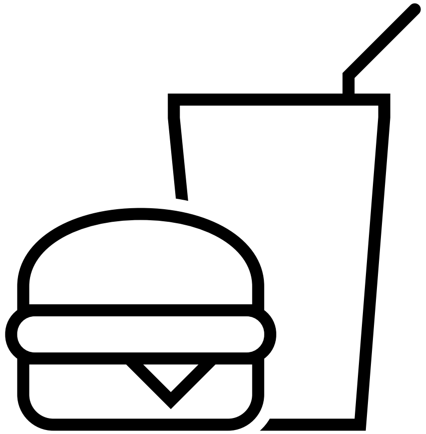
알고리즘 문제해결

Sejong Algorithm Lab

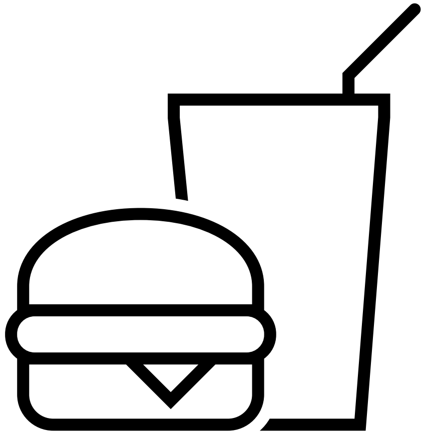
알고리즘의 성능을 판단하는 척도

- 얼마나 **정확한 답**을 구할 수 있는가?
- 얼마나 **적은 연산**을 필요로 하는가?
- 얼마나 **적은 공간**을 필요로 하는가?

관점의 차이

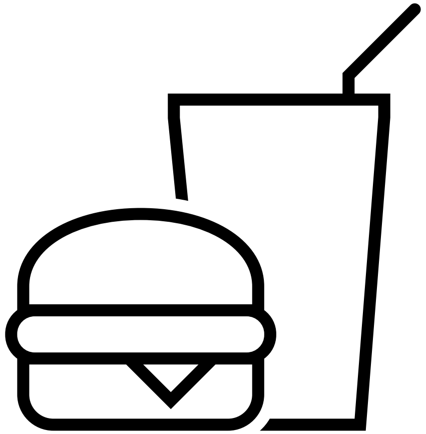


식당 A
99.99% 확률로 최고의 식사를 제공
0.01% 확률로 최악의 식사를 제공

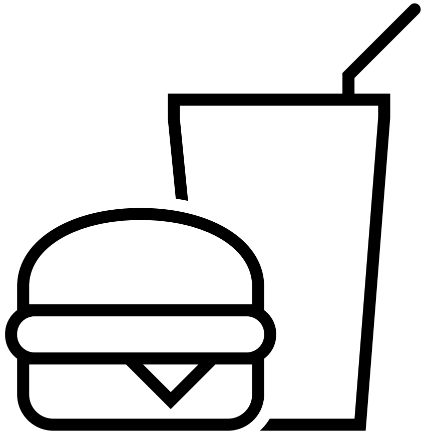


식당 B
100% 확률로 평이한 식사를 제공

알고리즘 문제해결에서는 최악의 경우를 고려



<



식당 A

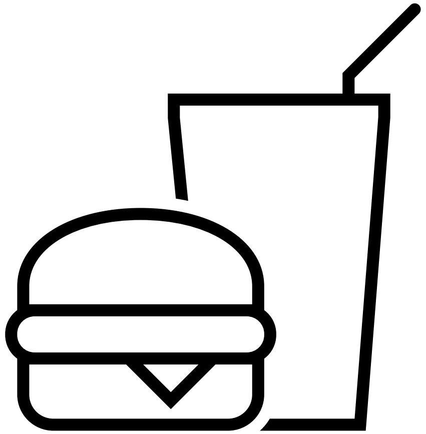
99.99% 확률로 최고의 식사를 제공

0.01% 확률로 최악의 식사를 제공

식당 B

100% 확률로 평이한 식사를 제공

가장 이상적인 알고리즘



식당 C

100% 확률로 최고의 식사를 제공

시간복잡도

들어가기 전에...

