목차

1. OT
   - 강의 목표와 목차
   - Prerequisite
   - 코딩 테스트와 대회 준비하기
   - 공지 사항
2. 제발 PS인이면 C++ 씁시다.
3. 시간복잡도
4. 정렬
   - $O\left(N^2\right)$ 정렬 알고리즘
   - $O(N \log N)$ 정렬 알고리즘
5. 이분 탐색

차례

1. OT
   - 강의 목표와 목차
   - Prerequisite
   - 코딩 테스트와 대회 준비하기
   - 공지 사항
2. 제발 PS인이면 C++ 씁시다.
3. 시간복잡도
4. 정렬
   - $O\left(N^2\right)$ 정렬 알고리즘
   - $O(N \log N)$ 정렬 알고리즘
5. 이분 탐색

강의 목표와 목차

스터디가 끝나면 여러분들이

- ✓ 전필 과목(자료구조, 알고리즘)을 아주 쉽게
- ✓ 기업 코딩 테스트 어렵지 않게

해결할 수 있는 수준을 가질 수 있도록 내용 구성을 하였습니다.

강의 목표와 목차

구체적으로,

- ✓ **solved.ac** 기준 **CLASS 5** 문제까지 해결하기 위해 필요한 내용
- ✓ 대기업 코딩테스트 기준 90% 정도의 문제를 해결할 수 있는 내용
- ✓ ICPC/UCPC/SCPC와 같은 대회에 입문하기 위해 필수적으로 알아야하는 내용
- ✓ Codeforces 기준 Div.2의 D번까지 해결할 수 있는 내용

을 다루고자 합니다.

여담으로, 스터디의 내용을 모두 숙지한다면 APC 및 shake!에서 수상권을 기대해볼 수 있습니다.

강의 목표와 목차

주차	주제	1회차 내용	2회차 내용
1	PS 기초	OT, 빠른 입출력, PS를 위한 C++	시간복잡도, 정렬, 이분탐색
2	기초 수학	소수 판별, 소인수 분해, 최대 공약수	기초 정수론 및 조합론
3	완전탐색 및 재귀	브루트 포스, 재귀, 백트래킹	재귀, 백트래킹, 분할 정복
4	자료구조 기초	스택, 큐, 덱	집합과 맵, 해싱, 좌표 압축, 우선순위 큐
5	그래프 이론	그래프의 표현, DFS/BFS 기초	DFS/BFS 심화
6	다이나믹 프로그래밍	DP 개념 : 오토마타, 상태전이, 최적부분구조	DP 응용 : LIS, LCS, Knapsack
7	그리디, 애드 혹	그리디 증명과 응용, 애드 혹	
8	자료구조 심화 및 기타 테크닉	BST, 우선순위 큐(복습), 분리집합	투 포인터, 비트마스킹
9	최단거리 알고리즘	다익스트라, 밸만포드, 플로이드워셜	
10	최종 정리 및 콘테스트	자체 내부 콘테스트	

Prerequisite

- 흥미와 열정
- 질문이 생기면 주저없이 갈기는 결단력
- 질문에 부끄러워하지 않는 용기

Prerequisite

- 수학적 배경, 사고력
- C 언어에 대한 이해
  - 함수 호출, 재귀 함수
  - 다중 반복문, 다차원 배열
  - 을 어느 정도 숙지하고 있는 상태 ~~모두 알거라 말진 않지만~~
- 투자할 시간
  - 많은 문제들을 풀 것입니다.
  - 모든 내용을 이해할 것이라 생각하지 않아요.
  - 반드시 혼자 풀어볼 시간 이 필요합니다.

Prerequisite

아래의 사이트들을 적극 이용합시다.

