 대해 (즉 a 자신은 제외한 값을 탐색)
        if a % i == 0: s[a] += i    # a % i == 0 이라면 i가 a의 약수이므로 s[a]에 i를 더함
```

(15) 먼저 위 코드가 무엇을 하는지부터 이해해 보자.
N=10인 경우에 대해 a 값과 i 값이 어떤 값을 가지는지를 가능성 트리로 그려 보시오.

```
<Divisor.py>
def findSarrayDiv(n):
    s = [0 for _ in range(n+1)]
    for a in range(2,n+1):
        for i in range(1, a):
            if a % i == 0: s[a] += i
    return s
```

```
<Divisor.py>
if __name__ == "__main__":
    …
    print(findSarrayDiv(30))
    print(findSarrayDiv(1210))
```

이번 시간 수업 자료에 첨부된 Divisor.py에는 앞에서 본 방식대로 약수의 합을 구하는 findSarrayDiv(n) 함수가 있다. 먼저 **이 함수를 실행해 결과가 올바름을 확인**해 보자. __main__ 아래에서 위 오른쪽과 같은 두 라인을 주석 해제해 실행해 보자.

(16) 입력 n=30을 주어 결과를 출력해 보자. 계산한 s[] 값이 올바른가?

(17) 입력 n=1210을 입력해 보자. 계산한 s[1210] 값이 올바른가?

<Divisor.py>

```
if __name__ == "__main__":
    …
    # Measure average running time of findSarray()
    findSarray = findSarrayDiv
    n, repeat = 10, 1
    tfindSarray = timeit.timeit(lambda: findSarray(n), number=repeat)/repeat
    print(f"Average running time of findSarray({n}) is {tfindSarray:.10f} seconds")
```

이제 findSarrayDiv 함수의 **실행 속도를 측정**해 보자. __main__ 아래에서 위 라인을 주석 해제해 실행해 보자.

(18) N <= $10^6$까지 수 초 이내에 답이 나오도록 코드를 작성해야 한다고 가정하자. N=10, 100, 1000, 10000, 100000, 1000000 각각에 대해 수행 시간을 측정해 보자. N이 최댓값일 때도 (사람이 느끼기에) 거의 즉시 답을 얻을 수 있는가? 혹은 그렇지 않아 시간을 단축해야 하는가?

(19) 앞에서 N=10에 대해 그린 가능성 트리를 다시 보자. 약수를 구하기 위해 **확인해볼 필요가 없는 가지**는 무엇인가? 이를 확인하지 않으려면 for 루프의 조건을 어떻게 수정해야 하나?

```
def findSarrayDiv(n):
    s = [0 for _ in range(n+1)]
    for a in range(2,n+1):
        for i in range(1, a):
            if a % i == 0: s[a] += i
    return s
```

(20) 앞 문제에서 제시한 것과 같이 for 루프를 수정한 후 N=30, N=1210에 대해 계산한 s[] 값이 올바른지 확인 하시오.

(21) N=10, 100, 1000, 10000, 100000, 1000000(문제에서 제시한 N의 최댓값) 각각에 대해 수행 시간을 다시 측정해 보자. 앞에서 측정한 값에 비해 얼마나 단축되었는가?

이번에는 N이 최댓값일 때도 (사람이 느끼기에) 거의 즉시 답을 얻을 수 있는가? 혹은 그렇지 않아 시간을 더 단축해야 하는가?

**N=10에 대해 그린 가능성 트리를 다시 보고 더 개선할 방법을 생각해 보자.**

```
def findSarrayDivSqrt(n):
    s = [0 for _ in range(n+1)]
    for a in range(2,n+1):
        for i in range(1, int(math.sqrt(a))+1):
            if a % i == 0:
                s[a] += i
                tmp = int(a/i)
                if tmp != i and tmp != a: s[a] += tmp
    return s
```

**개선한 방법으로 올바른 답이 나오는지 확인해 본 후 수행 시간도 다시 측정해 보자.**

```
def findSarrayDivSqrt(n):
    s = [0 for _ in range(n+1)]
    for a in range(2,n+1):
        for i in range(1, int(math.sqrt(a))+1):
            if a % i == 0:
                s[a] += i
                tmp = int(a/i)
                if tmp != i and tmp != a: s[a] += tmp
    return s
```

```
print(findSarrayDivSqrt(30))
print(findSarrayDivSqrt(1210))

