시간복잡도, Bruteforcing, STL

들어가기 전에: 컴퓨터처럼 생각하기

- Problem Solving은 주어진 문제를 **제한된 메모리와 시간** 안에 해결하는 것
- 그 과정에서 여러 **알고리즘을 활용**해 최적화
- 내가 생각한 풀이가 **해당 문제의 제한**과 맞을지 생각하는 것은 매우 중요
- 무작정 제출했다가는 **시간 초과, 메모리 초과** 등 오답을 받을 수 있음
- 어떻게 나의 코드, 풀이가 **해당 문제의 제한 안에 돈다는 것을 보장**할까?

쉬운 문제

- 1부터 100까지의 정수 중, 서로 다른 99개의 수가 주어진다
- 적히지 않은 하나의 수는 무엇일까?

```
99 95 61 29 25 18 24 32 66 46 65 70 81 93 56
89 20 64 98 38 83 31 86 60 85 68 48 77 88 50
42 43 90 44 13 72 19 8 34 97 51 53 5 3 27 76
74 87 11 58 96 73 1 40 15 54 67 35 55 63 75
100 52 14 92 57 94 10 62 82 23 45 69 33 28
39 26 7 12 78 16 6 47 71 80 21 49 37 36 17
22 59 79 2 9 4 91 30 84
```

쉬운 문제

- 간단한 풀이
- 1이 있나 확인한다, 2가 있나 확인한다, 3이 있나 확인한다, ...
- 이 풀이의 시간복잡도는 얼마나 될까?

```
99 95 61 29 25 18 24 32 66 46 65 70 81 93 56
89 20 64 98 38 83 31 86 60 85 68 48 77 88 50
42 43 90 44 13 72 19 8 34 97 51 53 5 3 27 76
74 87 11 58 96 73 1 40 15 54 67 35 55 63 75
100 52 14 92 57 94 10 62 82 23 45 69 33 28
39 26 7 12 78 16 6 47 71 80 21 49 37 36 17
22 59 79 2 9 4 91 30 84
```

쉬운 문제

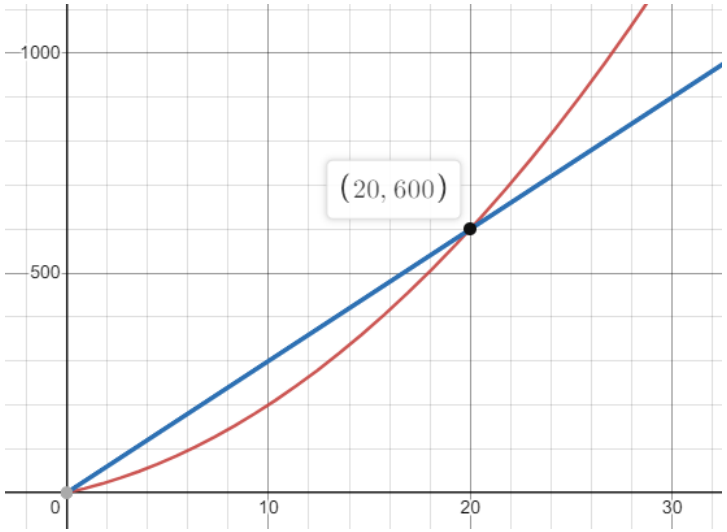
- 최적화된 풀이
- 100개의 체크 배열을 준비해둔 뒤, 배열을 돌면서 해당 수에 대응하는 공간에 체크
- 이 풀이는 배열을 한 번 돌고, 체크 배열에서 체크되지 않은 것을 찾으면 된다
- 배열 한 번 도는 데 N, 체크 배열 확인하는 데 N
- $O(N)$

쉬운 문제

- 수학적인 풀이
- 등차수열의 합 공식을 활용
- 1부터 N까지의 합은 $\frac{N\times(N+1)}{2}$
- 합에서 주어진 배열의 합을 빼면 된다
- 시간복잡도는 똑같이 $O(N)$이지만, 추가적인 배열 순회가 없고 공간 복잡도도 효율적