29	63	6	40	51	93	9	43	53	28
90	59	72	88	61	47	65	2	96	62
31	83	20	78	45	42	85	87	76	57
18	77	32	10	99	1	3	14	52	100
66	71	49	55	68	74	97	4	19	34
75	24	7	64	33	81	5	58	17	79
36	26	82	80	86	37	8	21	70	46
23	15	60	44	35	98	56	92	95	89
16	50	39	25	11	48	67	94	91	73
13	84	41	12	22	30	27	54	69	

시간복잡도

- 얼마나 **정확한 답**을 구할 수 있는가?
- 얼마나 **적은 연산**을 필요로 하는가?
- 얼마나 **적은 공간**을 필요로 하는가?

시간복잡도의 정의

- 프로그램이 문제를 해결하는 데에 걸리는 시간을 입력 크기를 나타내는 변수(들)로 나타낸 함수

걸리는 시간은 컴퓨터의 성능에 따라 다름

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, ans = 0; cin >> n;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) ans++;
    }
    cout << ans << '\n';
    return 0;
}
```

n	걸린 시간
10000000	0.019초
100000000	0.137초
1000000000	1.326초

내 컴퓨터

n	걸린 시간
10000000	0.006초
100000000	0.043초
1000000000	0.324초

친구 컴퓨터

걸리는 시간은 실행할 때마다도 다름

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, ans = 0; cin >> n;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) ans++;
    }
    cout << ans << '\n';
    return 0;
}
```

	걸린 시간
1차 실행	1.370초
2차 실행	1.332초
3차 실행	1.341초

$n = 1000000000$

시간복잡도를 일관적으로 나타내기 위한 방법

- 걸리는 시간이 일정하다고 기대되는 연산들을 **기본연산**으로 정의
- 시간복잡도는 기본연산의 **수행 횟수**

기본연산의 예시

- 대입 연산
- 비교 연산
- 사칙 연산
- 배열의 임의의 인덱스에 접근

Big O notation

- 시간복잡도를 나타내기 위해 가장 많이 사용하는 방법
- 알고리즘의 성능을 대략적으로 나타내기 위해 사용
- 보통은 가장 큰 영향을 끼치는 항만 표시

Big O notation 정의

- 모든 실수 $x > x_0$에 대하여 $|f(x)| \le c|g(x)|$를 만족하는 실수 x_0와 양의 실수 c가 존재한다면, $f(x) \in O(g(x))$로 표기함

- $f(x) = O(g(x))$와 같이 등호로도 나타낼 수 있음

Big O notation 예시

- $f(x) = 3x + 17$, $g(x) = x$

- $c = 4$, $x_0 = 17$로 두면 모든 실수 $x > x_0$에 대하여 $|f(x)| \leq c|g(x)|$

즉, 모든 실수 $x > 17$에 대하여 $3x + 17 \leq 4x$임을 알 수 있음

따라서 $f(x) = 3x + 17 \in O(x)$

Big O notation 예시

- $T(n) = 3 \rightarrow O(1)$
- $T(n) = 2n + 1000000 \rightarrow O(n)$
- $T(n) = 1000n^2 + 1000000n + 99 \rightarrow O(n^2)$
- $T(n) = 2n + 7n \log n \rightarrow O(n \log n)$
- $T(n) = 2^n + 3^n \rightarrow O(3^n)$
- $T(n) = n! + 100^n \rightarrow O(n!)$
- $T(n, m) = nm + \log n \rightarrow O(nm)$
- $T(n, m) = nm + m^2 \log n \rightarrow O(nm + m^2 \log n)$

시간복잡도 구하기

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, ans = 0; cin >> n;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) ans++;
    }
    cout << ans << '\n';
    return 0;
}
```

코드 1

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, ans = 1; cin >> n;
    for (int i = 1; i * 2 <= n; i++) {
        if (n % i == 0) ans++;
    }
    cout << ans << '\n';
    return 0;
}
```

코드 2

더 좋은 알고리즘은?

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, ans = 0; cin >> n;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) ans++;
    }
    cout << ans << '\n';
    return 0;
}
```

$O(n)$

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, ans = 1; cin >> n;
    for (int i = 1; i * 2 <= n; i++) {
        if (n % i == 0) ans++;
    }
    cout << ans << '\n';
    return 0;
}
```

$O(n)$

Big O notation의 한계

- 실제 문제에서는 입력의 크기가 무한히 크지는 않음
- 보통 1초에 1억개의 기본연산을 할 수 있다고 생각하면 편함

알고리즘 수업 - 알고리즘의 수행 시간 1 BOJ 24262

```
MenOfPassion(A[], n) {
    i = ⌊n / 2⌋;
    return A[i]; # 코드1
}
```

