#1 프로그래밍 문제 접근하기

2019 SCSC Summer Coding Workshop

서강대학교 컴퓨터공학과 박수현

me@shiftpsh.com

강사 소개

박수현 – shiftpsh.com
서강대학교 컴퓨터공학과 학부 18학번

약력

- 컴퓨터공학과 알고리즘 문제해결 연구 학회 Sogang ACM-ICPC Team 학회장 (2019년 1월–)
- 하이퍼커넥트에서 Android 엔지니어로 인턴 (2018년 6월–8월)
- 2019 삼성 대학생 프로그래밍 대회 Finalist
- 2019 Google Kick Start Round A 한국 16위
- 2018 서강 프로그래밍 대회 Master 부문 1위
- 백준 온라인 저지 43위

프로그래밍 문제

수 찾기

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	128 MB	35517	9460	6232	27.752%

문제

N개의 정수 A[1], A[2], ..., A[N]이 주어져 있을 때, 이 안에 X라는 정수가 존재하는지 알아내는 프로그램을 작성하시오.

입력

첫째 줄에 자연수 N(1≤N≤100,000)이 주어진다. 다음 줄에는 N개의 정수 A[1], A[2], ..., A[N]이 주어진다. 다음 줄에는 M(1≤M≤100,000)이 주어진다. 다음 줄에는 M개의 수들이 주어지는데, 이 수들이 A안에 존재하는지 알아내면 된다. 모든 정수들의 범위는 int 로 한다.

출력

M개의 줄에 답을 출력한다. 존재하면 1을, 존재하지 않으면 0을 출력한다.

예제 입력 1 복사

```
5
4 1 5 2 3
5
1 3 7 9 5
```

예제 출력 1 복사

```
1
1
0
0
1
```

프로그래밍 문제

시간 제한 코드가 실행되는 시간의 제한

메모리 제한 코드가 일정 순간 점유할 수 있는 메모리의 제한

이 문제의 시간 제한은 2초였으므로, 실행 시간이 2초를 초과하면 올바른 결과를 계산했더라도 문제를 틀리게 된다!

마찬가지로 메모리 제한은 128MB였으므로, 이를 초과하면 올바른 결과를 계산했더라도 문제를 틀리게 된다

프로그래밍 문제 해결 (Problem Solving)

- 논리적 또는 수학적 문제를 프로그래밍으로 해결
- 자료구조와 알고리즘을 이용해 효율적으로 해결
  - 시간 제한과 메모리 제한을 만족하면서 정답을 출력해야 한다 → 문제를 해결할 수 있는 최적의 자료구조와 알고리즘을 선택

PS를 연습할 수 있는 리소스

- **도서** - 여러 기법, 자료구조, 알고리즘 등을 배울 수 있다
- **★구글링★**
- **온라인 저지** - 온라인으로 문제를 풀어보고 채점받을 수 있다
- **각종 대회 출전** - 실력을 가늠해 볼 수 있다

PS를 연습할 수 있는 리소스 – 도서

- Antti Laaksonen, 『**알고리즘 트레이닝 : 프로그래밍 대회 입문 가이드**』, 인사이트, 2019
  - 원서 *Competitive Programmer's Handbook*은 cses.fi/book/에서 무료 공개 중이다
- 구종만, 『**알고리즘 문제 해결 전략**』, 인사이트, 2012
- 秋葉拓哉 외, 『**프로그래밍 콘테스트 챌린징**』, 로드북, 2011 (절판)
  - 원서 『プログラミングコンテストチャレンジブック』는 2판까지 나왔다

PS를 연습할 수 있는 리소스 – 온라인 저지

- **백준 온라인 저지**(BOJ) – acmicpc.net
  - 풀 수 있는 문제가 엄청 많다 (13000개+)
  - 지원되는 언어도 엄청 많다 (65개)
  - ACM-ICPC, IOI 등이나 Google Code Jam, 카카오 코드 페스티벌 등의 대회 문제들도 풀어볼 수 있다
  - 앞으로 문제에 BOJ #**1000** 같은 태그가 달려 있다면 BOJ에서 1000번 문제로 수록되어 있어 직접 풀어볼 수 있다는 뜻