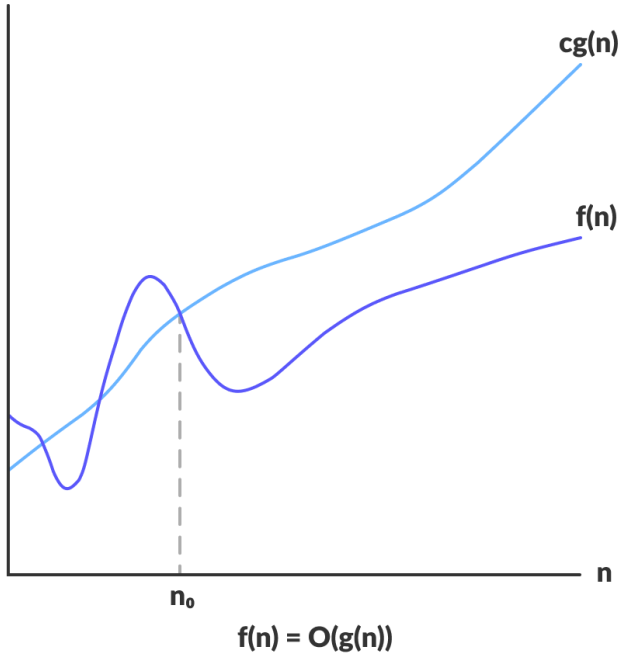
시간 복잡도

- 문제에서 제시된 **입력의 크기**와 **프로그램의 수행 시간**과의 관계를 나타내는 함수
- $f(n) = n^2 + 10n, \quad g(n) = 30n$
- n이 특정 지점부터 $f(n) \geq g(n)$, 해당 수를 n_0이라고 함



시간 복잡도

- $O(g(n)) = \{f(n) : n \geq n_0$인 n에 대해서 $0 \leq f(n) \leq cg(n)$ 를 만족하는 양의 상수 c와 n_0이 존재$\}$
- $f(n) = n^2 + 10n,\ g(n) = n^2$
- $g(n) = 100n$ 일 때에는 만족할까?
- 간단하게 생각하면 최고차항의 차수나, 지수의 밑이 더 중요하다



시간 복잡도

- 시간 복잡도를 계산할 때에는 1초에 1억번 계산할 수 있다고 생각
- N의 범위가 10인 경우에는 $N!, 2^N, N^3, N^2, \ldots$ 등의 시간복잡도 풀이가 통과
- $N \leq 100{,}000$의 경우에는 $NlogN, N$ 등의 시간복잡도 풀이가 통과

블랙잭 BOJ 2798

- 주어진 배열에서 M을 넘지 않으면서 최대한 가까운 세 수의 합을 구하기
- $N \leq 100, M \leq 300\ 000$
- 모든 경우의 수를 확인하는 데 얼마나 많은 시간이 걸릴까?

블랙잭 BOJ 2798

- 주어진 배열에서 M을 넘지 않으면서 최대한 가까운 세 수의 합을 구하기
- $N \leq 100, M \leq 300\ 000$
- 모든 경우의 수를 확인하는 데 얼마나 많은 시간이 걸릴까?
- 카드 세 개를 고정하는 경우의 수와 같음
- $O(N^3)$ 은 1초 안에 돌아간다

블랙잭 BOJ 2798

```cpp
#include <bits/stdc++.h>
using namespace std;
int N, M, arr[101], ans;
int main(){
    cin >> N >> M;
    for(int i = 0; i < N; i++) cin >> arr[i];

    for(int i = 0; i < N; i++){
        for(int j = i+1; j < N; j++){
            for(int k = j+1; k < N; k++){
                int tmp = arr[i] + arr[j] + arr[k];
                if (tmp <= M) ans = max(ans, tmp);
            }
        }
    }
    cout << ans << '\n';
    return 0;
}
```

맞았습니다!!

