# #01-1. PS 맛보기

나정휘
https://justiceHui.github.io/

# 목차

# 강의 소개

# 강의 소개

## 강의 목표

- 2학년 자료구조, 알고리즘 과목 전체 범위 커버
- 기업 코딩테스트에 나오는 알고리즘 대부분 커버

## SCCC는 대회를 준비하는 소모임

- 강의 내용은 중고등학생 올림피아드 대비 기초 강의와 동일함
- 처음 공부하면 어려울 수 있음

## 강의 구성

- 은 뒤에서 소개함

# 강의 소개

| | 날짜 | 내용 |
|---|---|---|
| 1차시 | 23.06.26 | OT, 입출력, PS를 위한 C++ |
| 2차시 | 23.06.29 | 다차원 배열, 재귀 함수 |
| 3차시 | 23.07.03 | 기초 정수론 |
| 4차시 | 23.07.06 | 시간 복잡도, 정렬, 이분 탐색 |
| 5차시 | 23.07.10 | 수학적 귀납법, 분할 정복 |
| 6차시 | 23.07.13 | 동적 계획법 |
| 7차시 | 23.07.17 | 자료구조 1 (연결 리스트, 스택, 큐, 덱) |
| 8차시 | 23.07.20 | 알쓸신잡 |
| 9차시 | 23.07.24 | 그리디 |
| 10차시 | 23.07.27 | 그래프 이론 1 (그래프의 표현법, DFS, BFS, 위상 정렬) |
| 11차시 | TBD | 자료구조 2 (서로소 집합, 힙) |
| 12차시 | TBD | 그래프 이론 2 (최단 경로, 최소 신장 트리) |

# 강의 소개

## 1달 동안 공부할 내용

- 빠르게 계산하는 방법
- 잘 생각하는 방법
- 코드를 작성하는 방법

## 다양한 예시를 보며 감을 잡아보자

- 이거 다 이해 못해도 전혀 상관 없음
- 수학/과학 유튜브 보는 느낌으로 편하게 들으세요

# 빠르게 계산하는 방법

# 빠르게 계산하는 방법

어떻게 하면 빠르게 계산할 수 있을까?

- 고려해야 하는 경우의 수 자체를 줄이는 것
- 편하게 계산할 수 있는 형태로 변형하는 것
- 똑같은 값을 더 효율적으로 계산하는 것
- ...

# 빠르게 계산하는 방법

## BOJ 2003. 수들의 합 2

- 양의 정수로 구성된 배열 A가 주어짐
- A[i] + A[i+1] + … + A[j-1] + A[j] = M이 되는 경우의 수를 구하는 문제

- 배열 크기 꿀팁
  - N개의 수가 주어진다 → N의 최댓값보다 큰 배열 생성
  - 10000, 100000 처럼 하면 배열 크기 실수하기 쉬움
  - 10101, 101010 처럼 하면 끝자리 보고 확인할 수 있음
  - 10’000, 100’000 처럼 중간에 작은 따옴표 찍어도 됨

```c
#include <stdio.h>

int N, M, A[10101], R;

int f(int i, int j){
    int res = 0;
    for(int k=i; k<=j; k++) res += A[k];
    return res;
}

int main(){
    scanf("%d %d", &N, &M);
    for(int i=1; i<=N; i++) scanf("%d", &A[i]);
    for(int i=1; i<=N; i++){
        for(int j=i; j<=N; j++){
            if(f(i, j) == M) R += 1;
        }
    }
    printf("%d", R);
}
```

# 빠르게 계산하는 방법

## BOJ 2003. 수들의 합 2

- 양의 정수로 구성된 배열 A가 주어짐
- A[i] + A[i+1] + … + A[j-1] + A[j] = M이 되는 경우의 수를 구하는 문제

- 얼마나 많은 연산이 필요할까?
  - $T(N) = \sum_{i=1}^{N}\sum_{j=i}^{N}\sum_{k=i}^{j} 1 = \sum_{i=1}^{N}\sum_{j=i}^{N}(j-i+1)$
  - $= \sum_{i=1}^{N}\sum_{j=i}^{N} j + \sum_{i=1}^{N}\sum_{j=i}^{N}(1-i)$
  - $= \sum_{i=1}^{N}\frac{N(N+1)-i(i-1)}{2} + \sum_{i=1}^{N}(N-i+1)(1-i)$
  - $= \frac{1}{2}\sum_{i=1}^{N}(N^2+N-i^2+i) + \sum_{i=1}^{N}(N-Ni-i+i^2+1-i)$
  - $= \frac{N^3}{2} + \frac{N^2}{2} - \frac{N(N+1)(2N+1)}{12} + \frac{N(N+1)}{4}$
    $+N^2 - \frac{N^2(N+1)}{2} - \frac{N(N+1)}{2} + \frac{N(N+1)(2N+1)}{6} + N - \frac{N(N+1)}{2}$
  - $\approx \frac{N^3}{2} - \frac{N^3}{6} - \frac{N^3}{2} + \frac{N^3}{3} = \frac{N^3}{6}$

```c
#include <stdio.h>

int N, M, A[10101], R;

int f(int i, int j){
    int res = 0;
    for(int k=i; k<=j; k++) res += A[k];
    return res;
}

int main(){
    scanf("%d %d", &N, &M);
    for(int i=1; i<=N; i++) scanf("%d", &A[i]);
    for(int i=1; i<=N; i++){
        for(int j=i; j<=N; j++){
            if(f(i, j) == M) R += 1;
        }
    }
    printf("%d", R);
}
```

