Chapter 03.
순열과 조합

Clip 01 | 재귀 함수 이론
재귀 함수의 정의와 예시
[2747] 피보나치 수

Clip 02 | [15654] N과 M(5)
중복이 없는 사전순 순열 출력

Clip 03 | [15655] N과 M(6)
중복이 없는 사전순 오름차순 순열 출력

Clip 04 | [15656] N과 M(7)
원소 중복이 가능한 사전순 순열 출력

Clip 05 | [15657] N과 M(8)
원소 중복이 가능한 사전순 비내림차순 순열 출력

Ch03. 순열과 조합

1. 재귀 함수 이론

재귀함수 (Recursive Function)

자기 자신을 호출하는 함수

ex)

```
class Main {
    public void myFunction(int n) {
        // do something
        myFunction(n - 1);
        // do something
    }
}
```

myFunction() 함수 안에서
myFunction() 을 다시 호출하고 있다

왜 사용할까?

1. 하나의 커다란 문제를
   작은 문제로 나누어 해결

2. 문제를 귀납적으로 생각하기 위해서
- {i} 번째 답을 구하기 위해 (i -1), (i -2)
  번째 결과 활용

재귀함수 예시 – 숫자출력

숫자를 출력하는 함수를 작성

1. 1부터 N까지 오름차순으로 출력
2. N부터 1까지 내림차순으로 출력

단, 반복문을 사용하지 않고 재귀함수를 이용해 작성

재귀함수 예시 – 숫자출력

```
public static void main(String[] args){
    PrintNumber pn = new PrintNumber();

    System.out.println("1부터 10까지 오름차순 출력");
    pn.asc(10);

    System.out.println();

    System.out.println("1부터 10까지 내림차순 출력");
    pn.desc(10);
}
```

```
class PrintNumber {
    public void asc(int n) {

    }
    public void desc(int n) {

    }
}
```

```
1부터 10까지 오름차순 출력
1 2 3 4 5 6 7 8 9 10
1부터 10까지 내림차순 출력
10 9 8 7 6 5 4 3 2 1
Process finished with exit code 0
```

재귀함수 예시 – 숫자출력

```
public static void main(String[] args){
    PrintNumber pn = new PrintNumber();

    System.out.println("1부터 10까지 오름차순 출력");
    pn.asc(10);

    System.out.println();

    System.out.println("1부터 10까지 내림차순 출력");
    pn.desc(10);
}
```

```
class PrintNumber {
    public void asc(int n) {
        if(n == 0) return;

        asc(n - 1);
        System.out.printf("%d ", n);
    }

    public void desc(int n) {
        if(n == 0) return;

        System.out.printf("%d ", n);
        desc(n - 1);
    }
}
```

재귀함수 예시 – 숫자출력

```
class PrintNumber {
    public void asc(int n) {
        if(n == 0) return;

        asc(n - 1);
        System.out.printf("%d ", n);
    }

    public void desc(int n) {
        if(n == 0) return;

        System.out.printf("%d ", n);
        desc(n - 1);
    }
}
```

```
asc(5)
 ㄴ if(5 == 0) return; (리턴 X)
 ㄴ asc(4)
     ㄴ if(4 == 0) return; (리턴 X)
     ㄴ asc(3)
         ㄴ if(3 == 0) return; // (리턴 X)
         ㄴ asc(2)
             ㄴ if(2 == 0) return; // (리턴 X)
             ㄴ asc(1)
                 ㄴ if(1 == 0) return; // (리턴 X)
                 ㄴ asc(0)
                     ㄴ if(0 == 0) return;
                 ㄴ printf("%d ", 1)
             ㄴ printf("%d ", 2)
         ㄴ printf("%d ", 3)
     ㄴ printf("%d ", 4)
 ㄴ printf("%d ", 5)
```

재귀함수 예시 – 숫자출력

```
class PrintNumber {
    public void asc(int n) {
        if(n == 0) return;

        asc(n - 1);
        System.out.printf("%d ", n);
    }

    public void desc(int n) {
        if(n == 0) return;

        System.out.printf("%d ", n);
        desc(n - 1);
    }
}
```

desc(5)
- if(5 == 0) return; (리턴 X)
- printf("%d ", 5)
- desc(4)
  - if(4 == 0) return; (리턴 X)
  - printf("%d ", 4)
  - desc(3)
    - if(3 == 0) return; // (리턴 X)
    - printf("%d ", 3)
    - desc(2)
      - if(2 == 0) return; // (리턴 X)
      - printf("%d ", 2)
      - desc(1)
        - if(1 == 0) return; // (리턴 X)
        - printf("%d ", 1)
        - desc(0)
          - if(0 ==0) return;