PS를 연습할 수 있는 리소스 – 온라인 저지

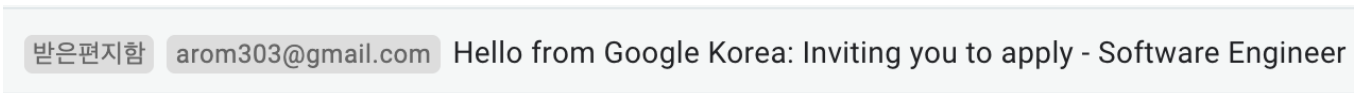
- **SW Expert Academy**(SWEA) – swexpertacademy.com
  - 삼성전자에서 운영하는 온라인 저지
  - 문제나 언어 수는 BOJ가 훨씬 많긴 하나 여기엔 삼성에서 만든 모의 SW 역량 테스트 문제들이 있다
  - 동영상 강의도 무료로 제공하는 듯 하다

PS를 연습할 수 있는 리소스 – 대회

- **ACM-ICPC** 국제 대학생 프로그래밍 경진대회
  - 세계 최고 권위의 프로그래밍 대회
  - 3명이 1팀, 함께 3-5시간 동안 10-12문제 정도를 해결
  - 기출 문제는 BOJ에서 풀어볼 수 있다

PS를 연습할 수 있는 리소스 – 각종 기업 대회

- Google – g.co/codingcompetitions
  - **hash code** – 팀 대회, 연초에 열린다
  - **code jam** – 개인 대회, 매년 3월에서 6월 정도에 열린다
  - **kick start** – 채용 성격의 학생 개인 대회 (상금은 없음), 매년 2월부터 11월까지 8번씩 열린다



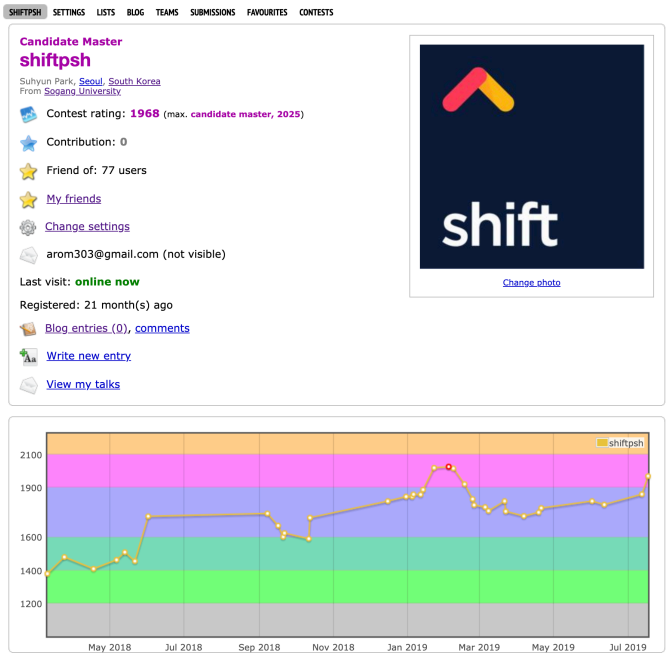
실제로 kick start에서 좋은 성적을 거두면 메일로 유용한 정보가 옵니다 (자랑 맞습니다)

PS를 연습할 수 있는 리소스 – 각종 기업 대회

- Facebook – facebook.com/hackercup/contest
  - **Hacker Cup** – 개인 대회, 매년 6월 정도에 열린다
- 삼성 – codeground.org
  - **SCPC** 삼성 대학생 프로그래밍 대회 – 대학(원)생 개인 대회, 매년 6월에서 7월 정도에 열린다
- 카카오 – kakaocode.com
  - **코드 페스티벌** – 개인 대회, 매년 8월 정도에 열린다

PS를 연습할 수 있는 리소스 – 커뮤니티 대회

- **Codeforces** – codeforces.com
  - 러시아 기반 사이트
  - 일주일에 한 번 이상 꾸준히 대회가 열리며, 결과에 따라 레이팅이 부여된다
  - 레이팅에 따라 자신의 실력을 가늠해 볼 수 있다



PS를 연습할 수 있는 리소스 – 커뮤니티 대회

- **Codeforces** – codeforces.com

Newbie		–	1199	상위	100%
Pupil	1200	–	1399	상위	84%
Specialist	1400	–	1599	상위	45%
Expert	1600	–	1899	상위	19%
Candidate Master	1900	–	2099	상위	7%
Master	2100	–	2399	상위	3%
Grandmaster	2400	–	2999	상위	0.7%
Legendary Grandmaster	3000	–		상위	0.05%

레이팅은 1500 근처에서 시작하며 컨테스트 결과에 따라 오르내린다

PS를 연습할 수 있는 리소스 – 커뮤니티 대회

이외에도..

