<< 알고리즘 >>

시간 복잡도 : 어떤 코드를 작성할 때 시간이 얼마나 걸리는 지를 예측해보는 방법

➔ 표기법으로 O를 사용, 최악의 경우에 시간이 얼마나 걸릴지 알 수 있음.

코드작성 전에 시간 복잡도를 계산해보고 시간 안에 수행될 것 같은 코드를 짜는 것이 좋다.

계산시에는 상수를 버림. 변수가 같으면 큰 것만 빼고 다 버림. 두항에서 변수가 다르면 냅둠

메모리 : 보통은 메모리가 넉넉해서 걱정할 필요가 없음. 배열을 너무 많이 사용시 시간초과.

➔ 메모리 초과를 받는 경우는 불필요한 공간을 할당 받았을 때.

입출력

➔ C : scanf, printf

➔ C++ : scanf/printf , cin/cout, cin/cout이 더 느림. 입출력이 많을 때 불리.

cin/cout에도 ios_base :: sync_with_stdio(false);

cin.tie(NULL);

cin.tie(NULL)

위 코드 추가시 scanf/printf 만큼 빨라짐.

➔ Java : 입력은 Scanner, 출력은 System.out을 사용

입력이 많을 땐 BufferReader을 사용, 출력이 많을 땐 StringBuilder

입력시에 테스트 케이스가 나오는 문제 – 입력 받아서 처리 후 바로 출력해주는게 효율적

입력을 EOF까지 받는 것 – While, Scanner등 각 언어에 맞게

<< 스택 >>

: 한쪽 끝에서만 자료를 넣고 뺄 수 있는 자료구조. 맨 위 값만 알 수 있음.

배열 하나로 구현 가능. 맨 위 값이 의미가 있을 경우 사용. 후입선출

- Push : 스택에 자료를 넣는 연산
- Pop : 스택에서 맨 위 자료를 빼는 연산
- Top : 스택의 가장 위에 있는 자료를 보는 연산
- Empty : 스택이 비어있는지 알아보는 연산

- Size : 스택에 저장된 자료 개수를 보는 연산

<< 큐 >>

: 한쪽 끝에서만 자료를 넣고 다른 한쪽 끝에서만 뺄 수 있는 자료구조.

먼저 넣은게 가장 먼저 나옴. 배열 하나로 구현 가능. 선입선출 ex) 줄서기

- Push : 큐에 자료를 넣는 연산
- Pop : 큐에서 자료를 빼는 연산, 제일 앞에 있는 거
- Front : 큐의 가장 앞에 있는 자료를 보는 연산
- Back : 큐의 가장 뒤에 있는 자료를 보는 연산
- Empty : 큐가 비어있는지 알아보는 연산
- Size : 큐에 저장된 자료 개수를 알아보는 연산

<< 덱 >>

: 양 끝에서만 자료를 넣고 양 끝에서 뺄 수 있는 자료구조

- Push_front : 덱의 앞에 자료를 넣는 연산
- Push_back : 덱의 뒤에 자료를 넣는 연산
- Pop_front : 덱의 앞에서 자료를 빼는 연산
- Pop_back : 덱의 뒤에서 자료를 빼는 연산
- Front : 덱의 가장 앞에 있는 자료를 보는 연산
- Back : 덱의 가장 뒤에 있는 자료를 보는 연산