- n의 크기와 관계 없이 코드1은 한 번 수행됨
- 시간복잡도 $T(n) = 1$
- Big O notation으로 나타내면 $O(1)$
- 최고차항의 차수는 0

알고리즘 수업 - 알고리즘의 수행 시간 2 BOJ 24263

```
MenOfPassion(A[], n) {
    sum <- 0;
    for i <- 1 to n
        sum <- sum + A[i]; # 코드1
    return sum;
}
```

- 시간복잡도 $T(n) = n$
- Big O notation으로 나타내면 $O(n)$
- 최고차항의 차수는 1

알고리즘 수업 - 알고리즘의 수행 시간 3 BOJ 24264

```
MenOfPassion(A[], n) {
    sum <- 0;
    for i <- 1 to n
        for j <- 1 to n
            sum <- sum + A[i] × A[j]; # 코드1
    return sum;
}
```

- 시간복잡도 $T(n) = n^2$
- Big O notation으로 나타내면 $O(n^2)$
- 최고차항의 차수는 2

알고리즘 수업 - 알고리즘의 수행 시간 4 BOJ 24265

```
MenOfPassion(A[], n) {
    sum <- 0;
    for i <- 1 to n - 1
        for j <- i + 1 to n
            sum <- sum + A[i] × A[j]; # 코드1
    return sum;
}
```

- 시간복잡도 $T(n) = \frac{n(n-1)}{2}$
- Big O notation으로 나타내면 $O(n^2)$
- 최고차항의 차수는 2

대표적인 시간복잡도

- $O(1)$ – 입력의 크기와 관계 없이 항상 상수시간을 유지하는 알고리즘
- $O(\log n)$ – 이분탐색, 최대공약수 구하기
- $O(\sqrt{n})$ – 소수 판정
- $O(n)$ – 선형시간 알고리즘
- $O(n \log n)$ – 가장 빠른 비교 정렬
- $O(n\sqrt{n})$ – 제곱근 분할법
- $O(n^2)$ – 선택 정렬, 버블 정렬, 삽입 정렬
- $O(n^3)$ – 행렬 곱셈
- $O(2^n)$ – 백트래킹, 비트마스킹
- $O(n!)$ – 외판원 문제

마무리하며...

- 시간복잡도는 사실 이론보다는 직접 문제를 풀면서 체감하는 것이 가장 중요

STL

많은 것들이 이미 STL에 잘 구현되어 있음

- 우리는 이미 잘 구현된 것들을 문제를 푸는데 활용하면 됨

가장 많이 사용되는 한가지만 소개

- 나머지는 해당되는 부분에서 소개

std::vector

- 쉽게 말하면 크기가 변할 수 있는 배열

- `void push_back(const T& val);`
- `void pop_back();`
- `T& front();`
- `T& back();`
- `size_t size() const;`
- `T& operator[] (size_t n);`

std::vector

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v;
    v.push_back(1);
    v.push_back(2);
    cout << v.front() << '\n';
    cout << v.back() << '\n';
    v.push_back(v[1]);
    cout << v.back() << '\n';
    cout << v.size() << '\n';
    v.pop_back();
    cout << v.size() << '\n';
}
```

실행결과

```
1
2
2
3
2
```

브루트포스

일곱 난쟁이 BOJ 2309

- 9명의 키가 주어졌을 때, 합이 100이 되는 7명을 찾는 문제
- 7명을 고르는 모든 경우를 다 확인해보면 됨
- 시간복잡도 $T \approx \binom{9}{7} = 36$

일곱 난쟁이 BOJ 2309

```cpp
#include <bits/stdc++.h>
using namespace std;
int arr[9];
int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    for (int i = 0; i < 9; i++) cin >> arr[i];
    sort(arr, arr + 9);
    for (int i = 0; i < 9; i++) {
        for (int j = i + 1; j < 9; j++) {
            for (int k = j + 1; k < 9; k++) {
                for (int l = k + 1; l < 9; l++) {
                    for (int m = l + 1; m < 9; m++) {
                        for (int n = m + 1; n < 9; n++) {
                            for (int o = n + 1; o < 9; o++) {
                                if (arr[i] + arr[j] + arr[k] + arr[l] + arr[m] + arr[n] + arr[o] == 100) {
                                    cout << arr[i] << '\n';
                                    cout << arr[j] << '\n';
                                    cout << arr[k] << '\n';
                                    cout << arr[l] << '\n';
                                    cout << arr[m] << '\n';
                                    cout << arr[n] << '\n';
                                    cout << arr[o] << '\n';
                                    return 0;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