재귀함수 잘 설계하는 방법

Base Case

- 계산 없이 바로 답을 구할 수 있는 경우
- 재귀 호출을 멈추고 함수가 종료되는 조건
- 적어도 하나 이상의 Base Case가 있어야 한다

Recursive Case

- 재귀 호출이 일어나는 경우
- 문제를 작은 부분으로 쪼개기 위함
- 함수가 호출될수록 부분 문제가 Base Case에 수렴해야 한다

```
class PrintNumber {
    public void asc(int n) {
        if(n == 0) return;
        asc(n - 1);
        System.out.printf("%d ", n);
    }
```

Base Case

Recursive Case

재귀함수 잘 설계하는 방법

귀납적으로 생각하기

- 큰 문제를 작은 부분문제로 분할
- 부분 문제들은 재귀를 통해 하위문제를 해결

중복 계산 방지 (memoization)

- 동일한 계산이 반복될 가능성이 있다면 메모리에 저장

재귀함수 예시 – BOJ2747: 피보나치 수

n번째 피보나치 수를 구하여라 (n<=45 자연수)

1	1	2	3	5	8	13	21

1	1	2	3	5	8	13	21

1	1	2	3	5	8	13	21

n번째 피보나치 수 ← (n-1)번째 피보나치 수 + (n-2)번째 피보나치 수

재귀함수 예시 – BOJ2747: 피보나치 수

Base Case

- 계산 없이 바로 답을 구할 수 있는 경우
- (n == 1) 혹은 (n == 2)

Recursive Case

- 재귀 호출이 일어나는 경우
- fibo(n) = fibo(n-1) + fibo(n-2)

재귀함수 예시 – BOJ2747: 피보나치 수

n번째 피보나치 수를 구하여라 (n<=45 자연수)

n번째 피보나치 수 ← (n-1)번째 피보나치 수 + (n-2)번째 피보나치 수

```
static int fibo(int n) {
    if (n == 1 || n == 2) return 1;
    return fibo(n - 1) + fibo(n - 2);
}
```

fibo(1) == ?, fibo(2) == ?
n 값이 자연수 미만으로 내려가지 않도록 2번째 값까지
Base Case로 처리한다

재귀함수 예시 – BOJ2747: 피보나치 수

n번째 피보나치 수를 구하여라 (n<=45 자연수)

n번째 피보나치 수 ← (n-1)번째 피보나치 수 + (n-2)번째 피보나치 수

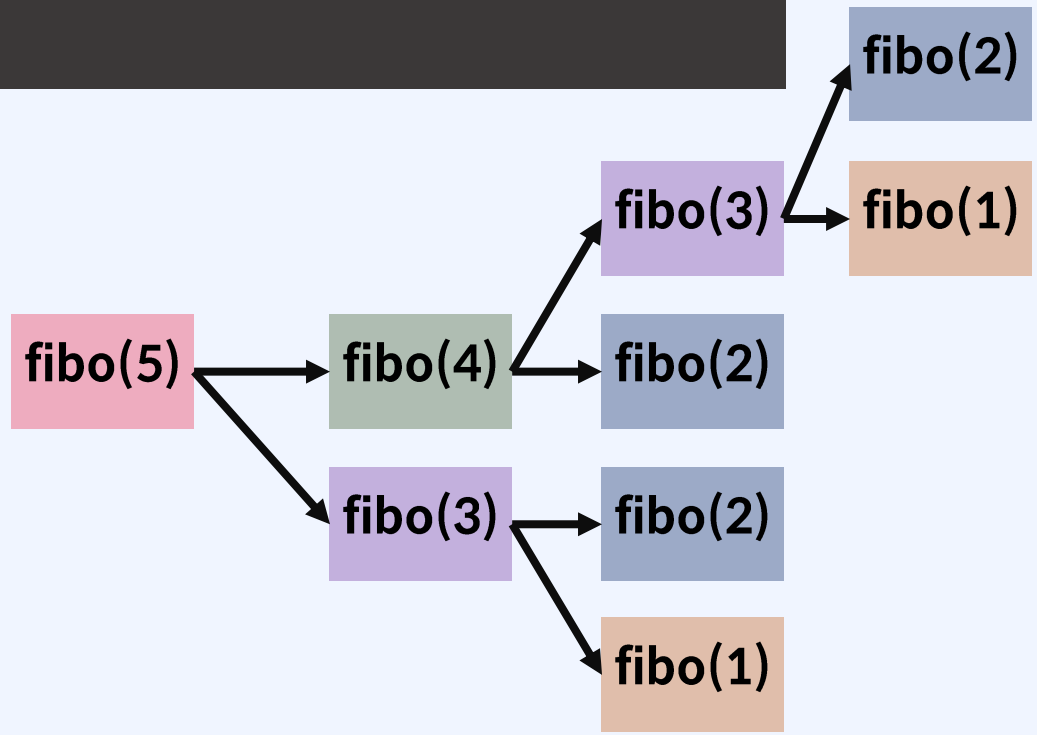
2747	시간 초과			Java 11 / 수정	381 B

```
    if (n == 1 || n == 2) return 1;
    return fibo(n - 1) + fibo(n - 2);
}
```