# 빠르게 계산하는 방법

## BOJ 2003. 수들의 합 2

- 양의 정수로 구성된 배열 A가 주어짐
- A[i] + A[i+1] + … + A[j-1] + A[j] = M이 되는 경우의 수를 구하는 문제

- 조금 더 빠르게!
  - $T(N) = \sum_{i=1}^{N} \sum_{j=i}^{N} 1 = \sum_{i=1}^{N} (N - i + 1)$
  - $= N^2 - \frac{N(N+1)}{2} + N \approx \frac{N^2}{2} < \frac{N^3}{6}$

```c
#include <stdio.h>

int N, M, A[10101], R;

int main(){
    scanf("%d %d", &N, &M);
    for(int i=1; i<=N; i++) scanf("%d", &A[i]);
    for(int i=1; i<=N; i++){
        int sum = 0;
        for(int j=i; j<=N; j++){
            sum += A[j];
            if(sum == M) R += 1;
        }
    }
    printf("%d", R);
}
```

# 빠르게 계산하는 방법

## BOJ 2003. 수들의 합 2

- 양의 정수로 구성된 배열 A가 주어짐
- A[i] + A[i+1] + … + A[j-1] + A[j] = M이 되는 경우의 수를 구하는 문제

- 더 빠르게!
  - 각각의 i마다 합이 M 이상이 되는 가장 작은 j인 J(i)를 생각해 보자.
  - A의 원소는 모두 양의 정수이기 때문에 J(i) ≤ J(i+1)
  - A에 0이 없으므로 i ~ J(i)의 합이 M인지만 확인해도 충분함
  - 따라서 실제로 고려해야 하는 구간의 개수는 N개

# 빠르게 계산하는 방법

## BOJ 2003. 수들의 합 2

- 양의 정수로 구성된 배열 A가 주어짐
- A[i] + A[i+1] + … + A[j-1] + A[j] = M이 되는 경우의 수를 구하는 문제

- 더 빠르게!
  - 각각의 i마다 합이 M 이상이 되는 가장 작은 j인 J(i)를 생각해 보자.
  - A의 원소는 모두 양의 정수이기 때문에 J(i) ≤ J(i+1)
  - A에 0이 없으므로 i ~ J(i)의 합이 M인지만 확인해도 충분함
  - 따라서 실제로 고려해야 하는 구간의 개수는 N개

  - i와 j는 감소하지 않고 증가만 하므로
  - 반복문은 최대 2N번 돌아감

  - 투 포인터 기법이라고 부름

```c
#include <stdio.h>

int N, M, A[10101], R;

int main(){
    scanf("%d %d", &N, &M);
    for(int i=1; i<=N; i++) scanf("%d", &A[i]);
    for(int i=1, j=1, sum=0; i<=N; sum-=A[i++]){
        while(j <= N && sum < M) sum += A[j++];
        if(sum == M) R += 1;
    }
    printf("%d", R);
}
```

# 질문?

# 빠르게 계산하는 방법

## 약수의 개수를 구하는 문제

- f(k) = k의 약수의 개수라고 정의하자.
- f(1) + f(2) + … + f(n-1) + f(n)을 계산하는 문제

- 가장 단순한 방법
  - $T(N) = \sum_{k=1}^{N} k = \frac{N(N+1)}{2}$

```c
#include <stdio.h>

int f(int k){
    int res = 0;
    for(int i=1; i<=k; i++){
        if(k % i == 0) res++;
    }
    return res;
}

int main(){
    int n, sum = 0;
    scanf("%d", &n);
    for(int i=1; i<=n; i++) sum += f(i);
    printf("%d", sum);
}
```

# 빠르게 계산하는 방법

## 약수의 개수를 구하는 문제

- f(k) = k의 약수의 개수라고 정의하자.
- f(1) + f(2) + … + f(n-1) + f(n)을 계산하는 문제

- 조금 더 빠른 방법
  - $T(N) = \sum_{k=1}^{N} \lfloor \sqrt{k} \rfloor \leq \sum_{k=1}^{N} \sqrt{N} = N\sqrt{N} < \frac{N(N+1)}{2}$
  - $N = pq\ (p, q > 0)$이면 $p, q$ 중 하나는 $\sqrt{N}$ 이하
  - 둘 모두 $\sqrt{N}$ 초과이면 $pq > N$이 되기 때문

```c
#include <stdio.h>

int f(int k){
    int res = 0;
    for(int i=1; i*i<=k; i++){
        if(k % i == 0) res++;
        if(k % i == 0 && i * i != k) res++;
    }
    return res;
}

int main(){
    int n, sum = 0;
    scanf("%d", &n);
    for(int i=1; i<=n; i++) sum += f(i);
    printf("%d", sum);
}
```