문제 풀이 (Problem Solving, PS)

- 백준 온라인 저지 (BOJ)
- solved.ac

대회 (Competitive Programming, CP)

- Codeforces
- AtCoder
- solved.ac Arena

코딩 테스트와 대회 준비하기

- 코딩 테스트 $\subset$ 대회 준비
- 코딩 테스트 준비 $\approx$ 대회 기초 준비
- 대회에는 어떤게 있나요?
  - 팀전 ICPC, UCPC ...
  - 개인전 SCPC, 현대모비스 대회, Meta Hackercup, APC, shake! ...

icpc.foundation

International Collegiate Programming Contest(ICPC)

- 세계에서 가장 권위 있는 프로그래밍 대회 입니다.
- 대학 예선 - 지역 본선 - 대륙 본선 - 세계 대회
- 3인 1팀, 3-5시간 동안 8-12문제
- 예선(10월), 본선(11월)

코딩 테스트와 대회 준비하기

코딩 테스트와 대회 준비하기

SAMSUNG COLLEGIATE PROGRAMMING CUP

삼성전자 대학생 프로그래밍 경진대회

Samsung Collegiate Programming Cup(SCPC)

- 대한민국에서 열리는 개인 프로그래밍 경시대회 중 가장 규모가 큽니다.
- 1차 예선 - 2차 예선 - 본선
- 1차 예선(24시간), 2차 예선(12시간), 본선(5시간)
- 7-9월 중 개최됩니다.

shake! '23

- 경인지역 6개 대학 연합 프로그래밍 경시대회 입니다.
- 성균관대, 아주대, 인하대, 항공대, 한양대 ERICA가 참여합니다.
- 참가자와 대회의 규모가 큽니다.
- 각 학교 선발전 후 가을에서 겨울 사이에 개최됩니다.

APC

Ajou University Programming Contest(APC)

- 아주대학교 교내 경시대회입니다.
- 초보자(Div.2)와 중급자(Div.1) 이상을 위한 두 개의 디비젼으로 나뉘어집니다.
- Div.1의 수상자는 shake!의 참여 티켓을 얻습니다.

코딩 테스트와 대회 준비하기

- 이외에도 LG, Meta, Google, 현대모비스 등 많은 기업에서 대회를 개최합니다.
- 또한, 전국 대학생 프로그래밍 동아리 연합(전대프연)에서 매년 아래의 대회들을 개최합니다.
  - UCPC 전대프연 여름대회, 팀전
  - Good Bye/Hello, BOJ! 전대프연 겨울대회, 개인전
- Codeforces, AtCoder, solved.ac Arena는 매주 대회를 개최하고 있습니다.
- **겁 없이 많이 나갈수록 실력이 늡니다.**

Q. 코딩테스트 준비에 대회 준비가 도움이 되나요?

- ✓ 네

Q. 코딩테스트 준비에 대회 준비가 도움이 되나요?

- ~~수학 올림피아드 치던 놈들이 수능 수학을 못볼까~~
- 결국 대회나 코테 모두 정해진 시간, 많은 문제, 빨리 풀기 의 본질을 공유합니다.
- 다른 점은 대회의 범위 뿐입니다. 한편, 이 스터디는 코테의 범위를 다룹니다.
- 고학년의 가장 큰 적은 시간이고, 알고리즘 문제 풀이 능력은 하루 아침에 자라지 않습니다.
- 따라서, 코테/대회는 꾸준히 준비 하는게 맞습니다.

공지 사항

- 스터디 보증금은 5만원 입니다.
  - 납부 계좌 : 12345-45-56789 (우리은행, 신지원)
  - (이름, 학번, 기초)
- 스터디 종료까지 4회 이하 결석 시 전액 환급 해 드립니다.
  - 부득이한 사정의 경우 임원진에게 말씀해주세요.
  - 4회 안에서는 째고 싶을 때 째셔도 좋습니다.
- 원활한 스터디를 위해 그 주에 다룬 내용의 문제들을 풀어보시기 바랍니다.
  - 매주 스터디 중/후 백준 ANSI 그룹에 해당 내용을 다루는 연습 셋이 올라갑니다.
- 강의 중 생기는 질문은 언제든 편하게 해주세요.
  - 강의 중 내용에 관한 질문, 예시 문항에 관련된 질문 및
  - 그냥 혼자 뭐 하다가 몰라서 때려친 문제까지 다 좋습니다.