- 일본 기반의 **AtCoder** – atcoder.jp
- 인도 기반의 **CodeChef** – codechef.com
- 미국 기반의 **TopCoder** – topcoder.com

PS를 연습할 수 있는 리소스

코딩 테스트만 준비할 거라면?

- 어려운 알고리즘을 익히려고 하기보단, 학부 수준에서 배우는 자료구조와 알고리즘들을 제대로 이해하고 문제의 요구사항을 코드로 완벽히 구현하는 능력이 있어야 한다

프로그래밍 대회에서 입상하고 싶다면?

- 빠르고 정확하게 구현하는 능력은 기본, 어려운 알고리즘도 일부 알아야 하며 많은 문제를 접해 보고 결과적으로 처음 보는 문제라도 아이디어가 빨리 떠오를 수 있어야 한다

언어 선택

알고리즘은 언어에 종속적이지 않기 때문에 어떤 언어를 선택해도 상관없다 $\rightarrow$ 따라서 가장 자신있는 언어를 하는 게 최선!

하지만..

- 코딩 테스트를 준비한다면 지원하는 회사가 허용하는 언어가 무엇인지 체크
  - C/C++만 사용가능한 곳도 있다
  - Java, Python, 특히 C++은 거의 모든 곳에서 허용되고 있으므로 이들이 상대적으로 유리
- 공부할 리소스는 C++이 압도적으로 많다
  - 프로그래밍 대회 커뮤니티에서 대부분 작성했기 때문
  - 프로그래밍 대회에서는 C++을 쓰는 게 압도적으로 유리하다

언어 선택

각 언어들이 기본적으로 제공하는 알고리즘과 자료구조들은 대부분 사용할 수 있으므로 미리 알아 두는 게 좋다

- **C++** – Standard Template Library (STL)
- **Java** – Collections API
- **Python** – Python Standard Library (stdlib)

이 강의에서는 언어적 요소는 모두 다 안다고 가정하고 비교적 생소한 것만 설명하는 것을 원칙으로 함

언어 선택

언어별 기본 제공 자료구조들

자료구조	C++	Java	Python
동적 배열	`vector`	`ArrayList`	-[1]
문자열	`string`	`String`	`string`
스택	`stack`	-[2]	-[2]
큐	`queue`	-[2]	-[2]
덱	`deque`	`ArrayDeque`	`deque`
힙	`priority_queue`[3]	`PriorityQueue`	`heapq`[4]

[1] Python 배열은 기본적으로 동적입니다.

[2] Java와 Python의 경우 덱을 스택과 큐로 사용하는 것이 좋습니다.

[3] `#include <queue>`

[4] `import heapq`

언어 선택

언어별 기본 제공 자료구조들 (cont.)

자료구조	C++	Java	Python
정렬된 맵	`map`	`TreeMap`	`OrderedDict`
해시 맵	`unordered_map`	`HashMap`	`dict`
정렬된 집합	`set`[5]	`TreeSet`	-[6]
해시 집합	`unordered_set`[5]	`HashSet`	`set`
2-튜플	`pair`	-[7]	-[8]
n-튜플	`tuple`	-[7]	-[8]

[5] 중복 원소 허용 집합의 경우 `set` 대신 `multiset`을 사용합니다.
[6] `OrderedDict`를 사용해야 합니다.
[7] 클래스를 새로 작성해야 합니다.
[8] Python에서 튜플은 기본 자료형입니다.