일곱 난쟁이 BOJ 2309

- 코드 가독성이 너무 떨어짐
- 9명 중 7명을 고르는 것은 **9명 중 2명을 고르지 않는 것**과 같음
- 9명의 키에서 2명의 키를 뺐을 때 100이 되는 2명 찾기

- 시간복잡도 $T \approx \binom{9}{2} = 36$

브루트포스

- 위의 예시처럼 가능한 모든 경우의 수를 확인해보는 방법을 **브루트포스**라고 함

문제해결의 사고 과정

- 문제를 읽고 바로 떠오르는 방법의 시간복잡도를 계산
- 계산한 시간복잡도가 제한 시간 안에 들어오는지 확인

- 보통 1초 ≈ 1억개의 기본연산으로 생각하면 편함

- 만약 제한 시간 안에 들어오지 않는다면 다른 풀이 방법을 생각

연습문제

1145 : 적어도 대부분의 배수
1436 : 영화감독 숌
2231 : 분해합
2798 : 블랙잭
12348 : 분해합 2
19532 : 수학은 비대면강의입니다
25793 : 초콜릿 피라미드
30446 : 회문수

정렬

정렬이란?

- 원소들을 특정 기준에 따라 순서대로 나열하는 것

단어들을 사전순으로 나열

사람들을 도착한 시간순으로 나열

정렬을 하는 이유

- 원하는 데이터를 효율적으로 탐색하기 위함

원하는 단어를 효율적으로 탐색

가장 먼저 도착한 사람을 효율적으로 탐색

여러가지 정렬 알고리즘

- 선택 정렬 – $O(n^2)$
- 삽입 정렬 – $O(n^2)$
- 버블 정렬 – $O(n^2)$
- 병합 정렬 – $O(n \log n)$
- 퀵 정렬

- 이외에도 매우 많은 정렬 알고리즘들이 존재
- 정렬 알고리즘들의 자세한 작동 방법은 생략

std::sort

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
void print(vector<int> &arr) {
    for (auto &e: arr) cout << e << ' ';
    cout << '\n';
}
int arr[10] = {1, 9, 5, 4, 7, 6, 8, 2, 3, 10};
int main() {
    vector<int> vec = {8, 4, 5, 6, 9, 7, 10, 1, 3, 2};
    print(vec);
    sort(vec.begin(), vec.end());
    print(vec);
    for (int i = 0; i < 10; i++) cout << arr[i] << ' ';
    cout << '\n';
    sort(arr, arr + 10);
    for (int i = 0; i < 10; i++) cout << arr[i] << ' ';
    cout << '\n';
    return 0;
}
```

실행결과

```
8 4 5 6 9 7 10 1 3 2
1 2 3 4 5 6 7 8 9 10
1 9 5 4 7 6 8 2 3 10
1 2 3 4 5 6 7 8 9 10
```

수 정렬하기 2 BOJ 2751

- $O(n \log n)$ 정렬 알고리즘을 직접 구현해서 풀어도 됨
- 이미 잘 구현된 함수 사용을 적극 추천

```cpp
#include <bits/stdc++.h>
using namespace std;
int arr[1000000];
int main() {
    int n; cin >> n;
    for (int i = 0; i < n; i++) cin >> arr[i];
    sort(arr, arr + n);
    for (int i = 0; i < n; i++) cout << arr[i] << '\n';
    return 0;
}
```

C++