# 빠르게 계산하는 방법

## 약수의 개수를 구하는 문제

- f(k) = k의 약수의 개수라고 정의하자.
- f(1) + f(2) + … + f(n-1) + f(n)을 계산하는 문제

- 더 빠른 방법
  - 1부터 n까지의 약수의 개수를 세는 문제
  - 1부터 n까지의 n 이하의 배수의 개수를 세는 문제
  - 둘이 똑같은 문제

# 빠르게 계산하는 방법

## 약수의 개수를 구하는 문제

- f(k) = k의 약수의 개수라고 정의하자.
- f(1) + f(2) + … + f(n-1) + f(n)을 계산하는 문제

- 더 빠른 방법
  - 1부터 n까지의 약수의 개수를 세는 문제
  - 1부터 n까지의 n 이하의 배수의 개수를 세는 문제
  - 둘이 똑같은 문제

  - $T(N) = \sum_{i=1}^{N} \frac{N}{i} = N \sum_{i=1}^{N} \frac{1}{i}$
  - $\approx N \int_{1}^{N} \frac{1}{x} dx = N[\ln x]_{1}^{N} = N \ln N < N\sqrt{N}$

```c
#include <stdio.h>

int main(){
    int n, sum = 0;
    scanf("%d", &n);
    for(int i=1; i<=n; i++){
        for(int j=i; j<=n; j+=i) sum++;
    }
    printf("%d", sum);
}
```

# 빠르게 계산하는 방법

## 약수의 개수를 구하는 문제

- f(k) = k의 약수의 개수라고 정의하자.
- f(1) + f(2) + … + f(n-1) + f(n)을 계산하는 문제

- 더 빠른 방법
  - 1부터 n까지의 약수의 개수를 세는 문제
  - 1부터 n까지의 n 이하의 배수의 개수를 세는 문제
  - 둘이 똑같은 문제

  - 이렇게 하면 나눗셈 N번으로 됨
  - 사실 $\sqrt{N}$번으로도 가능한데… 지금은 몰라도 됨

```c
#include <stdio.h>

int main(){
    int n, sum = 0;
    scanf("%d", &n);
    for(int i=1; i<=n; i++) sum += n / i;
    printf("%d", sum);
}
```

# 질문?

# 빠르게 계산하는 방법

## BOJ 11659. 구간 합 구하기 4

- 양의 정수로 구성된 배열 A가 주어짐
- i, j가 주어지면 A[i] + A[i+1] + … + A[j-1] + A[j]를 구하는 쿼리를 M번 처리해야 함

- 가장 단순한 방법
  - 매번 a = 1, b = N으로 주어지면 NM 만큼의 연산 필요
  - 더 빠르게 할 수 있을까?

```c
#include <stdio.h>

int N, M, A[101010];

int main(){
    scanf("%d %d", &N, &M);
    for(int i=1; i<=N; i++) scanf("%d", &A[i]);
    for(int i=1; i<=M; i++){
        int a, b, sum = 0;
        scanf("%d %d", &a, &b);
        for(int j=a; j<=b; j++) sum += A[j];
        printf("%d\n", sum);
    }
}
```

# 빠르게 계산하는 방법

## BOJ 11659. 구간 합 구하기 4

- 양의 정수로 구성된 배열 A가 주어짐
- i, j가 주어지면 A[i] + A[i+1] + … + A[j-1] + A[j]를 구하는 쿼리를 M번 처리해야 함

- 더 빠른 방법
  - 고등학교에서 수열의 합을 구하던 것을 생각해 보자.
  - $S_n = \sum_{k=1}^{n} A_k = S_{n-1} + A_n$ 이라고 정의하면
  - $\sum_{k=a}^{b} A_k = S_b - S_{a-1}$

  - $S$를 미리 계산해 두면 구간의 합은 뺄셈 1번으로 가능

# 빠르게 계산하는 방법

## BOJ 11659. 구간 합 구하기 4

- 양의 정수로 구성된 배열 A가 주어짐
- i, j가 주어지면 A[i] + A[i+1] + … + A[j-1] + A[j]를 구하는 쿼리를 M번 처리해야 함

- 더 빠른 방법
  - 고등학교에서 수열의 합을 구하던 것을 생각해 보자.
  - $S_n = \sum_{k=1}^{n} A_k = S_{n-1} + A_n$ 이라고 정의하면
  - $\sum_{k=a}^{b} A_k = S_b - S_{a-1}$

  - $S$를 미리 계산해 두면 구간의 합은 뺄셈 1번으로 가능
  - $S$를 보통 누적 합 배열이라고 부름

  - 인덱스 0부터 사용하면 S[a-1] 에서 오류 발생할 수 있음

```c
#include <stdio.h>

int N, M, A[101010], S[101010];

int main(){
    scanf("%d %d", &N, &M);
    for(int i=1; i<=N; i++) scanf("%d", &A[i]);
    for(int i=1; i<=N; i++) S[i] = S[i-1] + A[i];
    for(int i=1; i<=M; i++){
        int a, b;
        scanf("%d %d", &a, &b);
        printf("%d\n", S[b] - S[a-1]);
    }
}
```

# 질문?

# 잘 생각하는 방법

# 잘 생각하는 방법

잘 생각하는 것이란?