문제 접근

AC를 받으려면 모든 테스트 케이스를 다 맞춰야 하므로 빠르게 푸는 것보다는 정확히 푸는 것이 더 중요

1. 문제를 **이해**하고
2. 해결에 필요한 **아이디어**와 **식** 등을 정리한 뒤
3. 이를 **코드로 구현**하고
4. **예외가 생길 수 있을 만한 케이스들**을 최대한 생각해 본 뒤
5. 코드에 집어넣어 보고 **디버깅**
6. 가능할 경우 **최적화**까지

최소, 최대 BOJ #10818

- **문제** – N개의 정수가 주어진다. 이 때, 최솟값과 최댓값을 구하는 프로그램을 작성하시오.
- **입력** – 첫째 줄에 정수의 개수 $N(1 \le N \le 1,000,000)$이 주어진다. 둘째 줄에는 N개의 정수를 공백으로 구분해서 주어진다. 모든 정수는 $-1,000,000$보다 크거나 같고, $1,000,000$보다 작거나 같은 정수이다.
- **출력** – 첫째 줄에 주어진 정수 N개의 최솟값과 최댓값을 공백으로 구분해 출력한다.

최소, 최대 BOJ #10818

1. 문제 이해
   - N개의 정수가 주어진다
   - 최솟값과 최댓값을 구하는 게 목표
   - $1 \leq N \leq 1,000,000$이다

최소, 최대 BOJ #10818

2. 아이디어와 식 정리
   - 최솟값과 최댓값을 구하는 게 목표니까
   - 최솟값을 저장하는 변수 mn, 최댓값을 저장하는 변수 mx를 하나씩 만들어 두고
   - 모든 원소 a_i에 대해
     - $mn > a_i$면 mn을 a_i로 업데이트해야 mn이 최솟값으로 유지된다
     - $mx < a_i$면 mx을 a_i로 업데이트해야 mx가 최댓값으로 유지된다

최소, 최대 BOJ #10818

3. 코드로 구현

```cpp
#include <iostream>
using namespace std;
#define INF 987654321
int a[1000000];
int main() {
    int mn = INF, mx = 0, n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    for (int i = 0; i < n; i++) {
        if (mn > a[i]) mn = a[i];
        if (mx < a[i]) mx = a[i];
    }
    cout << mn << ' ' << mx << '\n';
    return 0;
}
```

최소, 최대 BOJ #10818

3. 코드로 구현 (cont.)

```cpp
#include <iostream>
using namespace std;
#define INF 987654321
int main() {
    int mn = INF, mx = 0, n, x;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> x;
        if (mn > x) mn = x;
        if (mx < x) mx = x;
    }
    cout << mn << ' ' << mx << '\n';
    return 0;
}
```

‘어 근데 굳이 배열에 입력을 받을 필요가 있을까?’

최소, 최대 BOJ #10818

4. 예외 케이스 생각
   - $N = 1$일 때는 잘 돌아갈까?
   - 배열에 음수가 섞여 있어도 잘 돌아갈까?

프로그램에 직접 입력을 넣어 보는 것도 좋다

최소, 최대 BOJ #10818

4. 예외 케이스 생각 (cont.)

```
1
5
```

```
5
-1 -2 -3 -4 -5
```

위 케이스에서는 `5  5`가, 아래 케이스에서는 `-5  0`이 각각 출력되었다

최소, 최대 BOJ #10818

5. 디버깅

```cpp
#include <iostream>
using namespace std;
#define INF 987654321
int main() {
    int mn = INF, mx = -INF, n, x;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> x;
        if (mn > x) mn = x;
        if (mx < x) mx = x;
    }
    cout << mn << ' ' << mx << '\n';
    return 0;
}
```

mx의 초기값이 0이어서 음수들만 있는 경우 최댓값을 제대로 계산하지 못했다

최소, 최대 BOJ #10818

딱히 최적화할 거리는 안 보이고, 충분히 검증한 거 같다

→ 제출 → **AC**

잠시 휴식!

시간 제한, 메모리 제한

- 코드가 점유하는 메모리는 계산하기 쉽다 - `int` 하나는 무조건 4바이트, 그러면 `int` 10^6 개의 배열은 4MB
- 게다가 메모리 제한은 보통 128MB, 256MB, 512MB와 같이 넉넉하다
- 그러면 코드가 동작하는 시간은?

