

| 시간복잡도, 브루트포스

|

광운대학교
소프트웨어학부

김석희

용문고등학교에 등교를 하려합니다. 방법이 두가지가 있네요.

안암역을 통해 가는 방법과 성신여대에서 04번 버스를 타고 가는 방법이 있고 택시를 타는 방법이 있습니다.

안암역을 통해 가면 45분이 걸리고 성신여대 04번 버스를 타면 25분이 걸립니다.

택시를 타면 10분이 걸리네요. 저는 늦어가지고 택시를 택했고, 이 과정을 알고리즘으로 표현하면 다음과 같습니다.



```
function TakeTaxy(from, to) {
```

1. 차량에 탑승한다.
 2. from에서 to까지 최단거리 루트를 선택한다.
 3. 목적지까지 가는 루트를 설명한다.
 4. 등교 시간의 경우 차가 막히는 루트는 피한다.
 5. 출발하는동안
 6. 만약 적색 신호등이면 정지한다.
 7. 만약 녹색 신호등이면 출발한다.
 8. 도착하면
 9. 요금을 정산한다.
 10. 차량에서 내린다.
- ```
}
```

## 따라서 알고리즘이란

- 어떤 목적을 달성하거나 결과물을 만들어내기 위해 거쳐야 하는 일련의 과정들을 의미한다.
- 가는 루트는 다양하며 여러가지 상황에 따른 알고리즘은 모두 다르다. 따라서 시간 복잡도가 가장 낮은 알고리즘을 선택하여 사용한다.

알고리즘의 실행시간을 두 부분으로 나누면

1. 입력값의 크기에 따라 알고리즘의 실행시간을 검증해볼 수 있다.
2. 입력값의 크기에 따른 함수의 증가량, 우리는 이것을 성장률이라고 부른다.  
이때 중요하지 않는 상수와 계수들을 제거하면 알고리즘의 실행시간에서 중요한  
|성장률에 집중할 수있는데 이것을 점근적 표기법(Asymptotic notation)이라 부른다.  
여기서, 점근적이라는 의미는 가장 큰 영향을 주는 항만 계산한다는 의미다.

점근적 표기법은 다음 세가지가 있는데 시간복잡도를 나타내는데 사용된다.

- 최상의 경우 : 오메가 표기법 (Big- $\Omega$  Notation)
- 평균의 경우 : 세타 표기법 (Big- $\theta$  Notation)
- 최악의 경우 : 빅오 표기법 (Big-O Notation)

평균인 세타 표기를 하면 가장 정확하고 좋겠지만 평가하기가 까다롭다.

그래서 최악의 경우인 빅오를 사용하는데 알고리즘이 최악일때의 경우를 판단하면 평균과 가까운 성능으로 예측하기 쉽기 때문이다.

# 얼마나 걸릴까?

👉 가장 자주 사용되는 표기법은?

- 빅오 표기법은 최악의 경우를 고려하므로, 프로그램이 실행되는 과정에서 소요되는 최악의 시간까지 고려할 수 있기 때문이다.
- “최소한 특정 시간 이상이 걸린다” 혹은 “이 정도 시간이 걸린다”를 고려하는 것보다 “이 정도 시간까지 걸릴 수 있다”를 고려해야 그에 맞는 대응이 가능하다.

아무리 오래걸려도  $O(?)$  만큼만 걸린다.

시간복잡도에서 중요하게 보는것은 가장큰 영향을 미치는  $n$ 의 단위이다.

