# 오늘 학습한 문제들에 대한 종합 정리

---

## 백준 11723 번 문제: 비트마스크를 활용한 집합 관리

### 문제 요약

- 다양한 명령(`add`, `remove`, `check`, `toggle`, `all`, `empty`)으로 집합을 관리.
- 숫자 1~20의 집합을 비트마스크 방식으로 효율적으로 처리.

### 필요한 알고리즘/개념

1. **비트마스크**
    - `BooleanArray(21)`를 사용하여 특정 숫자가 집합에 포함되는지 관리.
    - 명령에 따라 특정 비트를 설정하거나 해제.
2. **StringBuilder 활용**
    - 출력 최적화: `println` 대신 `StringBuilder`로 결과를 한 번에 출력.

### 핵심 코드

#### 명령 처리 (효율적)

```
when (mode) {
    "add" -> result[x] = true
    "remove" -> result[x] = false
    "check" -> output.append(if (result[x]) "1\n" else "0\n")
    "toggle" -> result[x] = !result[x]
    "all" -> result.fill(true)
    "empty" -> result.fill(false)
}

val output = StringBuilder()
println(output)
```

#### 꼭 외워야 할 부분

1. **fill 메서드**
    - 배열의 값을 한 번에 초기화
    - ex: `result.fill(true)`

## 백준 10431번 문제: 키 순서대로 줄세우기

### 문제 요약

- 학생들이 키 순서대로 줄을 서는 과정을 시뮬레이션
- 삽입 위치를 찾아 정렬하며, 이동 횟수(뒤로밀림)를 계산

## 필요한 알고리즘/개념

1. **삽입 정렬(Insertion Sort)**
   - 리스트의 특정 위치를 찾아 삽입하며, 뒤로 밀리는 학생 수를 계산
   - 시간 복잡도: O(n^2) (단, n = 20으로 제한되어 충분히 처리 가능).
2. **리스트의 특정 구간 조작**
   - 삽입 위치를 찾고 뒤로 밀리는 수를 계산: `line.size - position`.

## 핵심 코드

### 삽입 위칮 탐색

```
var position = line.size
for (i in line.indices) {
    if (line[i] > height) {
        position = i
        break
    }
}
```

### 뒤로 밀림 계산

```
moves += line.size - position
line.add(position, height)
```

### 전체 코드

```
import java.io.*

fun main() = with(File("10431_input.txt").bufferedReader()) {
    val N = readLine().toInt()
    val results = mutableListOf<String>()

    repeat(N) {
        val input = readLine().split(" ").map { it.toInt() }
        val caseNumber = input[0]
        val heights = input.subList(1, input.size) // 첫 번째 값을 제외한 배열

        var moves = 0
        val line = mutableListOf<Int>()

        // 줄서기 과정 (삽입 정렬 방식)
        for (height in heights) {
            var position = line.size
            for (i in line.indices) {
                if (line[i] > height) {
```

```
                    position = i
                    break
                }
            }

            // 밀려난 학생 수 = 뒤에 있는 학생 수
            moves += line.size - position
            line.add(position, height)
        }
        results.add("$caseNumber $moves")
    }

    println(results.joinToString("\n"))
}
```

#### 꼭 외워야 할 부분

1. `add(index, element`
    - 리스트의 특정 위치에 요소 삽입.
    - ex: `line.add(position, height)`.
2. **삽입 정렬의 원리**
    - 정렬된 부분에 새 데이터를 삽입 -> 이동 횟수 계산

## 백준 8979번 문제: 국가 순위 정하기

### 문제 요약

- 국가 간 금, 은, 동메달 순위를 계산
- 국가 K의 등수를 출력.

### 필요한 알고리즘/개념

1. **정렬 기준 설정**
    - 금메달 -> 은메달 -> 동메달 순으로 내림차순 정렬.
    - 같은 메달 수를 가진 국가끼리는 공동 등수.
2. **K 국가의 등수 찾기**
    - 정렬된 리스트에서 K의 인덱스를 찾아 출력.

### 핵심 코드

#### 정렬 기준 설정

```
countries.sortWith(compareBy<Triple<Int, Int, Int>> { -it.first } // 금메달
    .thenByDescending { it.second } // 은메달
    .thenByDescending { it.third }) // 동메달
```

#### K 국가의 등수 찾기

```
for (i in countries.indices) {
    if (countries[i] == triple) {
        println(i + 1)
        break
    }
}
```

**전체 코드**

```
import java.io.*

fun main() = with(File("8979_baek.txt").bufferedReader()) {
    val (N, K) = readLine().split(" ").map { it.toInt() }
    var triple = Triple(0, 0, 0)
    val countries = mutableListOf<Triple<Int, Int, Int>>()

    repeat(N) {
        val (id, gold, silver, bronze) = readLine().split(" ").map {
it.toInt() }
        countries.add(Triple(gold, silver, bronze))
        if (id == K) triple = Triple(gold, silver, bronze)
    }

    countries.sortWith(compareBy<Triple<Int, Int, Int>> {
        -it.first
    }.thenByDescending {
        it.second
    }.thenByDescending {
        it.third
    })

    for (i in countries.indices) {
        if (countries[i] == triple) {
            println(i + 1)
            break
        }
    }
}
```

**꼭 외워야 할 부분**

1. `sortWith(compareBy{})`
    - 다중 정렬 기준 설정.
    - ex: `compareByDescending`을 통해 내림차순 정렬.
2. `Triple`**의 활용**
    - 금, 은, 동메달 데이터를 저장하고 비교.
    - ex) `Triple(gold, silver, bronze)`.