fibo(1) == ?, fibo(2) == ?
n 값이 자연수 미만으로 내려가지 않도록 2번째 값까지
Base Case로 처리한다

재귀함수 예시 – BOJ10870: 피보나치 수

```
static int fibo(int n) {
    if (n == 1 || n == 2) return 1;
    return fibo(n - 1) + fibo(n - 2);
}
```



매개변수별 함수 호출 횟수

- fibo(5): 1회
- fibo(4): 1회
- fibo(3): 2회
- fibo(2): 3회
- fibo(1): 2회

중복되는 연산이 많다!

재귀함수 예시 – BOJ2747: 피보나치 수

```
static int fibo(int n) {
    if (n == 1 || n == 2) return 1;
    return fibo(n - 1) + fibo(n - 2);
}
```

fibo(n)은 상태 값을 함수내에 보유하지 않는
수학적 의미의 함수이다
→ 호출 횟수 / 타이밍에 따라 값이 변하지 않는다
→ 한번 계산한 값을 저장해두면, 재활용할 수 있다

매개변수별
함수 호출 횟수

- fibo(5): 1회
- fibo(4): 1회
- fibo(3): 2회
- fibo(2): 3회
- fibo(1): 2회

```
static int[] cache = new int[50];
static int fibo(int n) {
    if (n == 1 || n == 2) return 1;
    if (cache[n] != 0) return cache[n];

    cache[n] = fibo(n - 1) + fibo(n - 2);
    return cache[n];
}
```

fibo(n)이 한번 계산되었다면
cache[n]에 값이 들어있다
→ 재귀 없이 바로 리턴

Ch03. 순열과 조합

2. [15654] N과 M (5)

순열 (Permutation)
: 집합 안에서 가능한 모든 조합을 나열하는 것

ex) {1, 2, 3}의 집합에서 만들 수 있는 순열

{1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2}, {3, 2, 1}

순열 (Permutation)

특징

- 중복이 없는 n개의 원소 집합에서는 n!개의 순열이 생성됨
- 중복된 원소가 있는 경우에는 각 원소의 {중복횟수!}의 곱으로 나눈 값의 개수만큼 순열이 생성됨
  - ex: {1, 1, 2, 2, 2, 3} 집합에서 만들어지는 순열 개수
    - n1 = 2! (1의 중복횟수)
      n2 = 3! (2의 중복 횟수)
      n3 = 1! (3의 중복 횟수)
    - n! / (n1! * n2! * n3!) = 720 / (2 * 6 * 1) = 60

BOJ15654: N과 M (5)

문제 요약

- N개의 자연수 집합에서 M개를 고른 수열
  (1 <= M <= N <= 8, 원소 <= 10,000)
- 수열은 사전 순으로 출력
- 중복되는 수열을 여러 번 출력하면 안된다
  (N개의 자연수는 모두 다른 수)

BOJ15654: N과 M (5)

문제 분석

- 최악의 경우 시간 복잡도?
  - N == 8 , M == 8
    (8개의 숫자 집합에서 8개의 숫자를 나열)
  - N개의 숫자는 모두 다르므로 8!

BOJ15654: N과 M (5)

사전 순 출력?

