ICPC Sinchon

02. Sorting, Linear data structure

02. 정렬, 선형 자료 구조

소개

박찬솔 (chansol)

학교

- 숭실대학교 컴퓨터학부 (2022.03 ~)
- 숭실대학교 데이터베이스 연구실 학부연구생 (2022.03 ~)

대회 참가/수상

- 2022/23 ICPC World Finalist
- 2022 ICPC Asia Seoul Regional 5th place
- 제38회 한국정보올림피아드 고등부 2차 대회 장려상

2023 Summer Algorithm Camp

목차

1. 정렬이란
2. 버블 정렬, 삽입 정렬, 퀵 정렬, 병합 정렬
3. STL sort (비교 함수의 조건)
4. 배열, 연결 리스트
5. 스택, 큐, 덱
6. 예시 문제

정렬

- 원하는 조건에 맞게 데이터를 다시 배열하는 것
- 대표적인 정렬 방식 : 오름차순, 내림차순

- 오름차순 : a b c … , $a \leq b \leq c \leq \cdots$
- 내림차순 : a b c … , $a \geq b \geq c \geq \cdots$

버블 정렬

- 인접한 두 원소를 순서대로 보면서 정렬해 나가는 알고리즘

- 오름차순으로 정렬한다고 할 때,
- A[i]와 A[i + 1]을 비교하자.
- A[i] > A[i + 1]이면 A[i]와 A[i + 1]을 교환(swap)한다.

- 이 과정을 i = 1.. N - 1까지 순서대로 한 번 수행하는 것을 **순회**라고 하자. (N은 배열의 길이)
- 버블 정렬은 순회를 N - 1번 반복한다.

버블 정렬

- [4, 5, 2, 3, 1]을 정렬한다고 해보자.

- i = 1
- [4, 5, 2, 3, 1]

- i = 2
- [4, 2, 5, 3, 1]

- i = 3
- [4, 2, 3, 5, 1]

- i = 4
- [4, 2, 3, 1, 5]

- 첫 번째 과정을 수행했을 때, 가장 큰 수 5가 맨 뒤에 위치한다.

버블 정렬

- 계속해서 [4, 2, 3, 1, 5]

- i = 1
- [2, 4, 3, 1, 5]

- i = 2
- [2, 3, 4, 1, 5]

- i = 3
- [2, 3, 1, 4, 5]

- 두 번째 과정을 수행했을 때, 두 번째로 큰 수 4가 맨 뒤에서 두 번째에 위치한다.

버블 정렬

- 계속해서 [2, 3, 1, 4, 5]

- i = 1
- [2, 3, 1, 4, 5]

- i = 2
- [2, 1, 3, 4, 5]

- 세 번째 과정을 수행했을 때, 세 번째로 큰 수 3가 맨 뒤에서 세 번째에 위치한다.

- 마지막으로 [2, 1, 3, 4, 5]

- i = 1
- [1, 2, 3, 4, 5]

- 정렬 끝.

2023 Summer Algorithm Camp

버블 정렬 - 시간 복잡도

- 길이가 N인 배열을 한 번 순회할 때,
- 비교 N - 1번 (최대)
- 교환 N - 1번 (최대)

- i번째 순회에서 i번째로 큰 값이 뒤에서 i번째 위치로 이동한다.
- 순회를 N - 1번 반복하면 모든 수가 올바른 위치로 이동한다.

- $(N - 1)^2$번 연산을 수행하므로 시간 복잡도는 $O(N^2)$

질문?

삽입 정렬

- 적절한 위치에 원소를 옮김(삽입함)으로써 정렬해 나가는 알고리즘

- i = 2 .. N인 i에 대해서 순서대로 다음 과정을 수행한다.

- i번째 작업에서:
- A[i]를 부분 배열 [1, i]가 정렬된 상태가 되도록 적절한 위치에 삽입한다.

삽입 정렬

- [4, 5, 2, 3, 1]을 정렬한다고 해보자.

- i = 2
- [4, 5, 2, 3, 1]

- i = 3
- [2, 4, 5, 3, 1]

- i = 4
- [2, 3, 4, 5, 1]

- i = 5
- [1, 2, 3, 4, 5]

- i번째 과정을 수행하면, 부분 배열 [1, i]는 정렬된 상태이다.