# Measure average running time of findSarray()
findSarray = findSarrayDivSqrt
n, repeat = 10000, 1
tfindSarray = timeit.timeit(lambda: findSarray(n), number=repeat)/repeat
print(f"Average running time of findSarray({n}) is {tfindSarray:.10f} seconds")
```

# 필기

(22) 앞에서 N=10에 대해 그린 가능성 트리를 다시 보자. N이 커질수록 불필요한 탐색 시간이 많이 걸리는 주된 이유는 무엇인가? 이를 해결할 수 있는 방법을 생각해 보시오.

(22) 앞에서 N=10에 대해 그린 가능성 트리를 다시 보자 **(a=2~10에 대한 약수를 모두 구해 s[a]에 더해야 함)**. N이 커질수록 불필요한 탐색 시간이 많이 걸리는 주된 이유는 무엇인가? 이를 해결할 수 있는 방법을 생각해 보시오.

(23) 아래 코드가 무엇을 하는 코드인지 이해해 보시오.
작성 후에는 N=30, N=1210에 대해 계산한 s[] 값이 올바른지 확인 하시오.

<Divisor.py>

```
def findSarrayMult(n):
    s = [0 for _ in range(n+1)]
    for n1 in range(1, int(n/2)+1):
        for n2 in range(2, int(n/n1)+1):
            s[n1*n2] += n1
    return s
```

```
if __name__ == "__main__":
    …
    print(findSarrayMult(30))
    print(findSarrayMult(1210))
```

(24) 이제 새로운 방법에 대해 N=10, 100, 1000, 10000, 100000, 1000000 (문제에서 제시한 N의 최댓값) 각각에 대해 수행 시간을 다시 측정해 보자. 앞에서 측정한 값에 비해 얼마나 단축되었는가? 이번에는 N이 최댓값일 때도 (사람이 느끼기에) 거의 즉시 답을 얻을 수 있는가?

```
# Measure average running time of findSarray()
findSarray = findSarrayMult
n, repeat = 10000, 1
tfindSarray = timeit.timeit(lambda: findSarray(n), number=repeat)/repeat
print(f"Average running time of findSarray({n}) is {tfindSarray:.10f} seconds")
```

(25) 앞에서 N=10에 대해 그린 가능성 트리를 다시 보자. 탐색하는 가지의 수 or 불필요한 곱셈 횟수를 더 줄일 수 있는 방법은 없을까?

(26) 아래 코드를 읽어보고 기존 코드에서 변경된 부분이 무엇인지 이해 하시오.
그리고 N=30, N=1210에 대해 계산한 s[] 값이 올바른지 확인 하시오.

```
def findSarrayMultSqrt(n):
    s = [0 for _ in range(n+1)]
    for a in range(2,n+1): s[a]=1
    for n1 in range(2, int(math.sqrt(n))+1):
        for n2 in range(n1, int(n/n1)+1):
            s[n1*n2] += n1
            if n2 != n1: s[n1*n2] += n2
    return s
```

```
if __name__ == "__main__":
    …
    print(findSarrayMultSqrt(30))
    print(findSarrayMultSqrt(1210))

    # Measure average running time of findSarray()
    findSarray = findSarrayMultSqrt
    n, repeat = 1000000, 1
    tfindSarray = timeit.timeit(lambda: findSarray(n), number=repeat)/repeat
    print(f"Average running time of findSarray({n}) is {tfindSarray:.10f} seconds")
```

**<정리 문제>**

(27) 이번 시간에도 지난 시간에서와 같이 트리 형태의 solution space를 탐색하였다.
지난 시간에는 탐색을 위해 재귀호출을 사용하였는데 (혹은 재귀호출의 동작을 모방한 코드 작성),
이번 시간에는 왜 재귀호출을 사용하지 않았나?

(28) 이번 시간 문제에서는 N이 입력으로 주어졌을 때 **a=2~N까지의 여러 수 모두에 대한 약수를 구해야 했다**. 이를 위해 ① 처음에 사용한 방법에서는 각 a를 여러 수로 나누어 보며 약수를 구했다. 이 방법에는 불필요한 나눗셈이 많이 필요하다는 단점이 있어 이를 개선하기 위해 ② 여러 다른 수로 곱셈을 해 보며 약수를 구하는 방법을 사용했다.

**[문제]** 문제를 바꾸어 **하나의 수 a가 입력으로 주어졌을 때 a에 대한 모든 약수를 구해야 한다고 가정**하자. 이러한 경우에도 방법 ②가 방법 ①보다 더 효율적인가?

(29) 입력으로 정수 N이 주어졌을 때 a=1~N 까지 수 중 제곱근이 정수인 수를 모두 구하기 위해 아래 코드를 사용했다. 이 코드를 이해한 후 성능을 개선하는 방법을 제안 하시오.