집합의 숫자를 미리 사전 순 으로 정렬하고
낮은 숫자(인덱스)부터 뽑아서 순열을 생성

ex) {4, 5, 2} 의 집합에서 2개의 숫자를 뽑는 경우
→ {2, 4, 5} 의 집합에서 앞의 숫자부터 뽑기
→ {2, 4}, {2, 5}, {4, 5}

BOJ15654: N과 M (5)

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int m = sc.nextInt();

    int[] numbers = new int[n];
    for (int i = 0; i < n; i++) {
        numbers[i] = sc.nextInt();
    }
    Arrays.sort(numbers);
    // 순열 생성 코드
}
```

입력받은 숫자들을 오름차순으로 정렬

BOJ15654: N과 M (5)

중복되지 않도록 처리하는 방법?

- N개의 자연수가 모두 다른 수 (문제의 조건)
- i 번째 수를 단 한번만 사용하면 된다
- 배열을 만들어서 사용 여부를 체크를 한다

ex) 5개의 원소중 0, 1, 3번째 원소로 수열을 만든 경우

check[]	[0]	[1]	[2]	[3]	[4]
	true	true	false	true	false

BOJ15654: N과 M (5)

수열을 생성하는 방법 [0, n)

- {i 번째 수}로 수열 생성이 가능한 경우
  - 더 이상 사용되지 않도록 체크 표시
  - 출력을 위해 저장
  - {i+1번째 수} 선택을 위해 재귀함수 호출
    - // … 재귀함수 호출
  - {i번째 수}가 다시 사용될 수 있도록 체크 제거

BOJ15654: N과 M (5)

재귀 종료를 위한 Base Case는?

n개의 수에서 m개를 뽑아 수열 생성

- m개를 모두 뽑았을 때
- 재귀함수가 (depth == m)으로 호출된 경우

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	1	2	3	4	5
perm(depth = 1) i = 1	1	2	3	4	5
perm(depth = 2) i = 2	1	2	3	4	5

perm(depth = 3) → depth == M (Base Case로 return) 1 2 3

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	**1**	2	3	4	5
perm(depth = 1) i = 1	**1**	**2**	3	4	5
perm(depth = 2) i = 3	**1**	**2**	3	**4**	5

perm(depth = 3) → depth == M (Base Case로 return) 1 2 4

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	1	2	3	4	5
perm(depth = 1) i = 1	1	2	3	4	5
perm(depth = 2) i = 4	1	2	3	4	5

perm(depth = 3) → depth == M (Base Case로 return) 1 2 5

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	**1**	2	3	4	5
perm(depth = 1) i = 2	**1**	2	**3**	4	5
perm(depth = 2) i = 1	**1**	**2**	**3**	4	5
perm(depth = 3)	→ depth == M (Base Case로 return)			1 3 2	

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	**1**	2	3	4	5
perm(depth = 1) i = 2	**1**	2	**3**	4	5
perm(depth = 2) i = 3	**1**	2	**3**	**4**	5

perm(depth = 3) → depth == M (Base Case로 return) 1 3 4

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	1	2	3	4	5
perm(depth = 1) i = 2	1	2	3	4	5
perm(depth = 2) i = 4	1	2	3	4	5

perm(depth = 3) → depth == M (Base Case로 return) 1 3 5

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	1	2	3	4	5
perm(depth = 1) i = 3	1	2	3	4	5
perm(depth = 2) i = 1	1	2	3	4	5
perm(depth = 3)	→ depth == M (Base Case로 return)			1 4 2	

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	**1**	2	3	4	5
perm(depth = 1) i = 3	**1**	2	3	**4**	5
perm(depth = 2) i = 2	**1**	2	**3**	**4**	5

perm(depth = 3) → depth == M (Base Case로 return) 1 4 3

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	1	2	3	4	5
perm(depth = 1) i = 3	1	2	3	4	5
perm(depth = 2) i = 4	1	2	3	4	5

perm(depth = 3) → depth == M (Base Case로 return) 1 4 5

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	1	2	3	4	5
perm(depth = 1) i = 4	1	2	3	4	5
perm(depth = 2) i = 1	1	2	3	4	5
perm(depth = 3)	→ depth == M (Base Case로 return)			1 5 2	

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	1	2	3	4	5
perm(depth = 1) i = 4	1	2	3	4	5
perm(depth = 2) i = 2	1	2	3	4	5

perm(depth = 3) → depth == M (Base Case로 return) 1 5 3

BOJ15654: N과 M (5)

수열 출력 순서: (1) (2) (3)

ex) N(5)개의 수 중에서 M(3)개를 뽑는 경우

perm(depth = 0) i = 0	1	2	3	4	5
perm(depth = 1) i = 4	1	2	3	4	5
perm(depth = 2) i = 3	1	2	3	4	5

perm(depth = 3) → depth == M (Base Case로 return) 1 5 4

BOJ15654: N과 M (5)

수열을 생성하는 방법 [0, n)