- 문제를 잘 만져서 내가 아는 개념을 적용할 수 있는 형태로 변형하는 것
- 똑같은 것을 똑같다고 생각하는 것
- 문제를 수식으로 잘 표현한 다음 성질을 관찰하는 것
- ...

# 잘 생각하는 방법

## BOJ 8298. Coins

- R과 O로 구성되어 있는 문자열이 주어짐
- O의 개수가 정확히 R의 개수의 K배인 구간의 최대 길이를 구하는 문제
- ex. RORROOROOROOORO, K=3

# 잘 생각하는 방법

## BOJ 8298. Coins

- R과 O로 구성되어 있는 문자열이 주어짐
- O의 개수가 정확히 R의 개수의 K배인 구간의 최대 길이를 구하는 문제
- ex. RORROOROOROOORO, K=3

- K = 1이면 R의 개수와 O의 개수가 같은 구간을 찾는 문제
  - R을 +1, O를 -1이라고 생각하면 합이 0인 구간을 찾는 문제
  - S[j] = S[i-1] 인 구간을 찾는 문제

# 잘 생각하는 방법

## BOJ 8298. Coins

- R과 O로 구성되어 있는 문자열이 주어짐
- O의 개수가 정확히 R의 개수의 K배인 구간의 최대 길이를 구하는 문제
- ex. RORROOROOROOORO, K=3

- K = 1이면 R의 개수와 O의 개수가 같은 구간을 찾는 문제
  - R을 +1, O를 -1이라고 생각하면 합이 0인 구간을 찾는 문제
  - S[j] = S[i-1] 인 구간을 찾는 문제

- K > 1이면?
  - R을 +K, O를 -1이라고 생각하면 똑같이 풀 수 있음

# 질문?

# 잘 생각하는 방법

## BOJ 4307. 개미

- 길이가 L인 막대 위에 N마리의 개미가 있음
- 각 개미는 왼쪽 또는 오른쪽으로 1 단위 속도로 이동하고 있음
- 개미가 막대의 끝에 도달하면 떨어짐
- 두 개미가 만나면 방향을 반대로 바꾸어 이동함
- 개미의 초기 위치와 방향이 주어지면 가장 먼저/마지막으로 개미가 떨어지는 시간 구하는 문제

# 잘 생각하는 방법

## BOJ 4307. 개미

- 길이가 L인 막대 위에 N마리의 개미가 있음
- 각 개미는 왼쪽 또는 오른쪽으로 1 단위 속도로 이동하고 있음
- 개미가 막대의 끝에 도달하면 떨어짐
- 두 개미가 만나면 방향을 반대로 바꾸어 이동함
- 개미의 초기 위치와 방향이 주어지면 가장 먼저/마지막으로 개미가 떨어지는 시간 구하는 문제

- 두 개미가 튕겨져 나가는 것 = 서로를 스쳐 지나가는 것
  - 각 개미가 초기 상태에서 끝까지 이동하는데 걸리는 최소/최대 시간을 구하면 됨

# 질문?

# 잘 생각하는 방법

## BOJ 9012. 괄호

- 괄호 문자열이 주어지면 올바른 괄호 문자열인지 판별하는 문제
- 예시
  - (())() 는 올바른 괄호 문자열
  - )()(() 는 올바르지 않은 괄호 문자열
  - ))(() 는 올바르지 않은 괄호 문자열

# 잘 생각하는 방법

## BOJ 9012. 괄호

- 괄호 문자열이 주어지면 올바른 괄호 문자열인지 판별하는 문제
- 예시
  - (())() 는 올바른 괄호 문자열
  - )()(() 는 올바르지 않은 괄호 문자열
  - ())(() 는 올바르지 않은 괄호 문자열

- 올바른 괄호 문자열의 조건
  - 여는 괄호의 개수 = 닫는 괄호의 개수
  - 앞에서부터 봤을 때 항상 (여는 괄호의 개수 ≥ 닫는 괄호의 개수)를 만족해야 함

# 잘 생각하는 방법

## BOJ 9012. 괄호

- 괄호 문자열이 주어지면 올바른 괄호 문자열인지 판별하는 문제
- 예시
  - (())() 는 올바른 괄호 문자열
  - ()()(() 는 올바르지 않은 괄호 문자열
  - ())(() 는 올바르지 않은 괄호 문자열