```
<Divisor.py>
def review29(n):
    result = []
    for a in range(1,n+1):
        if math.sqrt(a).is_integer(): result.append(a)
    return result
```

**float type 변수에 대한 is_integer() 함수는 변수에 저장된 값이 정수인지 (즉 소수점 아래가 0인지) 판단해 True/False를 반환함**
**예: a=3.1 이라면 a.is_integer() = False. a=3.0 이라면 a.is_integer() = True**

(30) 자연수 입력 N과 p가 주어졌을 때 $N^p$를 계산하는 아래 두 방식 ①과 ②를 비교해 보자. 어느 방식에서 곱셈 연산의 횟수가 더 적은가? 이유는 무엇인가? ※ $a^2 = a \times a$ 로 계산한다.

| 계산할 값 $N^p$ | 방법 ① | 방법 ② |
|---|---|---|
| $2^4$ | 2×2×2×2 | $(2\times2)^2$ |
| $2^5$ | 2×2×2×2×2 | $(2\times2)^2 \times2$ |
| $2^{10}$ | 2×2×2×2×2×2×2×2×2×2 | $( (2\times2)^2 \times2 )^2$ |
| $2^{16}$ | 2×2×2×2×2×2×2×2<br>×2×2×2×2×2×2×2×2 | 방법 ②로는 어떻게 구해야 하나? |



**<정리>**

- 이번 주 문제에서 solution space의 탐색 시간을 줄일 수 있었던 이유를 요약하면 아래 ①~②와 같다. 이 중 ①에 의한 기여가 더 크다.

  ① 나누어 떨어지지 않는 수로 굳이 나누어 보지 않도록 하여 연산 횟수 줄임

  ② 같은 횟수의 연산을 하더라도 나눗셈 연산보다 곱셈 연산이 더 빠름

- 위 ①을 다르게 표현하면 다음과 같다. Solution space를 탐색하며 해를 찾는 문제를 풀 때는 가능하면 불필요한 탐색을 제거하여 탐색하는 범위를 줄여나가는 것이 좋다. 이 때 탐색 범위를 최소로 만드는 것은 **직접 해로가는 방법**이다(즉 탐색하는 것 마다 100% 문제에 대한 해인 경우임). 이번 주 문제에서 곱셈 tree를 사용한 방법도 tree의 매 가지마다 유효한 해를 하나씩 얻었으므로 탐색 범위를 최소화 한 것으로 볼 수 있다. 의외로 많은 경우에 이와 같이 탐색 범위를 최소화하는 것이 가능하므로 탐색 문제에서는 항상 직접 해로가는 방법이 있는지 생각해 보자.

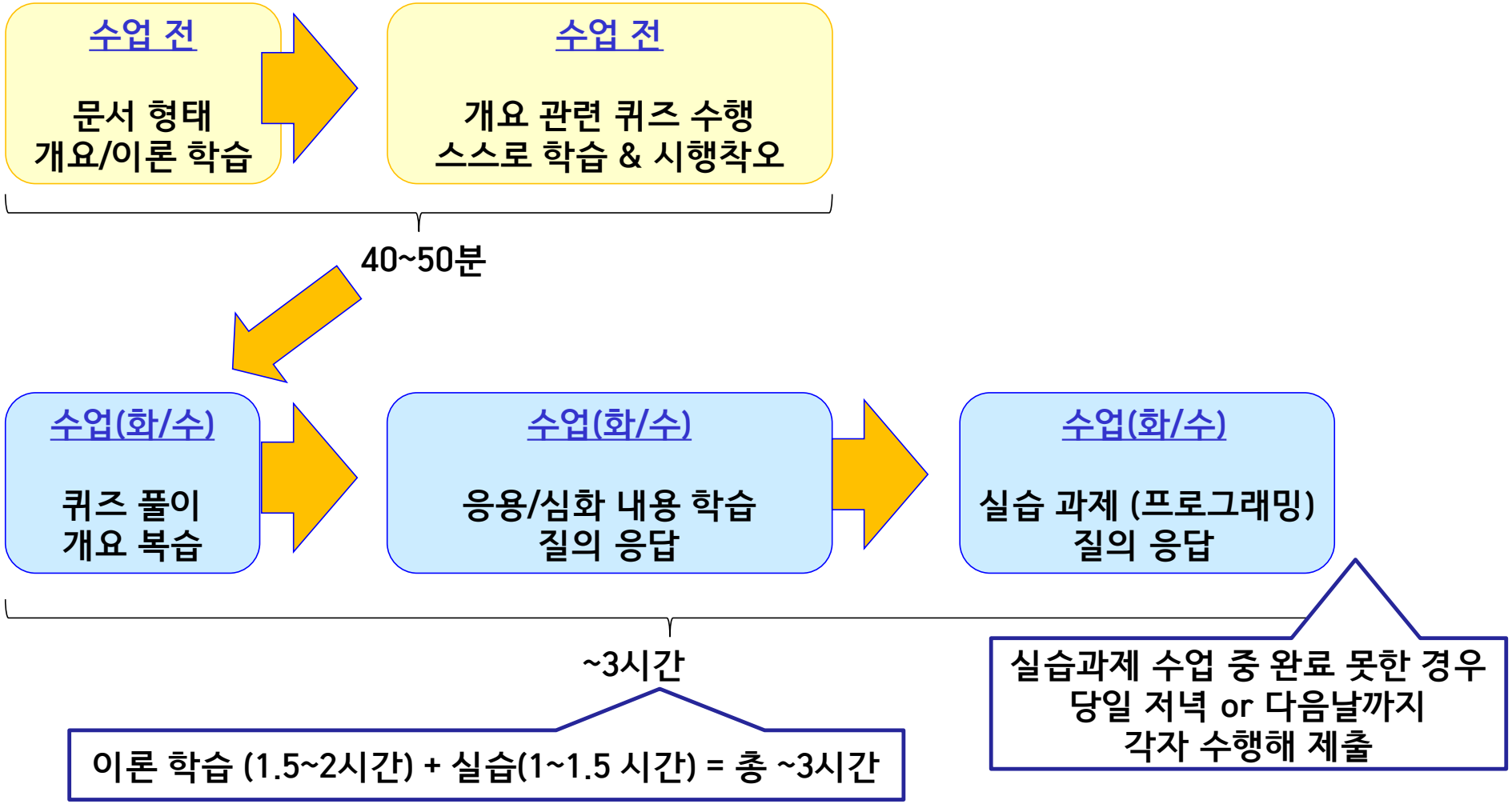
**<정리>**

- 프로그램 작성 시 **수행 시간을 직접 측정해 보는 습관**을 들이자. 각 구간의 수행 시간을 정확한 숫자로 체감한다면 특정 구간의 성능을 개선할 필요가 있는지, 코드 수정 전후에 성능이 어느 정도로 개선되었는지, 그리고 성능이 충분이 개선 되었으므로 성능 개선 작업을 중단해도 괜찮은지 등을 판단하는데 도움이 된다.