```
for (int i = 0; i < n; i++) {
    if (!check[i]) {
        check[i] = true;
        output[depth] = numbers[i];
        perm(depth + 1, n, m);
        check[i] = false;
    }
}
```

BOJ15654: N과 M (5)

재귀 종료를 위한 Base Case는?

```
if (depth == m) {
    for(int i = 0; i < m; i++) {
        System.out.print(output[i] + " ");
    }
    System.out.println();
    return;
}
```

Ch03. 순열과 조합

3. [15655] N과 M (6)

BOJ15655: N과 M (6)

문제 요약

- N개의 자연수 집합에서 M개를 고른 수열
  (1 <= M <= N <= 8, 원소 <= 10,000)
- 수열은 사전 순으로 출력
- 중복되는 수열을 여러 번 출력하면 안된다
  (N개의 자연수는 모두 다른 수)
- 고른 수열은 오름차순이어야 한다 ← NEW

BOJ15654: N과 M (5)

문제 분석

직전 문제의 조건에서
수열 내의 수가 오름차순이여야 하는 조건이 추가되었다

ex)
{1, 4, 5} ← 조건 만족
{1, 5, 4} ← 조건 불만족 (직전 문제에서는 만족)

어떻게 접근해야 할까?

BOJ15655: N과 M (6)

아이디어 1. 출력 직전에 오름차순을 검사하자

{수열의 모든 경우의 수} * {오름차순 검사 비용}

- 모든 경우의 수는 O(n!)이다 (n==8)
- arr[i-1] < arr[i] 모든 원소에서 만족하는지 체크 O(n)

O(n*n!) (n==8인 경우 322,560)

시간안에 정답은 나오지만
n이 늘어날수록 경우의 수가 크게 증가한다

BOJ15655: N과 M (6)

아이디어 2.
재귀를 오름차순으로만 진입할 수 있게 조건을 변경한다

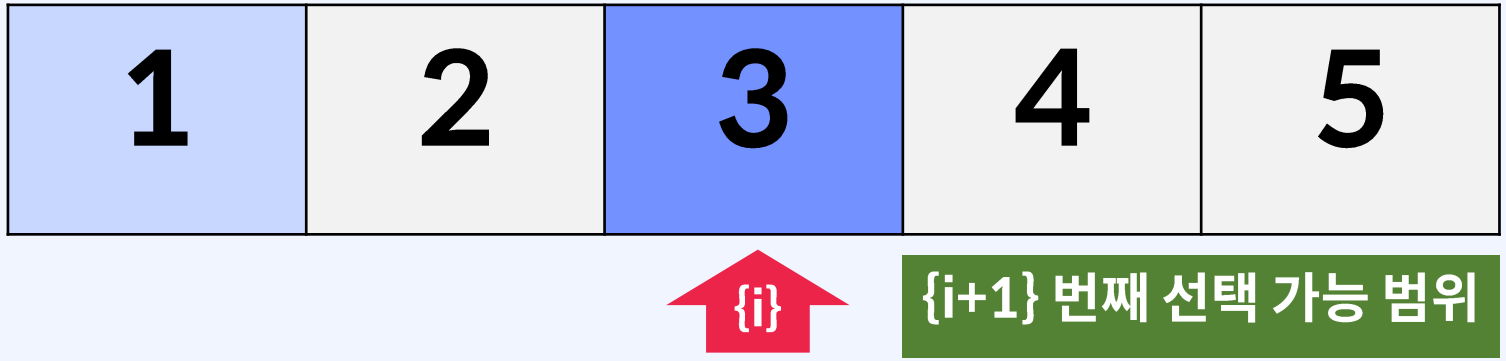
직전의 문제와 동일하게 사전 순 정렬을 위해
입력 받은 수가 정렬된 상태이다
(arr[i] < arr[i+1] 가 보장됨)

따라서
{d} 번째 재귀에서 배열의 {i}번째 수를 사용했다면
{d+1} 번째 재귀에서는 {i+1} 번째 수부터 사용하면 된다

BOJ15655: N과 M (6)

{d} 번째 재귀에서 배열의 {i}번째 수를 사용했다면

{d+1} 번째 재귀에서는 {i+1} 번째 수부터 사용하면 된다



BOJ15655: N과 M (6)

{d} 번째 재귀에서 배열의 {i}번째 수를 사용했다면
{d+1} 번째 재귀에서는 {i+1} 번째 수부터 사용하면 된다