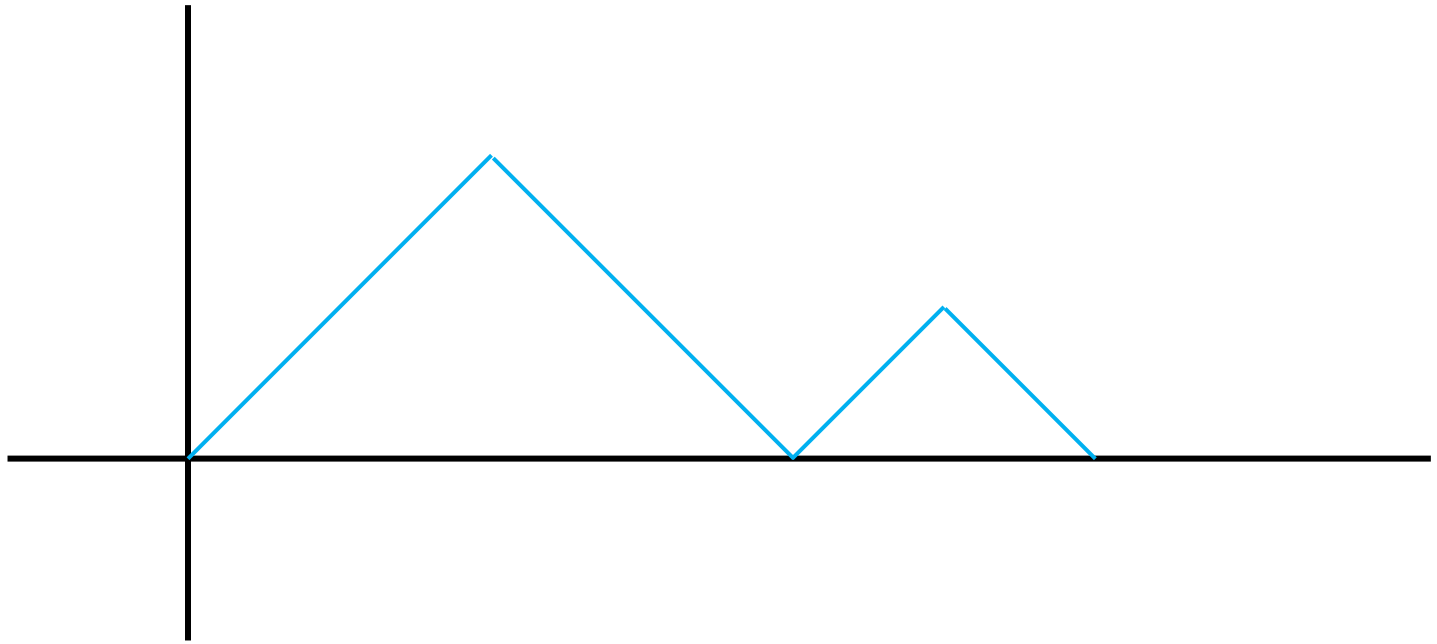
- 올바른 괄호 문자열의 조건
  - 여는 괄호의 개수 = 닫는 괄호의 개수
  - 앞에서부터 봤을 때 항상 (여는 괄호의 개수 ≥ 닫는 괄호의 개수)를 만족해야 함

  - 여는 괄호를 +1, 닫는 괄호를 -1로 취급하는 수열 A와 누적 합 배열 S를 만들면
  - S[N] = 0 and ∀i, A[i] ≥ 0을 만족하는지 확인하면 됨

# 잘 생각하는 방법

## BOJ 9012. 괄호

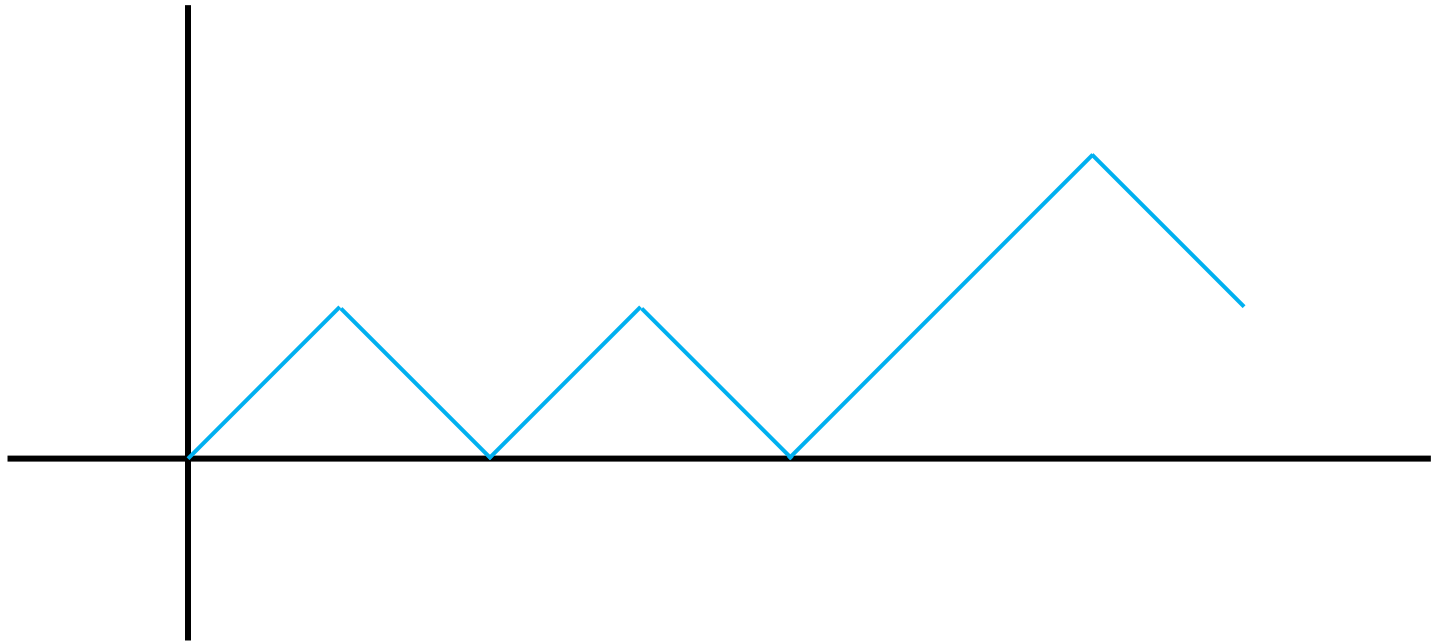
- 괄호 문자열이 주어지면 올바른 괄호 문자열인지 판별하는 문제
- 예시
  - (())() 는 올바른 괄호 문자열
  - )()(() 는 올바르지 않은 괄호 문자열
  - ())(() 는 올바르지 않은 괄호 문자열