**<정리>**

- 새로운 문제를 풀 때는 **간단한 경우에서 시작**해 **손으로** 풀어가 보며 **문제를 충분히 이해할 수 있는 시간**을 갖자. 이러한 과정 중에 문제에 대한 첫 번째 해결 방법을 떠올릴 수 있을 것이고, 그 후에는 이를 보다 효율적으로 만드는 것에 집중하면 된다.
- **처음에는 비효율적인 방법(예: 전 범위를 탐색하는 brute-force algorithm)에서 시작**하여도 괜찮다. 이로부터 **중복된 경우**를 탐색하는 경우를 발견하며 탐색 범위를 **조금씩 줄여나가다** 보면 훨씬 더 효율적인 방법에 도달할 수 있을 것이다.
- 문제에 대한 해법을 더 효율적으로 개선하기에 앞서 반드시 **어느 부분이 가장 시간이 많이 소요되는 부분(major performance bottleneck)인지를 생각**해 보고 이 부분을 개선해 나가자. 그 외의 중요하지 않은 부분을 개선한다면 전체적인 성능이 크게 개선되지 않으므로 대신 중요한 부분을 개선하는데 더 많은 시간을 투자하자.
- 때때로 처음 생각한 **방법을 완전히 뒤집어 생각**함으로써 더 나은 방법에 도달하기도 한다. 항상 여러 다른 해법을 생각해 보고 어느 방법이 나은지 비교해 보자.

# 스마트 출결



# 05. 실습문제풀이 & 질의응답

- **이번 시간에 배운 내용에 대한 실습 문제 풀이 & 질의 응답**
- **채점 방식은 지난 시간 문제 풀이때와 유사함**

- **왜 중요한가?**
- 이번 주 배운 내용을 총괄하는 문제 풀이 통해 배운 내용 활용 & 복습
- 문제 풀이 점수는 이번 주 과제 점수에 포함됨

# findLongestAmicableChain (n) 함수 구현 조건: amicable chain 탐색 코드 작성

- 자연수 **n**이 입력으로 주어졌을 때, 모든 원소가 1~**n** 범위에 속하는 가장 긴 amicable chain을 찾아 반환

  def findLongestAmicableChain (**n**):

- 입력 **n**: $6 \leq n \leq 10^6$ 범위의 정수
  - 위 범위를 벗어나는 값은 입력으로 들어오지 않는다고 가정 (즉 오류 처리 하지 않아도 됨)

- 반환 값: 모든 원소가 1~**n** 범위에 속하는 가장 긴 amicable chain에 속하는 모든 원소를 담은 리스트
  - 같은 숫자를 두 번 이상 중복으로 담으면 안 됨
  - 원소를 담는 순서는 중요하지 않으며, 모든 원소가 빠짐없이 담겨 있으면 됨
  - 길이가 같은 chain이 둘 이상이라면, 각 chain의 최소 원소를 비교해 더 작은 쪽을 반환
  - 예: **n**=1000일 때, 수업 중에 보았던 것처럼 길이가 2인 chain이 가장 김. 이들은 [220,284] 및 [1184,1210]인데, 최소 원소가 더 작은 쪽을 반환하므로 반환 값은 [220, 284]

- 이번 시간에 제공한 코드 AmicableChain.py의 findLongestAmicableChain() 함수 내에 코드 작성해 제출

# 입출력 예 (모든 원소가 1~n 범위 속하는 가장 긴 amicable chain 반환)