실행 시간 예측하기

시간 복잡도

- 보통 $\mathcal{O}(f(n))$으로 나타냄 – 여기서 n은 입력의 크기
- $f(n)$은 간단히 말하자면 입력의 크기에 따라 연산이 실행되는 횟수에 대략적으로 비례하는 함수

실행 시간 예측하기 – 시간 복잡도 Review

```cpp
for (int i = 1; i ⩽ n; i++) {
    cout << i << '\n';
}
```

$$\mathcal{O}(n)$$

실행 시간 예측하기 – 시간 복잡도 Review

```cpp
for (int i = 2; i ⩽ n; i += 2) {
    cout << i << '\n';
}
```

$$\mathcal{O}(n)$$

실행 시간 예측하기 – 시간 복잡도 Review

```cpp
for (int i = 1; i ⩽ n; i++) {
    for (int j = 1; j ⩽ n; j++) {
        cout << i * j << ' ';
    }
    cout << '\n';
}
```

$$\mathcal{O}\left(n^2\right)$$

실행 시간 예측하기 – 시간 복잡도 Review

```cpp
for (int i = 1; i ⩽ n; i++) {
    for (int j = 1; j ⩽ i; j++) {
        cout << i * j << ' ';
    }
    cout << '\n';
}
```

$$\mathcal{O}\left(n^2\right)$$

실행 시간 예측하기 – 시간 복잡도 Review

```cpp
for (int i = 1; i ⩽ n; i++) {
    cout << i << '\n';
}
for (int i = 1; i ⩽ n; i++) {
    for (int j = 1; j ⩽ i; j++) {
        cout << i * j << ' ';
    }
    cout << '\n';
}
```