차례

1. OT
   - 강의 목표와 목차
   - Prerequisite
   - 코딩 테스트와 대회 준비하기
   - 공지 사항
2. 제발 PS인이면 C++ 씁시다.
3. 시간복잡도
4. 정렬
   - $O\left(N^2\right)$ 정렬 알고리즘
   - $O(N \log N)$ 정렬 알고리즘
5. 이분 탐색

C++을 왜 쓸까요?

- ✓ 빠릅니다.
- ✓ standard library가 강력합니다.

`using namespace std;`

- ✓ standard library에 있는 모든 함수와 클래스는 std안에 있습니다.
- ✓ 따라서 `using namespace std;`를 통해 써야할 `std::`를 생략할 수 있습니다.

~~Python은 억까가 좀 있습니다.~~

제발 PS인이면 C++ 씁시다.

`std::cin, std::cout`

- 입출력 시 formating이 필요 없습니다.
- `cin » a; cout « a;`

bits/stdc++.h

- GCC 컴파일러 상에 포함되어 있는 모든 헤더 파일을 포함하는 헤더 파일입니다.
- `#include <bits/stdc++.h>`
- 초보자인 여러분께는 권장하지 않습니다.

각 헤더엔 뭐가 있나요?

- 각 헤더(`algorithm`, `queue`, `string...`마다 모두 정의되어있는 함수와 자료구조들이 있습니다. 차차 알아갑시다.
- `sort(a,a+n)`; `lower_bound(a.begin(), a.end(), x)` 등 같이 알아가봅시다.

std::vector

- 자동으로 메모리가 할당되는 배열
- vector<int> a; 로 선언합니다.
- a.push_back(3); a.pop_back(); a.size(); 등
- 아주 아주 아주 아주 많이 쓰입니다.
- 참고

Fast Input/Output의 필요성

- 개행을 의미하는 `std::endl`의 경우, buffer flush가 일어납니다.
- 실행 시간에 많은 부분을 차지하므로, 이 기능을 끕니다.
- 위 문장들을 입력한 뒤에는 `scanf/cin`과 `printf/cout`을 혼용하면 안됩니다.

사용 방법

```
std::ios_base::sync_with_stdio(false);
std::cin.tie(nullptr);
std::cout.tie(nullptr);
```

PS를 위한 C++ 세팅

같이 한번 실습해 봅시다.

- ✓ `using namespace std;`
- ✓ `#include`
- ✓ `std::cin/cout`
- ✓ `std::vector`
- ✓ Fast IO

차례

1. OT
   - 강의 목표와 목차
   - Prerequisite
   - 코딩 테스트와 대회 준비하기
   - 공지 사항
2. 제발 PS인이면 C++ 씁시다.
3. 시간복잡도
4. 정렬
   - $O\left(N^2\right)$ 정렬 알고리즘
   - $O(N \log N)$ 정렬 알고리즘
5. 이분 탐색

시간복잡도

- 입력의 크기(N)과 알고리즘 실행 시간 간의 관계 함수
- **반복문**이 연산 횟수를 지배합니다.
- 1초 $\approx 10^8$ 번의 연산
- **최악의 연산 횟수**를 계산해야합니다.

Example

- 원소가 N 개인 배열에서 최댓값 찾기 : $T(N) = N$
- 주어진 수 N 의 약수의 갯수 찾기 : $T(N) = \sqrt{N}$

시간 복잡도의 계산

- 가장 중요한 정보만 남깁니다.
- 적당히 큰 수 $x > x_0$ 에 대해서 $f(x) > g(x)$ 라면, $g(x)$ 는 무시 가능

Big-O Notation

- $x > x_0$ 를 만족하는 모든 x 에 대해 $f(x) \leq c \times g(x)$ 라면,

$$f(x) \in O\left(g(x)\right)$$

- 따라서 최고 차항의 계수와 최고 차항을 제외한 부분을 무시합니다.