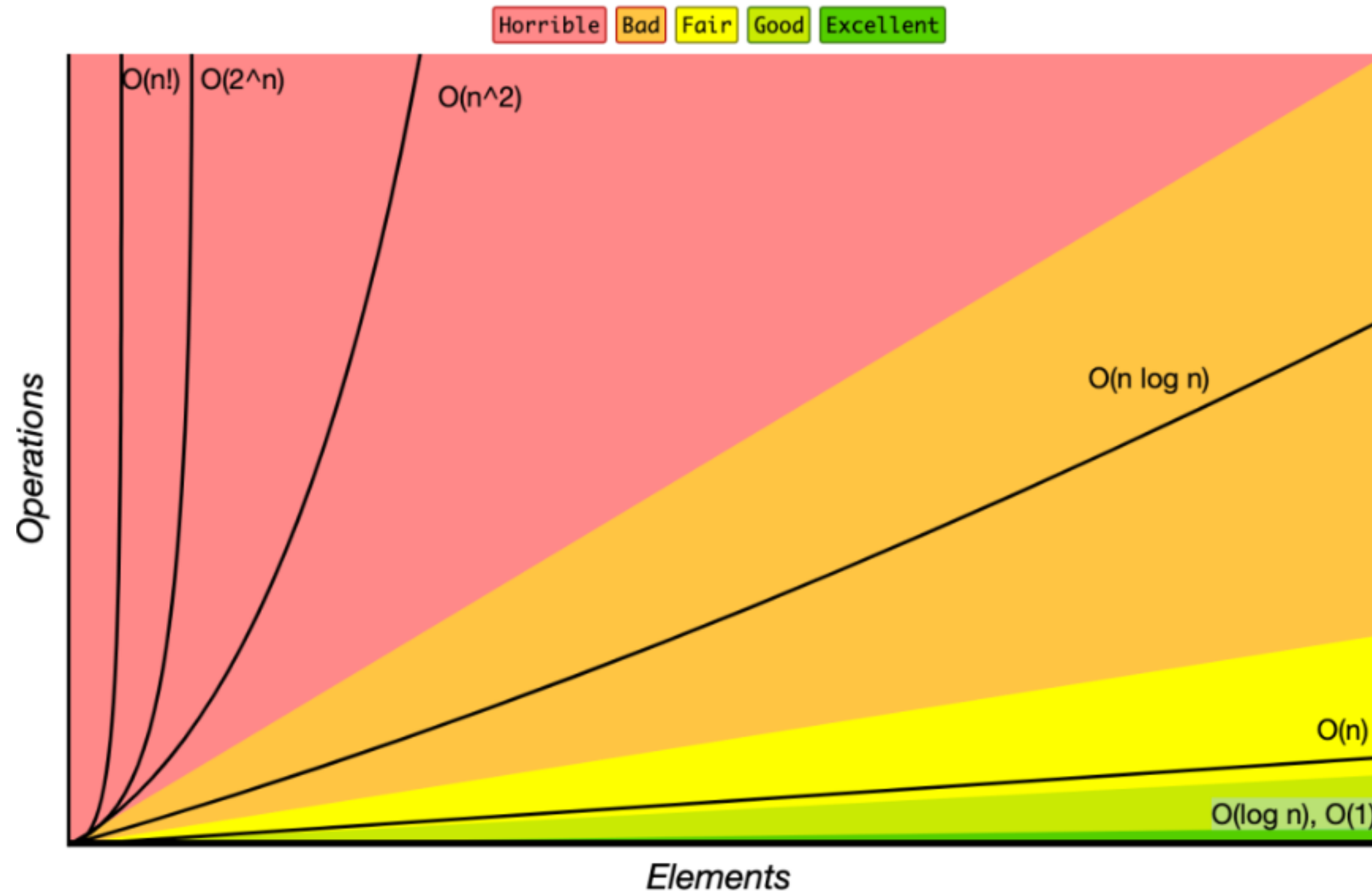
|           |          |                          |
|-----------|----------|--------------------------|
| 1         | $O(1)$   | --> 상수                   |
| $2n + 20$ | $O(n)$   | --> $n$ 이 가장 큰영향을 미친다.   |
| $3n^2$    | $O(n^2)$ | --> $n^2$ 이 가장 큰영향을 미친다. |

시간복잡도의 문제해결 단계를 나열 하면 아래와같다.

- $O(1)$  - 상수 시간 : 문제를 해결하는데 오직 한 단계만 처리함.
- $O(\log n)$  - 로그 시간 : 문제를 해결하는데 필요한 단계들이 연산마다 특정 요인에 의해 줄어듬.
- $O(n)$  - 직선적 시간 : 문제를 해결하기 위한 단계의 수와 입력값  $n$ 이 1:1 관계를 가짐.
- $O(n \log n)$  : 문제를 해결하기 위한 단계의 수가  $N * (\log_2 N)$  번만큼의 수행시간을 가진다.  
(선형로그형)
- $O(n^2)$  - 2차 시간 : 문제를 해결하기 위한 단계의 수는 입력값  $n$ 의 제곱.
- $O(C^n)$  - 지수 시간 : 문제를 해결하기 위한 단계의 수는 주어진 상수값  $C$  의  $n$  제곱.