```
public static void perm(int depth, int n, int m, int start) {
    // base case return
    for (int i = start; i < n; i++) {
        if (!check[i]) {
            check[i] = true;
            output[depth] = arr[i];
            perm(depth + 1, n, m, i + 1);
            check[i] = false;
        }
    }
}
```

BOJ15655: N과 M (6)

시간 복잡도는?

기존의 n! 경우의 수에서
{1, 4, 5} 는 조건을 만족하지만
{1, 5, 4}, {5, 1, 4} 등은 만족하지 않는다

모든 수는 문제의 조건에 의해 서로 다르기 때문에
m개의 수를 뽑은 경우에서 만들어지는 경우의 수가
m! → 1 로 바뀐다

따라서 경우의 수는 n! / m! 로 좁혀진다

Ch03. 순열과 조합

4. [15656] N과 M (7)

BOJ15656: N과 M (7)

요약

- N개의 자연수 집합에서 M개를 고른 수열
  (1 <= M <= N <= 7, 원소 <= 10,000)
- 수열은 사전 순으로 출력
- 중복되는 수열을 여러 번 출력하면 안된다
  (N개의 자연수는 모두 다른 수)
- 같은 수를 여러 번 골라도 된다 ← NEW

BOJ15656: N과 M (7)

같은 수를 여러 번 골라도 된다 ← NEW