Bruteforcing

- 앞 문제와 같은 방법으로 모든 경우의 수를 확인해보는 방법을 브루트포스라고 함
- 모든 경우의 수를 따져볼 때에는 해당 경우의 수가 몇 가지인지, 시간 제한에 넉넉히 통과하는지를 검증해봐야 함

- 다른 문제를 보자

일곱 난쟁이 BOJ 2309

- 블랙잭 문제보다 더 어려운 문제도 도전해 보자
- 아홉 난쟁이의 키가 주어졌을 때, 합이 100이 되는 7명의 난쟁이 찾기
- 간단하게 구현하면 어떻게 풀 수 있을까?

일곱 난쟁이 BOJ 2309

```
for(int i = 0; i < N; i++){
    for(int j = i+1; j < N; j++){
        for(int k = j+1; k < N; k++){
            for(int l = k+1; l < N; l++){
               for(int m = l+1; m < N; m++){
                   for(int n = m+1; n < N; n++){
                       for(int o = n+1; o < N; o++){
                           if(arr[i] + arr[j] + arr[k] + arr[l] + arr[m] + arr[n] + arr[o] == 100){
                               cout << arr[i] << '\n' << arr[j] << '\n' << arr[k] << '\n';
                               cout << arr[l] << '\n' << arr[m] << '\n' << arr[n] << '\n' << arr[o] << '\n';
                               return 0;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

일곱 난쟁이 BOJ 2309

- 아무리 가독성을 중요하게 생각하지 않는다고 해도, 내가 읽는 가독성은 중요함
- 아홉 명 중에 일곱 명을 고른다는 것은, **아홉 명 중에 두 명을 고르지 않는 것**과 동치
- 전체 합에서 두 명의 합을 뺐을 때, 100이 되는 경우의 수 찾기

- $O(N^2)$

Bruteforcing

- 주어진 지문을 읽고 바로 떠오르는 풀이법을 시간 복잡도로 나타내자
- 해당 제한을 보고, 제한의 상한을 시간 복잡도에 대입해 보자
- 제한 시간 안에 돌아가는 지 확인한 뒤, 나의 풀이를 코드로 옮기자
- 만약 시간 안에 돌아가지 않는다면, 다른 자료구조나 알고리즘을 적용해 최적화

- 무작정 풀이부터 내고 난 뒤에 시간 초과를 받게 되면 왜 틀렸는지 알 수 없음

분해합 BOJ 2231

- N의 분해합은 N과 N을 이루는 자리수의 합
- 198의 분해합은 $198 + 1 + 9 + 8 = 216$
- 이 때, 198은 216의 생성자라고 한다
- 수가 주어질 때, 그 수의 가장 작은 생성자를 구해 보자
- $N \leq 1\,000\,000$

분해합 BOJ 2231

- 먼저 떠오르는 풀이가 주어진 제한을 만족하는지 생각해 보자
- N이 주어지면, 1부터 분해합을 만든 뒤 N이 되는지 확인
- 생성자 여부를 확인하는 데 최대 $O(7)$, N이 최대 $1\,000\,000$이므로
- $O(N \times \log_{10} N)$

분해합 2 BOJ 12348

- $N \leq 10^{18}$로 제한이 증가했을 때에도, 같은 방법으로 문제를 해결할 수 있을까?
- $O(N)$ 자체로도 시간초과

분해합 2 BOJ 12348

- $N \leq 10^{18}$로 제한이 증가했을 때에도, 같은 방법으로 문제를 해결할 수 있을까?
- 분해합의 생성자는 항상 1부터 탐색해야 할까?
- $100\,000$은 9999에서는 절대 등장할 수 없다, 즉 4자리 숫자에서는 등장할 수 없다
- 나올 수 있는 생성자의 후보의 하한을 구해보자

분해합 2 BOJ 12348

- $N \leq 10^{18}$로 제한이 증가했을 때에도, 같은 방법으로 문제를 해결할 수 있을까?
- $N - 9 \times K$, K는 자릿수의 개수
- 찾아야 하는 수의 후보가 최대 $O(9 \times 18)$로 $O(1)$에 탐색할 수 있다

Standard Template Library (STL)

- C++에서 제공하는 표준 템플릿 라이브러리
- 컨테이너: `pair`,`vector`,`tuple`,`deque`,`queue`,`priority_queue`,`stack`,`set`,`map`
- 알고리즘: `sort`,`lower_bound`,`merge`, …
- Iterator: `begin`,`end`, …

- Template 자료형 사용으로 호환이 장점

pair<T1, T2>

- 두 개의 자료형을 담는 컨테이너 (2-tuple)
- 가져올 때에는 first, second로 가져온다