## Big-O Complexity Chart



| Complexity    | 1 | 10      | 100                             |
|---------------|---|---------|---------------------------------|
| $O(1)$        | 1 | 1       | 1                               |
| $O(\log N)$   | 0 | 2       | 5                               |
| $O(N)$        | 1 | 10      | 100                             |
| $O(N \log N)$ | 0 | 20      | 461                             |
| $O(N^2)$      | 1 | 100     | 10000                           |
| $O(2^N)$      | 1 | 1024    | 1267650600228229401496703205376 |
| $O(N!)$       | 1 | 3628800 | 화면에 표현할 수 없음!                   |

# $O(1)$

## $O(1)$ : 상수

아래 예제 처럼 입력에 관계없이 복잡도는 동일하게 유지된다.

```
void hello(){
 cout << "hello";
}
```

# $O(\log n)$ $O(n \log n)$

## $O(\log n)$ $O(n \log n)$

주로 입력 크기에 따라 처리 시간이 증가하는 정렬알고리즘에서 많이 사용된다.  
다음은 이진검색의 예이다.

```
int binarySearch(int start, int end, int target) {
 if (end - start <= 1) {
 if (target == arr[start])
 return start;
 return -1;
 }
 int result = -1;
 int mid = (int)((start + end) / 2);
 if (arr[mid] == target)
 result = mid;
 else if (arr[mid] > target)
 result = binarySearch(start, mid, target);
 else if (arr[mid] < target)
 result = binarySearch(mid + 1, end, target);
 return result;
}
```

# $O(\log n)$ $O(n \log n)$

$O(\log n)$ 은 로그 복잡도(logarithmic complexity)라고 부르며, Big-O표기법중  $O(1)$  다음으로 빠른 시간 복잡도를 가진다.

- 자료구조에서 배웠던 BST(Binary Search Tree)를 기억하는가?
- BST에선 원하는 값을 탐색할 때, 노드를 이동할 때마다 경우의 수가 절반으로 줄어든다.
- 이해하기 쉬운 게임으로 비유해 보자면 up & down을 예로 들 수 있다.
  1. 1~100 중 하나의 숫자를 플레이어1이 고른다. (30을 골랐다고 가정한다.)
  2. 50(가운데) 숫자를 제시하면 50보다 작으므로 down을 외친다.
  3. 1~50중의 하나의 숫자이므로 또다시 경우의 수를 절반으로 줄이기 위해 25를 제시한다.
  4. 25보다 크므로 up을 외친다.
  5. 경우의 수를 계속 절반으로 줄여나가며 정답을 찾는다.
- 매번 숫자를 제시할 때마다 경우의 수가 절반이 줄어들기 때문에 최악의 경우에도 7번이면 원하는 숫자를 찾아낼 수 있게 된다.
- **BST의 값 탐색** 또한 이와같은 로직으로,  **$O(\log n)$ 의 시간 복잡도를 가진 알고리즘(탐색기법)**이다.

# $O(\log n)$ $O(n \log n)$

실행횟수 :  $x$       입력 값 :  $N$

$$2^x = N$$

$$x = \log N$$

$$O(x) = O(\log N)$$

2번씩  $x$ 번 반복하면  
 $N$ 개의 데이터에  
접근할 수 있다고  
할때.

# $O(N)$

## $O(N)$ : 선형

입력이 증가하면 처리 시간또는 메모리 사용이 선형적으로 증가한다.

```
void N(int a) {
 for (int i = 0; i < a; i++) {
 cout << i;
 }
}
```

# $O(N^2)$

## $O(N^2)$ : Square

반복문이 두 번 있는 케이스

```
void N(int a, int b) {
 for (int i = 0; i < a; i++) {
 for (int j = 0; j < b; j++) {
 cout << i * j;
 }
 }
}
```



# 시간복잡도를 구하는 요령

각 문제의 시간복잡도 유형을 빨리 파악할 수 있도록 아래 예를 통해 빠르게 알아 볼수 있다.