Example

- ✓ $f(n) = 6n^3 + 3n + 1 \in O(n^3)$
- ✓ $f(n) = 3 \times 4^n + 29 \times 3^n \in O(4^n)$
- ✓ $f(n) = 4n \log n + 4n\sqrt{n} \in O(n\sqrt{n})$
- ✓ $f(n) = n \log n + \dfrac{n^2}{\log n} \in O\left(\dfrac{n^2}{\log n}\right)$

복잡도의 위계

$1 < \log\log N < \log^K N < \sqrt{N} < N < N\log^K N < N\sqrt{N} < N^K < 2^N < N! < N^N$

대략적인 입력 제한

- ✓ $O(N!) : N \approx 10$
- ✓ $O(2^N) : N \approx 20$
- ✓ $O(N^3) : N \approx 500$
- ✓ $O(N^2) : N \approx 5 \times 10^3$
- ✓ $O(N\log N) : N \approx 2 \times 10^5$
- ✓ $O(N) : N \approx 10^6$
- ✓ $O(\log N) : N \approx 10^9 \sim 10^{18}$

차례

1. OT
   - 강의 목표와 목차
   - Prerequisite
   - 코딩 테스트와 대회 준비하기
   - 공지 사항
2. 제발 PS인이면 C++ 씁시다.
3. 시간복잡도
4. 정렬
   - $O\left(N^2\right)$ 정렬 알고리즘
   - $O(N \log N)$ 정렬 알고리즘
5. 이분 탐색

Bubble sort

- 서로 인접한 원소의 대소를 비교하여 정렬합니다.
- $A_j > A_{j+1}$ 이라면, 두 원소를 swap합니다.
- i번째 iteration에서 i번째로 큰 값이 제 위치에 도착합니다.
- $O(N)$ 번의 비교 연산을 $O(N)$ 번 진행하므로 시간복잡도는 $O(N^2)$ 입니다.

```
void bubble_sort() {
    for (int i=1; i<N; i++) {
        for (int j=1; j<N; j++) {
            if (A[j]>A[j+1]) swap(A[j], A[j+1]);
        }
    }
}
```

Selection sort

- 각 단계마다 가장 작은 원소를 맨 앞으로 보내며 정렬합니다.
- i 번째 iteration에서 i 번째로 작은 값이 제 위치에 도착합니다.
- $O(N)$ 번의 비교 연산을 $O(N)$ 번 진행하므로 시간복잡도는 $O(N^2)$ 입니다.

```
void selection_sort() {
    for (int i=1; i<N; i++) {
        int position=i;
        for (int j=i+1; j<=N; j++) {
            if (A[position]>A[j]) position=j;
        }
        swap(A[i], A[position]);
    }
}
```

quick sort, merge sort

- 분할 정복을 이용하여 정렬합니다.
- 이후 분할 정복을 다룰 때 이야기 해봅시다.
- 시간 복잡도는 $O(N \log N)$으로, 가장 강력합니다.

`std::sort`

- 여러 알고리즘을 섞어 사용하고, 가장 효율적이며 빠릅니다.
- `sort(begin iterator, end iterator, compare function)`
  - `vector : sort(a.begin(), a.end());`
  - `Array : sort(a, a+N);`

차례

1. OT
   - 강의 목표와 목차
   - Prerequisite
   - 코딩 테스트와 대회 준비하기
   - 공지 사항
2. 제발 PS인이면 C++ 씁시다.
3. 시간복잡도
4. 정렬
   - $O\left(N^2\right)$ 정렬 알고리즘
   - $O(N \log N)$ 정렬 알고리즘
5. 이분 탐색

제가 마음 속으로 1 부터 10^6 사이의 숫자 x 를 생각할테니 스무고개를 통해 맞혀봅시다. 만약 여러분이 부른 숫자가

- ✓ x 보다 크면 down을,
- ✓ x 보다 작다면 up을,
- ✓ x 와 같다면 answer를 부르겠습니다.

반드시 20번 안에 답을 찾아낼 수 있을까요?