```
print(findLongestAmicableChain(10))
```

```
[6]
```

```
print(findLongestAmicableChain(100))
```

```
[6]
```

```
print(findLongestAmicableChain(1000))
```

```
[220,284]
```

```
print(findLongestAmicableChain(10000))
```

```
[220,284]
```

```
print(findLongestAmicableChain(100000))
```

```
[12496, 14288, 15472, 14536, 14264]
```

그 외 예제는 __main__ 아래 테스트 코드를 참조하세요.

# 그 외 프로그램 구현 조건

- 최종 결과물로 AmicableChain.py 파일 하나만 제출하며, 이 파일만으로 코드가 동작해야 함

- import는 사용할 수 없음

- __main__ 아래의 코드는 작성한 함수가 올바름을 확인하는 코드로
- 코드 작성 후 스스로 채점해 보는데 활용하세요.
- 각 테스트 케이스에 대해 P(or Pass) 혹은 F(or Fail)이 출력됩니다.
- 코드를 제출할 때는 __main__ 아래 코드는 제거하거나 수정하지 말고 제출합니다. (채점에 사용되기 때문)

# 이번 시간 제공 코드 함께 보기

- AmicableChain.py

# 실습 문제 풀이 & 질의 응답

- 종료 시간(17:00) 이전에 **일찍 모든 문제를 통과**한 경우 각자 **퇴실 가능**
- 실습 문제에 대한 코드는 lms 과제함에 **다음 날 23:59까지 제출** 가능합니다.
- 마감 시간까지 작성을 다 못한 경우는 (제출하지 않으면 0점이므로) 그때까지 작성한 코드를 꼭 제출해 부분점수를 받으세요.

- 실습 문제는 **개별 평가**입니다. 제출한 코드에 대한 유사도 검사를 실습 문제마다 진행하며, 코드를 건내 준 사례가 발견되면 0점 처리됩니다.
- 로직에 대해 서로 의견을 나누는 것은 괜찮지만, 코드를 직접 건내 주지는 마세요.
- 본인 실력 향상을 위해서도 **코드는 꼭 각자 직접 작성**해 주세요.