- 하나의 루프를 사용하여 단일 요소 집합을 반복 하는 경우 :  $O(n)$
- 컬렉션의 절반 이상 을 반복 하는 경우 :  $O(n / 2) \rightarrow O(n)$
- 두 개의 다른 루프를 사용하여 두 개의 개별 컬렉션을 반복 할 경우 :  $O(n + m) \rightarrow O(n)$
- 두 개의 중첩 루프를 사용하여 단일 컬렉션을 반복하는 경우 :  $O(n^2)$
- 두 개의 중첩 루프를 사용하여 두 개의 다른 컬렉션을 반복 할 경우 :  $O(n * m) \rightarrow O(n^2)$
- 컬렉션 정렬을 사용하는 경우 :  $O(n \cdot \log(n))$

# 브루트포스

## 1. 완전 탐색이란?

컴퓨터의 빠른 계산 능력을 이용하여 가능한 경우의 수를 일일이 나열하면서 답을 찾는 방법을 의미한다.  
'무식하게 푼다'라는 의미인 **Brute-Force (브루트 포스)**라고도 부른다.

이름이 거창하게 지어져 있지만 사실 완전 탐색 자체로는 알고리즘이라고 부르긴 그렇고, 문제 푸는 '방법'이라고 이해하면 편할 것 같다. 알고리즘을 모르는 사람이 프로그래밍 문제를 푼다면, 당연히 가능한 모든 경우들을 다 구해서 그중에 만족하는 답을 찾아낼 것이고 이 과정 자체가 완전 탐색이다. 대신에 이 방식은 절대 답을 못 구하는 경우는 없으므로 나름 강력한 방법이다.

예를 한번 들어보자.

**"10개의 정수 원소로 이루어진 수열이 있다. 이 수열에서 두 원소를 선택해서 구한 합의 최댓값을 구하시오."**

이 문제에서 완전 탐색을 이용하는 방식은, 10개의 원소에서 두 원소를 고르는 모든 경우를 다 고려하는 것이다.  
즉,  $10C2$  가지의 경우를 모두 구한 후 나온 합 중 최댓값을 출력하게 되는 것이다.

이 예시에서는  $10C2 = 45$ 이므로 경우가 45가지밖에 나오지 않아 충분히 빠르게 해결이 가능하다.  
하지만 만약 원소의 개수가 10만이라면 경우의 수가 50억 지나므로 제한된 시간 내에 경우들을 모두 구할 수 없게 된다.  
(이를 해결하기 위해서는 가장 큰 원소와 두 번째로 큰 원소를 찾아 더하면 답이 된다.)

이처럼 완전 탐색은 답으로 가능한 경우의 수가 많은 경우에는 이용하기가 어려우므로 완전 탐색을 이용할 수 있는 경우인지 잘 파악하는 게 중요하다.

완전 탐색 자체가 알고리즘은 아니기 때문에 완전 탐색 방법을 이용하기 위해서 여러 알고리즘 기법이 이용된다. 주로 이용되는 기법들은 다음과 같다.

- 단순 *Brute-Force*
- 비트마스크(*Bitmask*)
- 재귀 함수
- 순열 (*Permutation*)
- *BFS / DFS*

## 1. 단순 Brute-Force

어느 기법을 사용하지 않고 단순히 for문과 if문 등으로 모든 case들을 만들어 답을 구하는 방법이다. 이는 아주 기초적인 문제에서 주로 이용되거나, 전체 풀이의 일부분으로 이용하며, 따라서 당연히 대회나 코테에서는 이 방법만을 이용한 문제는 거의 나오지 않는다.

# 완전 탐색 실전 이용 예시

보통 완전 탐색 문제는 난이도가 쉬운 편이기 때문에 초급자들도 큰 어려움을 겪지 않는다. 하지만 약간의 난이도가 있는 문제에서 오히려 알고리즘 공부를 어느 정도 한 사람들이 "이게 완전 탐색이라고?" 하는 반응을 보이는 경우가 꽤나 있다. 완전 탐색 유형이라고 미처 생각하지 못하는 경우가 많기 때문이다. 특히나 코드포스 같은 아이디어를 많이 요구하는 CP에서 자주 발생한다. 그래서 필자의 경험 상, 문제를 봤을 때 완전 탐색이지 않을까?라는 생각을 한번 정도는 해볼 만한 경우들을 생각해봤다.