이분 탐색

1. 정답이 반드시 포함되는 구간을 관리한다고 생각해봅시다. 초기엔 $[1, 10^6]$ 이겠네요.
2. 만약 현재 여러분이 들고 있는 구간이 $[l, r]$ 라면, 여러분은 $m = \left\lfloor \frac{l+r}{2} \right\rfloor$ 을 저에게 부릅니다.
3. 두 가지 응답을 들을 수 있습니다.
   - up : x 가 m 보다 크므로, $[l, m]$ 은 절대 정답이 아닙니다.
     따라서 정답의 범위를 $[m+1, r]$ 로 수정합니다.
   - down : x 가 m 보다 작으므로, $[m, r]$ 은 절대 정답이 아닙니다.
     따라서 정답의 범위를 $[l, m-1]$ 로 수정합니다.
4. 정답의 범위는 늘 절반씩 줄어듭니다.
5. 운이 지지리도 없어 정답을 마지막에 알아내도 $\lceil \log_2 10^6 \rceil = 20$ 번 안에 알아낼 수 있습니다.

이분 탐색(Binary Search)

위의 up/down 게임이 정확하게 이분 탐색의 정의입니다.

- 어떤 결정 문제(decision problem)의 답이 단조성을 띨 때,
- 정답을 포함하고 있는 구간의 크기를 절반씩 줄여가며
- 질의(query)의 횟수를 최대 $O(\log N)$ 회로 제한할 수 있습니다.

최적화 문제(Optimization Problem)와 결정 문제(Decision Problem)

- 최적화 문제 : 최댓값/최솟값을 찾아야하는 문제
  - 최대/최소가 되는 해를 구성하시오.
- 결정 문제 : YES/NO로 대답할 수 있는 문제
  - 정답이 X 이하이도록 해를 구성할 수 있는가?
- 최적화 문제는 결정 문제보다 쉬울 수 없습니다.
- 최적화 문제를 풀 수 있다면, 반드시 결정 문제를 풀 수 있기 때문입니다.

최적화 문제(Optimization Problem)를 결정 문제(Decision Problem)로

- ✓ 만약 결정 문제들에 단조성이 있음을 보일 수 있다면,
- ✓ 최적화 문제를 이분 탐색을 통해 풀 수 있음을 장담할 수 있습니다.

배열에서 최댓값 찾기

배열 A에서 최댓값이 10일 때, 최댓값을 찾는 과정을 살펴봅시다. 당연히 모든 원소를 탐색하면 $O(N)$에 해결됩니다.

- ✓ 결정 문제 : "배열의 원소 중에서 x보다 큰 원소가 존재하는가?"
  - $x = 0, 1, \ldots 9$까진 YES 입니다.
  - $x = 10, 11, \ldots$ 부터는 NO 입니다.
  - YY...NNNNN의 형태이므로, 정답에 단조성이 존재합니다.
- ✓ 따라서, (효율적이진 않지만) 이 문제를 이분탐색을 통해 풀 수 있습니다.
- ✓ 시간 복잡도 $O(N \log N)$

lower_bound, upper_bound

lower/upper_bound(begin, end, value)

- 오름차순으로 정렬되어 있는 vector 또는 배열에서
  이분탐색을 통해 value의 iterator를 반환하는 함수입니다.
  - lower : value보다 크거나 같은 값을 처음으로 가지는 iterator를 반환합니다.
  - upper : value보다 큰 값을 처음으로 가지는 iterator를 반환합니다.

Example

```
vector<int> a={1,4,6,6,6,9,12,12,19}
```

- ✓ `lower_bound(a.begin(), a.end(), 4)`
- ✓ `lower_bound(a.begin(), a.begin(),6)`
- ✓ `lower_bound(a.begin(), a.begin(),7)`
- ✓ `upper_bound(a.begin(), a.end(), 21)`
- ✓ `upper_bound(a.begin(), a.end(), 4)`
- ✓ 6의 개수를 구하는 아름다운 방법을 고민해볼까요?

질문하세요!

감사합니다