```python
import sys
def input():
    return sys.stdin.readline().rstrip()
n = int(input())
arr = []
for _ in range(n):
    arr.append(int(input()))
arr.sort()
for e in arr:
    print(e)
```

Python

좌표 정렬하기 2 BOJ 11651

- 비교 정렬에서는 비교함수가 중요

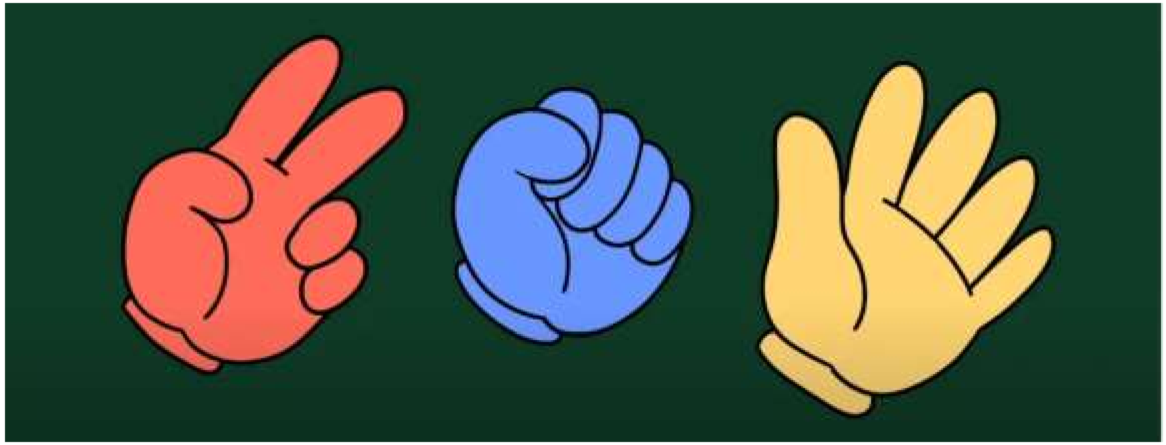
```cpp
#include <bits/stdc++.h>
using namespace std;
struct dot {
    int x, y;
};
dot arr[100000];
bool comp(const dot& a, const dot& b) {
    if (a.y != b.y) return a.y < b.y;
    return a.x < b.x;
}
```

```cpp
int main() {
    int n; cin >> n;
    for (int i = 0; i < n; i++)
        cin >> arr[i].x >> arr[i].y;
    sort(arr, arr + n, comp);
    for (int i = 0; i < n; i++)
        cout << arr[i].x << ' ' << arr[i].y << '\n';
    return 0;
}
```

다음 기준들로 아래 데이터들을 정렬해봅시다.

	시험 1	시험 2
학생 A	60	80
학생 B	80	90
학생 C	90	70
학생 D	70	80
학생 E	60	70
학생 F	80	90
학생 G	100	100
학생 H	50	50

기준: 두 시험 모두 점수가 높은 학생이 위로 오도록 정렬



기준: 가위바위보 게임에서 이기는 모양이 왼쪽에 오도록 정렬

Strict weak ordering

정렬 알고리즘의 비교 기준은 다음 4가지 조건을 모두 만족해야 함

- **비반사성**: 모든 x에 대하여 $x < x$는 거짓
- **비대칭성**: 모든 x, y에 대하여 $x < y$가 참이면 $y < x$는 거짓
- **추이성**: 모든 x, y, z에 대하여 $x < y$와 $y < z$가 모두 참이면 $x < z$는 참
- **비비교성의 추이성**: 모든 x, y, z에 대하여 $x < y$와 $y < x$가 거짓이고 $y < z$와 $z < y$가 거짓이면 $x < z$와 $z < x$는 모두 거짓

std::정렬부터 시작하는 디버깅 생활 BOJ 13316

지구이의 코드

```
#include<stdio.h>
#include<algorithm>
#include<vector>
#include<stdlib.h>
#include<cassert>

using namespace std;

typedef pair<int,int> pii;

int main()
{
    int N, a, b;
    assert(scanf("%d", &N) == 1);
    assert(2 <= N && N <= 1000);

    vector<pii> G;
    for(int i = 1; i <= N; i++){
        assert(scanf("%d%d", &a, &b) == 2);
        assert(0 <= a && a <= 1000 && 0 <= b && b <= 1000);
        G.push_back(pii(a, b));
    }
    sort(G.begin(), G.end(), [](const pii &l, const pii &r){ return l.first * r.second < l.second * r.first; });

    bool ch = false;
    for(int i = 0; i < N; i++){
        for(int j = i+1; j < N; j++){
            ch |= G[i].first * G[j].second > G[i].second * G[j].first;
        }
    }
    assert(ch);
    return 0;
}
```

가장 빠른 비교 정렬의 시간복잡도

- $O(n \log n)$보다 빠른 비교 정렬 알고리즘은 존재하지 않음

마무리하며...

- Strict weak ordering이 이해가 잘 안 된다고 걱정할 필요는 전혀 없음

연습문제

1181 : 단어 정렬

14674 : STOP USING MONEY

10814 : 나이순 정렬