(지금까지의 적은 경험 하에서는 코드포스 등의 CP 이외에는 크게 유용할 것 같지는 않다... 그래도 혹시 모르니..)


## 1. 입력으로 주어지는 데이터(N)의 크기가 매우 작다.

보통 프로그래밍 문제들을 풀게 되면  $N = 10$ 만,  $20$ 만 같은 크기를 주는 경우가 많다. 하지만 대부분의 경우 완전 탐색 문제는  $N$ 의 크기가 매우 작다. 위에서 언급한 부분집합 문제나, 순열, 조합 같은 문제들은 완전 탐색으로 푼다면 기본적으로 시간 복잡도가  $O(2^N)$ 이나,  $O(N!)$  이므로 당연히  $N$ 의 크기가 매우 작아야 한다.

## 2. 답의 범위가 작고, 임의의 답을 하나 선택했을 때 문제 조건을 만족하는지 역추적할 수 있다.

1번과 연관되는 내용일 수도 있다.  $N$ 의 크기가 작으면 답의 범위 또한 작을 확률이 높기 때문이다. 하지만, 2번의 경우는  $N$ 이 작다고 하더라도 완전 탐색인 것을 떠올리기가 어려운 경우가 많다.

정리하자면 브루트포스(완전탐색)는 모든 경우의 수를 탐색하는 것.


[HOME](#)
[ABOUT](#)
[CONTACT](#)
[SUSTAINABILITY](#)
[CAREERS](#)
[SPECIAL EVENTS](#)
[SPECIAL REPORTS](#)

성공



#### 4 브론즈 IV

| 시간 제한 | 메모리 제한  | 제출   | 정답   | 맞힌 사람 | 정답 비율   |
|-------|---------|------|------|-------|---------|
| 1 초   | 1024 MB | 2390 | 1867 | 1635  | 79.834% |

세 양의 정수  $a, b, c$ 가 주어질 때, 다음 조건을 만족하는 정수 쌍  $(x, y, z)$ 의 개수를 구하시오.

- $1 \leq x \leq a$
- $1 \leq y \leq b$
- $1 \leq z \leq c$
- $(x \bmod y) = (y \bmod z) = (z \bmod x)$

$(A \bmod B)$ 는  $A$ 를  $B$ 로 나눈 나머지를 의미한다.

첫째 줄에 테스트 케이스의 수  $T$ 가 주어진다. ( $1 \leq T \leq 100$ )

다음  $T$ 개의 각 줄에는 세 정수  $a, b, c$ 가 공백으로 구분되어 주어진다. ( $1 \leq a, b, c \leq 60$ )

한 줄에 하나씩 정답을 출력한다.

세 양의 정수  $a, b, c$ 가 주어질 때, 다음 조건을 만족하는 정수 쌍  $(x, y, z)$ 의 개수를 구하시오.

- $1 \leq x \leq a$
- $1 \leq y \leq b$
- $1 \leq z \leq c$
- $(x \bmod y) = (y \bmod z) = (z \bmod x)$

$(A \bmod B)$ 는  $A$ 를  $B$ 로 나눈 나머지를 의미한다.

```
void solve() {
 for (int i = 1; i <= x; i++) {
 for (int j = 1; j <= y; j++) {
 for (int k = 1; k <= z; k++) {
 if ((i % j == j % k) && (j % k == k % i))
 answer++;
 }
 }
 }
 cout << answer << "\n";
}
```



기초 : [boj.kr/25494](http://boj.kr/25494) 단순한 문제 (Small)

[boj.kr/2798](http://boj.kr/2798) 블랙잭  
[boj.kr/13410](http://boj.kr/13410) 거꾸로 구구단  
[boj.kr/2309](http://boj.kr/2309) 일곱 난쟁이

응용 : [boj.kr/3085](http://boj.kr/3085) 사탕 게임  
[boj.kr/2503](http://boj.kr/2503) 숫자 야구  
[boj.kr/2231](http://boj.kr/2231) 분해합