```
pair<int, int> p1 = {1, 5};
pair<int, int> p2 = make_pair(1, 5);
auto p3 = make_pair(1, 5);
cout << p1.first << " " << p2.second << '\n';
```

tuple<...T>

- 세 개 이상의 자료형을 담는 컨테이너 (n-tuple)
- 가져올 때에는 get<index>(tuple)로 가져온다

```
tuple<int, int, string> t = {1, 2, "123"};
cout << get<1>(t) << '\n';

vector<tuple<string, int, int>> v;
v.emplace_back("abc", 100, 123);
```

vector<T>

- 여러 개의 자료를 담는 가변 길이의 컨테이너
- push_back을 통해 배열의 끝에 값을 추가할 수 있다
- emplace_back을 통해 해당 객체의 생성자를 부를 수 있다
- 삽입 시간복잡도는 amortized $O(1)$

```cpp
vector<int> v;
v.push_back(10);

vector<pair<int, int>> points;
points.push_back(make_pair(2, 3));
points.push_back({2, 3})
points.emplace_back(2, 3);
```

iterator

- 컨테이너의 위치를 나타내는 클래스, 포인터와 같이 동작한다
- begin, end는 각각 시작 위치, 끝 뒤 위치를 가리킨다
- 포인터처럼 참조연산이 가능하다

```
vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
cout << *v.begin() << '\n';
cout << *(v.end() - 1) << '\n';
```

sort

- [first, last) 범위를 비교 기준에 따라 정렬
- 기본적인 비교 함수는 **operator<**
- 비교 함수를 원하는 대로 작성할 수 있다
- sort(시작 주소, 끝 주소, 비교 함수)

```
vector<int> v;
v.push_back(4);
v.push_back(1);
v.push_back(2);
sort(v.begin(), v.end(), cmp);
```

sort

- [first, last) 범위를 **비교 기준**에 따라 정렬
- 두 수를 서로 비교할 수 있어야 정렬이 가능함, 어떤 것이 앞으로 올지 결정해야 함
- 비교함수 `bool compare(T a, T b)`
- a가 b보다 반드시 앞에 와야한다면 true, 아니면 false
- **Strict weak ordering**을 만족해야 함

Strict weak ordering

- **비반사성**: 모든 x에 대해 $R(x, x)$는 거짓
- **비대칭성**: 모든 x, y에 대해 $R(x, y)$가 참이면 $R(y, x)$는 거짓
- **추이성**: 모든 x, y, z에 대해서 $R(x, y)$와 $R(y, z)$가 참이면 $R(x, z)$는 참
- **비비교성의 추이성**: 모든 x, y, z에 대해 $R(x, y)$와 $R(y, x)$가 거짓이고 $R(y, z)$와 $R(z, y)$가 거짓이면 $R(x, z)$와 $R(z, x)$는 거짓 (비교할 수 없는 것의 추이성)
- 위 네가지 중 하나라도 만족하지 않는다면 **두 원소의 순서를 정할 수 없는** 상황 발생

나이순 정렬 BOJ 10814

- 간단한 정렬 문제
- 나이가 증가하는 순으로, 나이가 같으면 먼저 가입한 사람이 앞에 오도록 정렬

나이순 정렬 BOJ 10814

- 간단한 정렬 문제
- 나이가 증가하는 순으로, 나이가 같으면 먼저 가입한 사람이 앞에 오도록 정렬
- `bool compare(T a, T b)`에서, a가 앞에 오려면 true를 반환함에 유의하고 비교함수 작성
- C++에서의 정렬은 introsort를 사용 (quick + heap + insertion)
- $O(NlogN)$

나이순 정렬 BOJ 10814