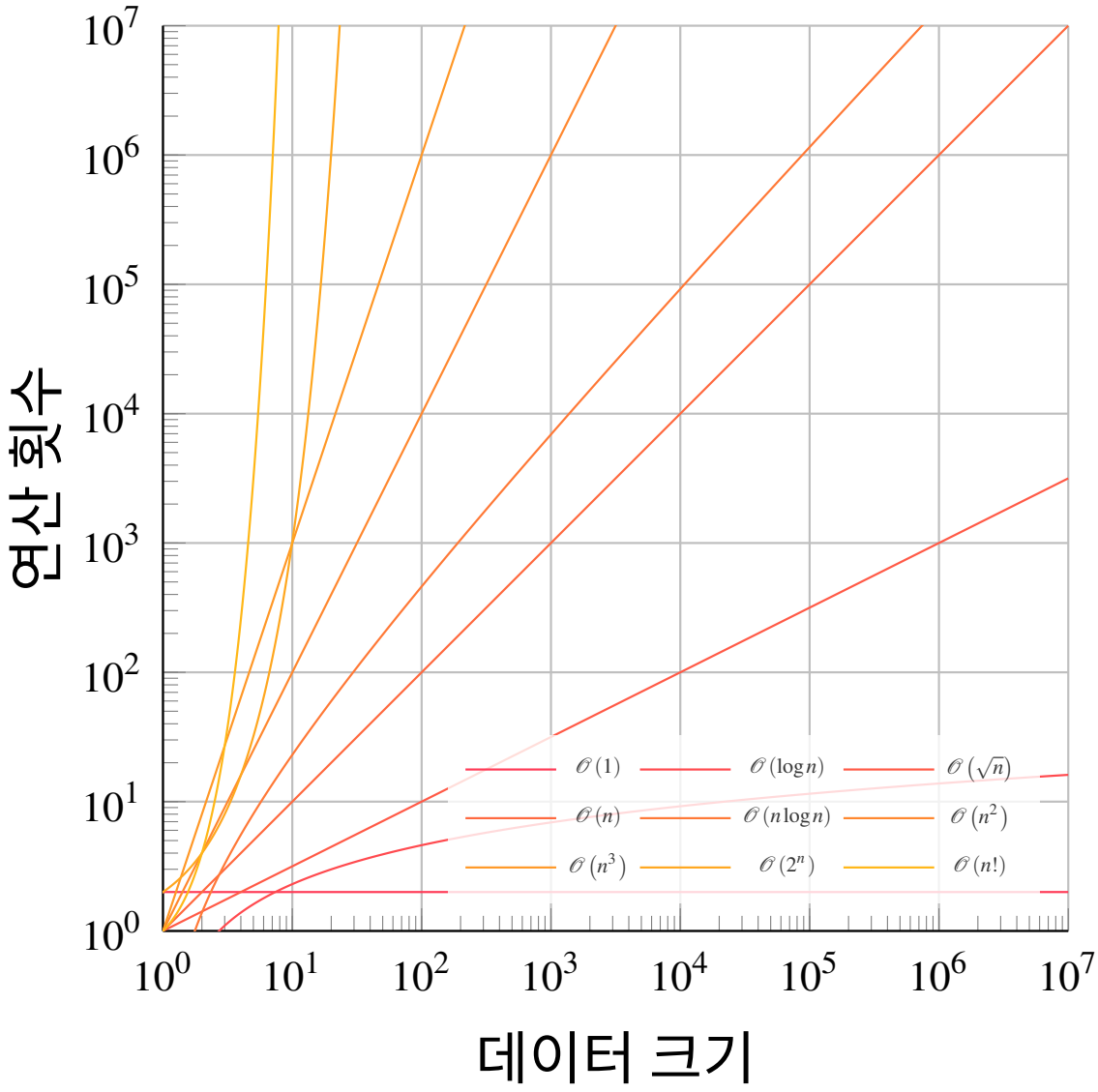
$$\mathcal{O}\left(n^2\right)$$

실행 시간 예측하기 – 시간 복잡도 Review

```cpp
for (int i = 1; i ⩽ n; i++) {
    for (int j = 1; j ⩽ m; j++) {
        cout << i * j << ' ';
    }
    cout << '\n';
}
```

$$\mathcal{O}(nm)$$

실행 시간 예측하기 – 자주 등장하는 복잡도



10^8 룰

컴퓨터는 1 초에 간단한 동작을 대략 10^8 번 할 수 있다

→ 시간 복잡도에 상수를 대입하면 시간 제한을 초과하는지 아닌지 대략적으로 알 수 있다

- $\mathcal{O}(n)$, $n \leq 10^6$? → AC
- $\mathcal{O}(n^2)$, $n \leq 10^5$? → TLE
- $\mathcal{O}(n \log n)$, $n \leq 10^5$? → AC

10^8 룰

n 제한이 주어지면 역으로 허용되는 시간 복잡도를 계산할 수 있다

시간 복잡도	n
$\mathcal{O}(1)$	∞
$\mathcal{O}(\log n)$	$\cdots 2^{10^8}$
$\mathcal{O}(\sqrt{n})$	$\cdots 10^{16}$
$\mathcal{O}(n)$	$\cdots 10^8$
$\mathcal{O}(n \log n)$	$\cdots 4 \times 10^6$
$\mathcal{O}(n^2)$	$\cdots 10^4$
$\mathcal{O}(n^3)$	$\cdots 450$
$\mathcal{O}(2^n)$	$\cdots 25$
$\mathcal{O}(n!)$	$\cdots 11$

10^8 룰 – 맹신하지 말 것

- 시간 복잡도에는 계수가 생략되어 있음
- 같은 $\mathcal{O}(n \log n)$ 짜리 알고리즘이라도 어떤 알고리즘은 $n \log n$ 번의 연산을, 어떤 알고리즘은 $10n \log n$ 번의 연산을 한다고 하면 실행 속도는 당연히 10 배 차이
- CPU 동작 같은 세세한 요소에 따라 프로그램 실행 시간이 영향을 받을 수도 있음

10^8 룰 – 맹신하지 말 것

- 시간 복잡도에 상수를 대입했더니 10^7 에서 10^9 사이 정도가 나왔다 $\rightarrow$ 코드를 더 자세히 분석해야 한다
- 운이 좋으면 적절한 최적화로 AC 를 받을 수도 있다

예를 들어 이번 SCPC Round 2의 2번 문제는 $T \leq 40, N \leq 5000$ 이었으나 $\mathcal{O}\left(TN^2\right) \rightarrow 10^9$ 가 0.6초만에 돌아갔다

Maximum Subarray BOJ #10211

크기 N인 정수형 배열 X가 있을 때, X의 부분 배열(X의 연속한 일부분) 중 각 원소의 합이 가장 큰 부분 배열을 찾는 maximum subarray problem은 컴퓨터 과학에서 매우 잘 알려져 있다.

여러분은 N과 배열 X가 주어졌을 때, X의 maximum subarray의 합을 구하자. 즉, $\max_{1 \le i \le j \le N} (X_i + \cdots + X_j)$를 구하자.