# 잘 생각하는 방법

## BOJ 9012. 괄호

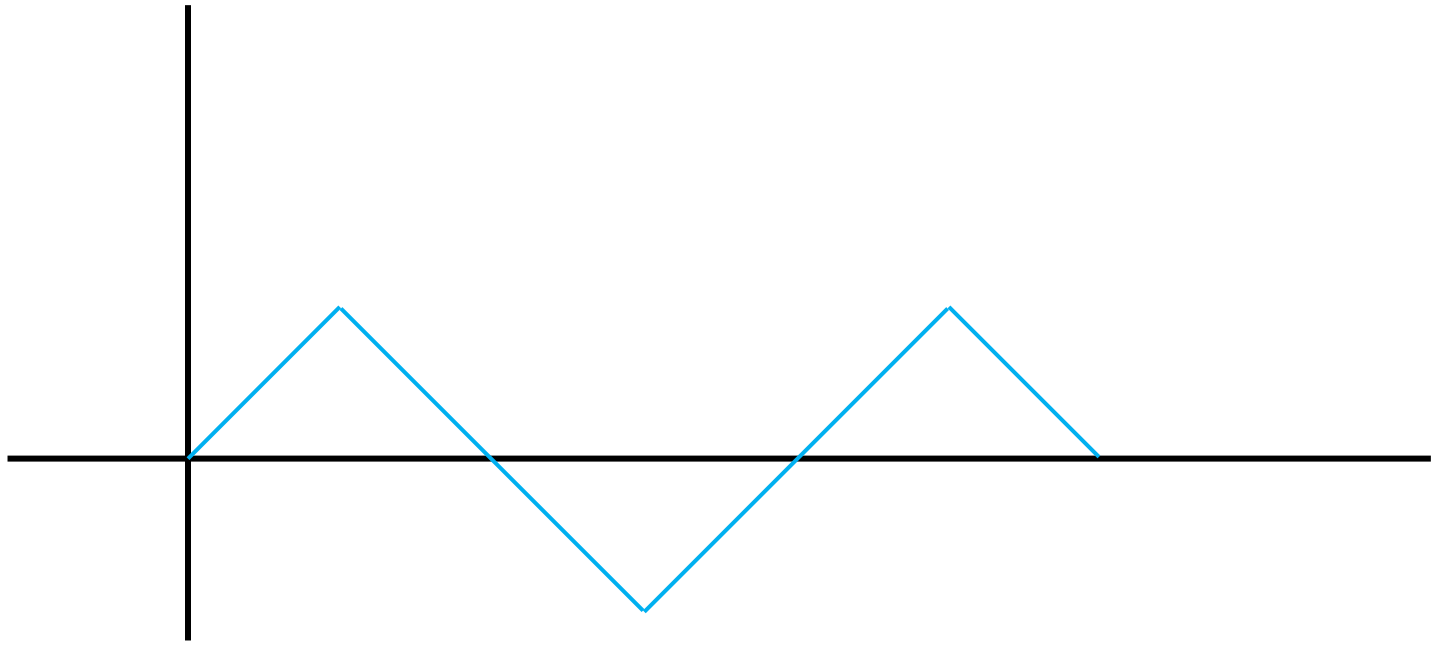
- 괄호 문자열이 주어지면 올바른 괄호 문자열인지 판별하는 문제
- 예시
  - (())() 는 올바른 괄호 문자열
  - )()(() 는 올바르지 않은 괄호 문자열
  - ())(() 는 올바르지 않은 괄호 문자열



# 잘 생각하는 방법

## BOJ 9012. 괄호

- 괄호 문자열이 주어지면 올바른 괄호 문자열인지 판별하는 문제
- 예시
  - (())() 는 올바른 괄호 문자열
  - )()(() 는 올바르지 않은 괄호 문자열
  - ))(() 는 올바르지 않은 괄호 문자열