```cpp
vector<tuple<int, int, string>> v;
bool cmp(tuple<int, int, string>& t1, tuple<int, int, string>& t2){
    if (get<1>(t1) != get<1>(t2)){
        return get<1>(t1) < get<1>(t2);
    }
    return get<0>(t1) < get<0>(t2);
}
```

sort

- 정렬하는 데 드는 시간도 존재, 왜 정렬해야 할까?
- 시간복잡도의 정의에서 봤듯이, 정렬하는 데 드는 시간복잡도보다 낮은 차수는 무시됨
- 정렬한 뒤에, 이를 이용해서 빠르게 계산할 수 있는 여러 테크닉이 존재

- 이분 탐색, 그리디, 그래프에서의 위상 정렬, ...
- 나중에 배워보자!

빠른 입출력 BOJ 15552

- **15552: 빠른 A+B**를 풀어보자

입력

첫 줄에 테스트케이스의 개수 T가 주어진다. T는 최대 1,000,000이다. 다음 T줄에는 각각 두 정수 A와 B가 주어진다. A와 B는 1 이상, 1,000 이하이다.

출력

각 테스트케이스마다 A+B를 한 줄에 하나씩 순서대로 출력한다.

예제 입력 1 복사

```
5
1 1
12 34
5 500
40 60
1000 1000
```

예제 출력 1 복사

```
2
46
505
100
2000
```

빠른 입출력 BOJ 15552

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
    int a, b, t;
    cin >> t;
    for(int i = 0; i < t; i++){
        cin >> a >> b;
        cout << a + b << endl;
    }
    return 0;
}
```

빠른 입출력 BOJ 15552

- 왜 시간초과가 발생할까?
- 시간복잡도에 문제가 있을까?

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
    int a, b, t;
    cin >> t;
    for(int i = 0; i < t; i++){
        cin >> a >> b;
        cout << a + b << endl;
    }
    return 0;
}
```

제출 번호	아이디	문제	문제 제목	결과
66106766	aru0504	15552	빠른 A+B	시간 초과

빠른 입출력

- 프로그램의 입출력 시간은 수행 시간에 포함됨
- 입출력은 굉장히 오래 걸리는 작업
- 특히 endl의 경우, 입력 버퍼를 비우는 작업을 추가적으로 시행
- 입/출력이 방대한 경우, 그 시간만으로 제한 시간을 초과할 수 있다

빠른 입출력

- 프로그램의 입출력 싱크를 끊고, 한 번에 출력하도록 설정
- 아래 세 줄을 main 바로 아래에 추가
- 이 경우, cin, cout만을 사용하고 scanf, printf와 혼용 금지
- endl는 버퍼를 비우므로 개행문자('\n')를 사용하자
- 입출력 양이 10,000줄이 넘어가면 사용을 고려하자

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

빠른 입출력

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios::sync_with_stdio(false);
    int a, b, t;
    cin >> t;
    for(int i = 0; i < t; i++){
        cin >> a >> b;
        cout << a + b << '\n';
    }
    return 0;
}
```

제출 번호	아이디	문제	문제 제목	결과
66107037	aru0504	15552	빠른 A+B	맞았습니다!!

빠른 입출력

- 파이썬을 사용하는 경우, 아래와 같이 사용하자
- readline의 경우 개행문자까지 받아온다
- 문자열을 받고 싶다면 rstrip을 통해 개행문자를 제거하자

```
import sys
s = sys.stdin.readline().rstrip()
```

Reference

- https://github.com/justiceHui/SSU-SCCC-Study/blob/master/2023-summer-basic/slide/04-3-cpp-sort.pdf