```
for (int i = 0; i < n; i++) {
    if (!check[i]) {
        check[i] = true;
        output[depth] = arr[i];
        perm(depth + 1);
        check[i] = false;
    }
}
```

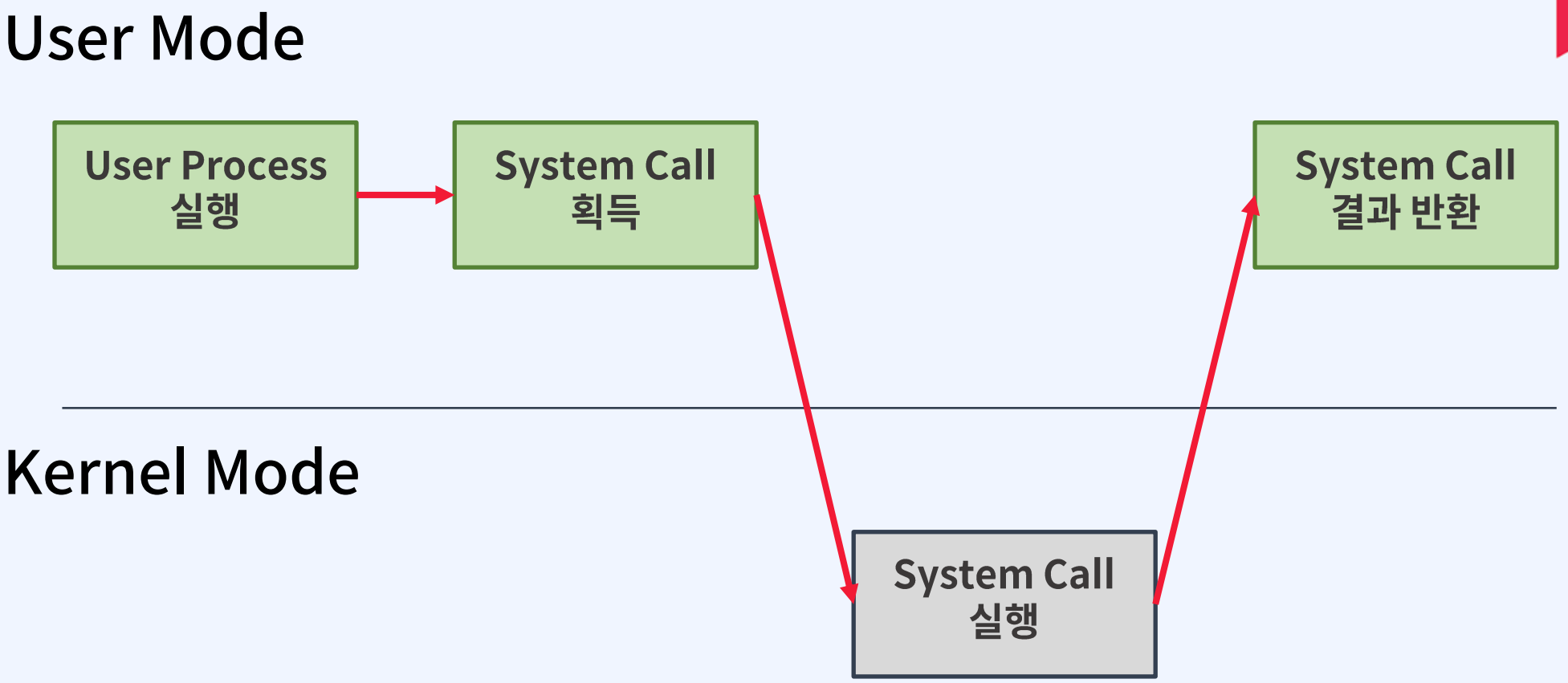
같은 수를 한번만 고르기 위해 check 배열을 사용했었다

BOJ15656: N과 M (7)

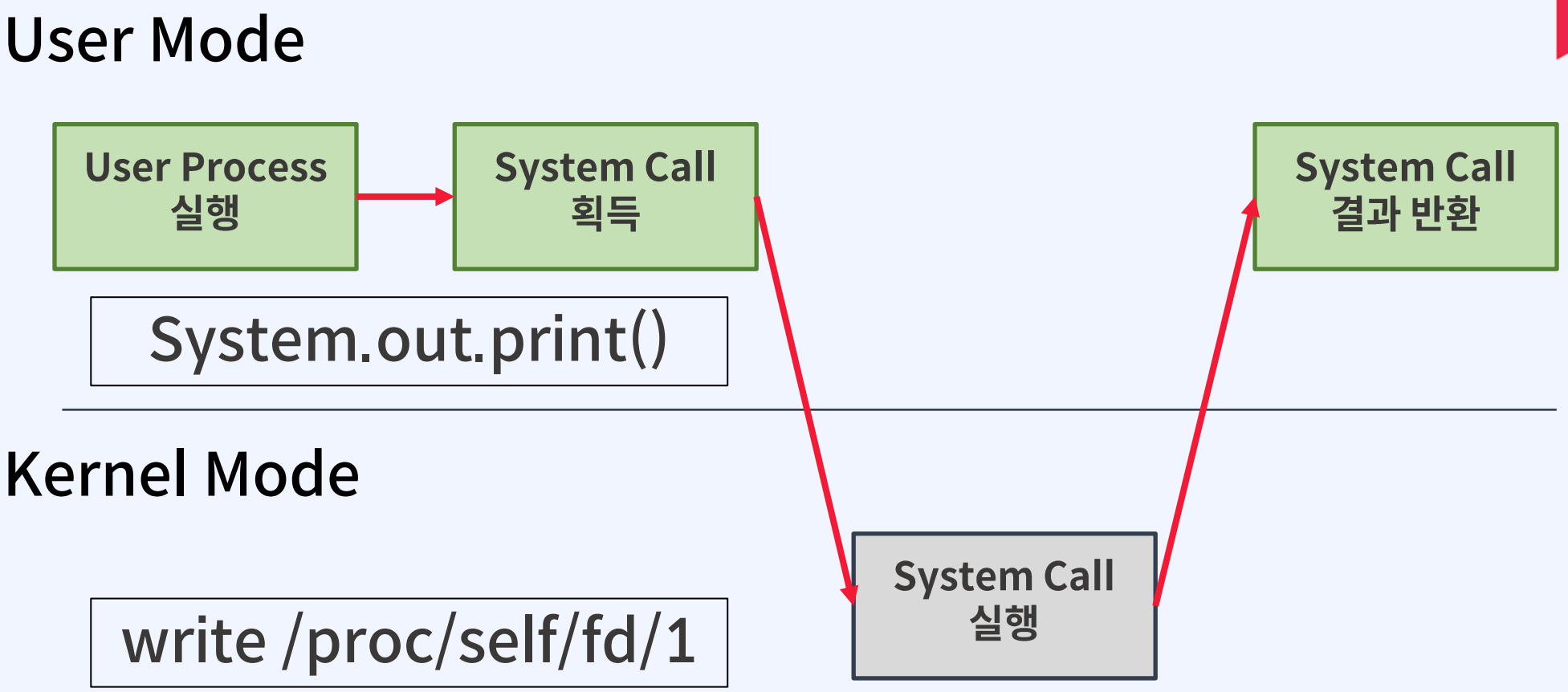
15656	시간 초과			Java 15 / 수정

출력하는 방식에 따라 시간초과가 발생할 수 있다

BOJ15656: N과 M (7)



BOJ15656: N과 M (7)



stdout 은 fd: 1에 write하는 시스템콜을 호출한다

BOJ15656: N과 M (7)

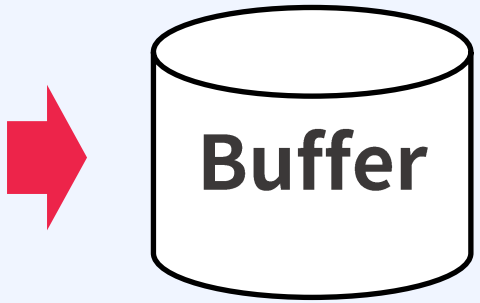
System.out.print()

System.out.print()

System.out.print()

System.out.print()

System.out.print()



write /proc/self/fd/1

대부분의 언어는 여러 출력을 버퍼에 담고 한번에 write를 시도한다

BOJ15656: N과 M (7)

한번에 write 시도 조건

1. Buffer의 용량이 가득 찼을 때
   - write를 실행하고 버퍼를 비운다

2. 명시적으로 flush 요청이 발생했을 때
   - System.out.flush()

BOJ15656: N과 M (7)

그런데?

PrintStream.java