- $1 \le N \le 1,000$, $|X_i| < 1,000$

Maximum Subarray BOJ #10211 – $\mathcal{O}\left(n^3\right)$

$1 \leq N \leq 1,000$을 무시한다면 $\mathcal{O}\left(n^3\right)$ 솔루션은 쉽게 생각할 수 있다

- 모든 $1 \leq i \leq j \leq N$에 대해 $X_i \cdots X_j$를 전부 더해 보고 그 중 최댓값을 출력

Maximum Subarray BOJ #10211 – $\mathcal{O}\left(n^3\right)$

```
5   int X[1000];
6   int main() {
7       int t;
8       cin >> t;
9       while (t--) {
10          int mx = -INF, n;
11          cin >> n;
12          for (int i = 0; i < n; i++) {
13              cin >> X[i];
14          }
15          for (int i = 0; i < n; i++) {
16              for (int j = i; j < n; j++) {
17                  int sum = 0; // sum of X_i ··· X_j
18                  for (int k = i; k <= j; k++) sum += X[k];
19                  mx = max(mx, sum);
20              }
21          }
22          cout << mx << '\n';
23      }
24      return 0;
25  }
```

Maximum Subarray BOJ #10211 – $\mathcal{O}\left(n^2\right)$

근데 생각해 보면 $X_i \cdots X_j$를 매번 전부 더할 필요는 없을지도 모르겠다

- 모든 $1 \leq i \leq N$에 대해
  - $sum = 0$이라는 정수를 하나 만들어 두고..
  - 모든 $i \leq j \leq N$에 대해 X_j를 sum에 누적

하면 sum에는 각 j마다 $X_i \cdots X_j$의 합이 들어있게 된다

Maximum Subarray BOJ #10211 – $\mathcal{O}\left(n^2\right)$

```
5   int X[1000];
6   int main() {
7       int t;
8       cin >> t;
9       while (t--) {
10          int mx = -INF, n;
11          cin >> n;
12          for (int i = 0; i < n; i++) {
13              cin >> X[i];
14          }
15          for (int i = 0; i < n; i++) {
16              int sum = 0; // sum of X_i ··· X_j
17              for (int j = i; j < n; j++) {
18                  sum += X[j];
19                  mx = max(mx, sum);
20              }
21          }
22          cout << mx << '\n';
23      }
24      return 0;
25  }
```

Maximum Subarray BOJ #10211 – $\mathcal{O}(n)$

놀랍게도 $\mathcal{O}(n)$만에 풀 수도 있는 문제다 (Kadane's Algorithm)

- $mxi = 0$이라고 두자
- 이제 i번째 수를 확인했을 때
  - $mxi = mxi + X_i$
  - 최댓값을 업데이트한다
  - $mxi < 0$이라면 $mxi = 0$

이 알고리즘은 부분 배열의 합이 0 이하로 내려가면 확인하던 구간을 버려버리고, 양수인 구간들만을 유지하면서 체크한다

Maximum Subarray BOJ #10211 – $\mathcal{O}(n)$

```
5  int X[1000];
6  int main() {
7      int t;
8      cin >> t;
9      while (t--) {
10         int n;
11         cin >> n;
12         for (int i = 0; i < n; i++) {
13             cin >> X[i];
14         }
15         int mx = -INF, mxi = 0;
16         for (int i = 0; i < n; i++) {
17             mxi += X[i];
18             mx = max(mx, mxi);
19             if (mxi < 0) mxi = 0;
20         }
21         cout << mx << '\n';
22     }
23     return 0;
24 }
```

Maximum Subarray BOJ #10211 – $\mathcal{O}(n)$

```
5  int main() {
6      int t;
7      cin >> t;
8      while (t--) {
9          int n, x;
10         cin >> n;
11         int mx = -INF, mxi = 0;
12         for (int i = 0; i < n; i++) {
13             cin >> x;
14             mxi += x;
15             mx = max(mx, mxi);
16             if (mxi < 0) mxi = 0;
17         }
18         cout << mx << '\n';
19     }
20     return 0;
21 }
```

최적화 – 이 경우 배열에 먼저 입력받을 필요가 없다

문제 풀어보고, 질문하는 시간 (~17시까지)