# 질문?

# 잘 생각하는 방법

## 택시 거리 최소화

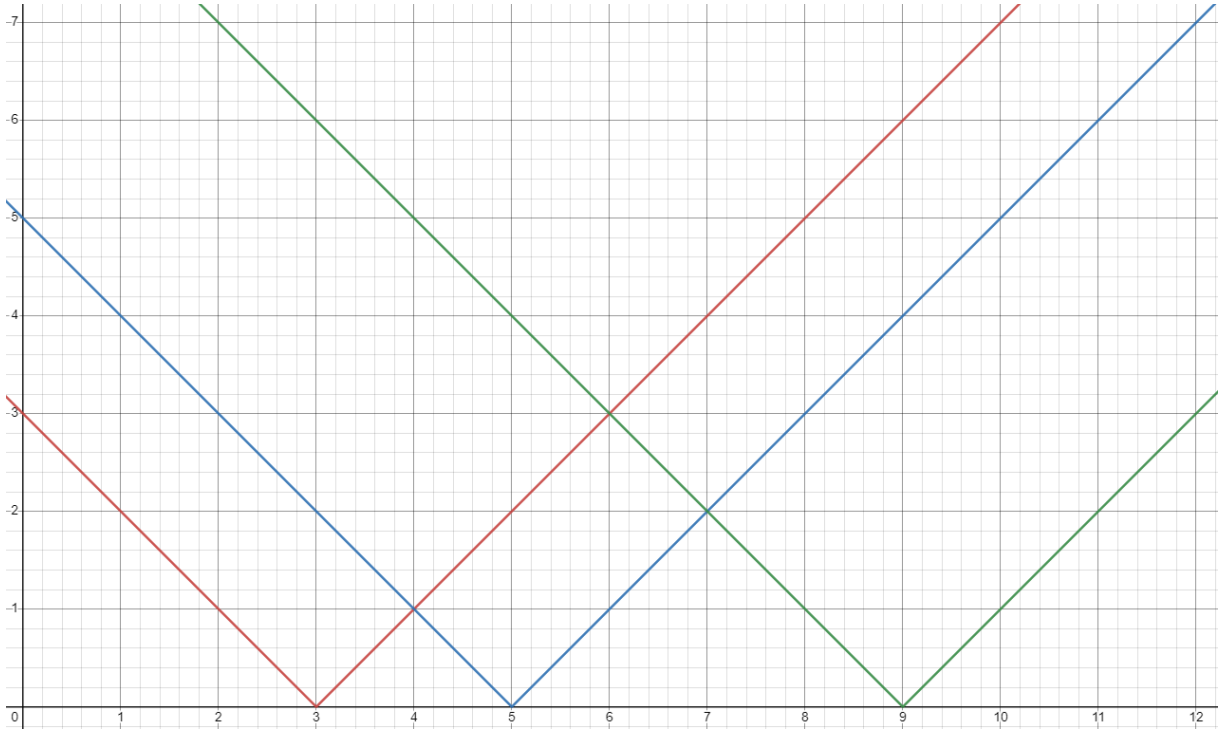
- 두 점 $(x_1, y_1), (x_2, y_2)$ 사이의 거리를 $|x_1 - x_2| + |y_1 - y_2|$ 라고 정의하자.
- $N$개의 점이 주어졌을 때 $\sum(|x_i - x| + |y_i - y|)$ 를 최소화하는 점 $(x, y)$ 를 찾는 문제

- 필요한 관찰
  - x좌표와 y좌표는 분리해서 생각해도 됨
  - $\sum |x_i - x|$ 를 최소화하는 $x$와 $\sum |y_i - y|$ 를 최소화하는 $y$를 각각 구하는 문제

# 잘 생각하는 방법

## 택시 거리 최소화

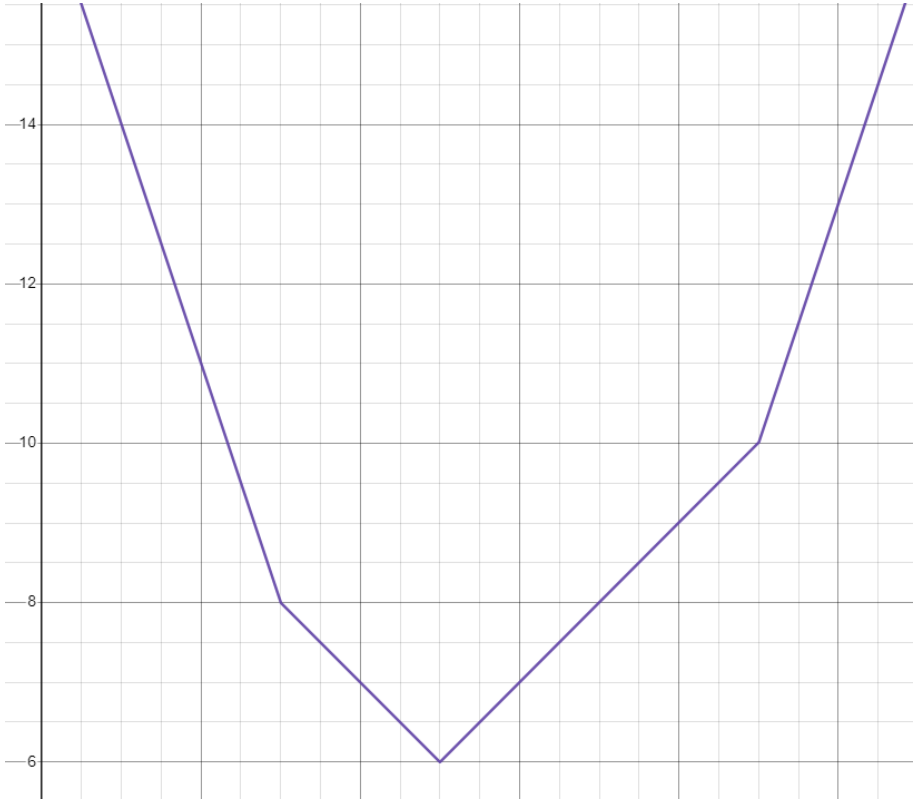
- 필요한 관찰
  - x좌표와 y좌표는 분리해서 생각해도 됨
  - $\sum |x_i - x|$ 를 최소화하는 $x$와 $\sum |y_i - y|$ 를 최소화하는 $y$를 각각 구하는 문제
  - $f_i(x) = |x_i - x|$ 라고 정의하면 $f_i$는 $x_i$를 중심으로 V자 형태로 뻗어나가는 형태
    - $x_1 = 3, x_2 = 5, x_3 = 9$ 일 때의 그래프