```
    private void newLine() {
        try {
            synchronized (this) {
                ensureOpen();
                textOut.newLine();
                textOut.flushBuffer();
                charOut.flushBuffer();
                if (autoFlush)
                    out.flush();
            }
        }
```

System.out.println()에서
호출되는newLine()은

flushBuffer() 를 호출한다!!!

→ n^n 회 시스템 콜을 호출한다

BOJ15656: N과 M (7)

```
public static void print(int[] arr, int n) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < n; i++) {
        sb.append(arr[i]).append(" ");
    }
    System.out.println(sb);
}
```

println이 호출될 때마다
buffer가 flush된다
→ 출력을 한번에 모아서 하도록 수정

```
public static StringBuilder sb = new StringBuilder();
public static void print(int[] arr, int n) {
    for (int i = 0; i < n; i++) {
        sb.append(arr[i]).append(" ");
    }
    sb.append("\n");
}
```

```
public static BufferedWriter bw
  = new BufferedWriter(new OutputStreamWriter(System.out));
public static void print(int[] arr, int n) throws IOException {
    for (int i = 0; i < n; i++) {
        bw.write(arr[i] + " ");
    }
    bw.write("\n");
}
```

Ch03. 순열과 조합

5. [15657] N과 M (8)

BOJ15657: N과 M (8)

요약

- N개의 자연수 집합에서 M개를 고른 수열
  (1 <= M <= N <= 7, 원소 <= 10,000)
- 수열은 사전 순으로 출력
- 중복되는 수열을 여러 번 출력하면 안된다
  (N개의 자연수는 모두 다른 수)
- 같은 수를 여러 번 골라도 된다
- 고른 수열은 비내림차순을 만족해야 한다 ← NEW

BOJ15657: N과 M (8)

분석

BOJ15656 (7): 같은 수를 여러 번 골라도 된다
BOJ15655 (6): 고른 수열은 오름차순을 만족해야 한다

→ 두 문제에서 적용했던 조건을 응용하면 된다

BOJ15657: N과 M (8)

아이디어 2. 재귀를 오름차순으로만 진입할 수 있게 조건을 변경한다

(depth = n) 번째 재귀에서 배열의 {i}번째 수를 사용했다면
(depth = n+1) 재귀에서는 {i+1} 번째 수부터 사용하면 된다

```
public static void perm(int depth, int start) {
    // base case return
    for (int i = start; i < n; i++) {
        if (!check[i]) {
            check[i] = true;
            output[depth] = arr[i];
            perm(depth + 1, i);
            check[i] = false;
        }
    }
}
```