# 잘 생각하는 방법

## 택시 거리 최소화

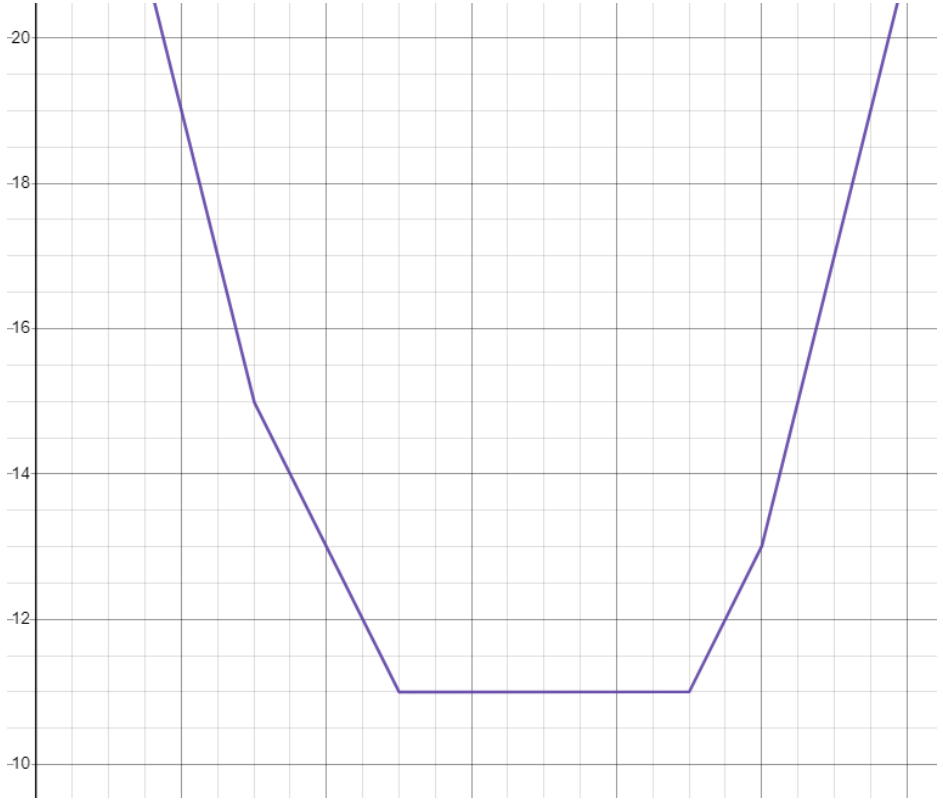
- 필요한 관찰
  - $f_i(x) = |x_i - x|$ 라고 정의하면 $f_i$는 $x_i$를 중심으로 V자 형태로 뻗어나가는 형태
  - $\sum f_i(x)$가 최소가 되는 $x$를 구하는 문제
    - $x_1 = 3, x_2 = 5, x_3 = 9$ 일 때의 그래프



# 잘 생각하는 방법

## 택시 거리 최소화

- 필요한 관찰
  - $f_i(x) = |x_i - x|$ 라고 정의하면 $f_i$는 $x_i$를 중심으로 V자 형태로 뻗어나가는 형태
  - $\sum f_i(x)$가 최소가 되는 $x$를 구하는 문제
    - $x_1 = 3, x_2 = 5, x_3 = 9, x_4 = 10$ 일 때의 그래프



# 잘 생각하는 방법

## 택시 거리 최소화

- 필요한 관찰
  - $f_i(x) = |x_i - x|$ 라고 정의하면 $f_i$는 $x_i$를 중심으로 V자 형태로 뻗어나가는 형태
  - $\sum f_i(x)$가 최소가 되는 $x$를 구하는 문제
    - 점이 홀수 개면 기울기가 음수에서 양수로 바뀌는 지점
    - 점이 짝수 개면 기울기가 0인 지점
  - $f_i(x)$는 $x < x_i$인 지점의 기울기에 -1, $x > x_i$인 지점의 기울기에 +1 만큼 기여함
    - 점이 홀수 개면 중앙값에서 기울기가 음수 → 양수로 바뀜
    - 점이 짝수 개면 가운데 있는 두 점 사이에서 기울기 0임
  - 중앙값을 구하면 해결!

# 질문?