2023 Summer Algorithm Camp

삽입 정렬 - 정당성 증명

- 수학적 귀납법

- i = 2 (수행 전) : [1, 1]은 길이가 1인 배열이므로 정렬된 상태이다.
- i = 2 (수행 후) : A[2]를 적절한 위치에 넣었으므로 [1, 2]는 정렬된 상태이다.

- i > 2인 모든 i에 대해서
- i번째 과정을 수행하기 전, [1, i - 1]은 정렬된 상태이다.
- i번째 과정을 수행한 후, A[i]를 적절한 위치에 넣은 후인 [1, i]는 정렬된 상태이다.

- i = N, N번째 과정을 수행하면 [1, N]은 정렬된 상태이다. 증명 끝.

2023 Summer Algorithm Camp

삽입 정렬 - 시간 복잡도

- (best)이미 정렬된 배열인 경우:
- i번째 작업에서 A[i]를 이동할 필요 없이 그대로 i번째 위치에 둔다.
- 매 작업에 O(1)이 걸리므로, O(N)

- (worst)반대로 정렬된 배열인 경우:(ex. 5 4 3 2 1 을 1 2 3 4 5로 정렬하기)
- i번째 작업에서 A[i]를 매번 가장 앞으로 옮겨야 한다.
- 배열에서 i번째 원소를 가장 앞으로 보내는 데 i번의 수행이 필요하다. (각 과정마다 O(N) 시간이 걸린다고 생각할 수 있다.)
- $1 + \cdots + N - 1 = O(N^2)$

- (average)평균적으로 i번째 작업에서 A[i]를 i / 2번째 위치로 옮기는 경우:
- i번째 작업에서 O(N)의 시간이 걸린다고 할 수 있다.
- 작업을 N번 수행해야 하므로 $O(N^2)$

질문?

퀵 정렬

- 배열이 주어지면 다음 작업을 수행하는 함수 sort를 정의하자.

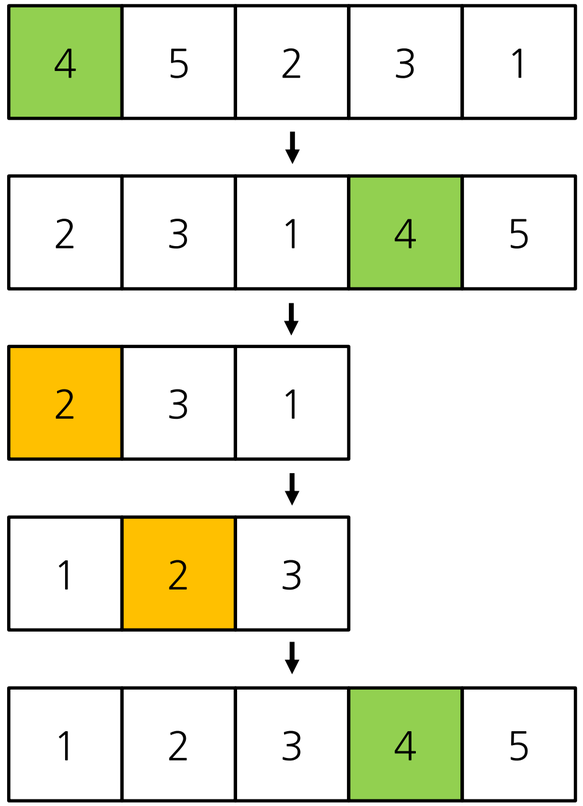
- sort(int A[])
- 배열 A의 길이가 0 또는 1이면, 이미 정렬된 배열이므로 함수를 종료한다.

- 배열 A에 있는 아무 원소를 pivot으로 잡는다.
- pivot보다 **작은 원소**를 pivot의 **왼쪽**으로 옮기고,
- pivot보다 **큰 원소**를 pivot의 **오른쪽**으로 옮긴다.

- pivot을 기준으로 왼쪽에 있는 배열에 대해서 sort를 다시 호출한다. (왼쪽 배열을 다시 정렬)
- pivot을 기준으로 오른쪽에 있는 배열에 대해서 sort를 다시 호출한다. (오른쪽 배열을 다시 정렬)

2023 Summer Algorithm Camp

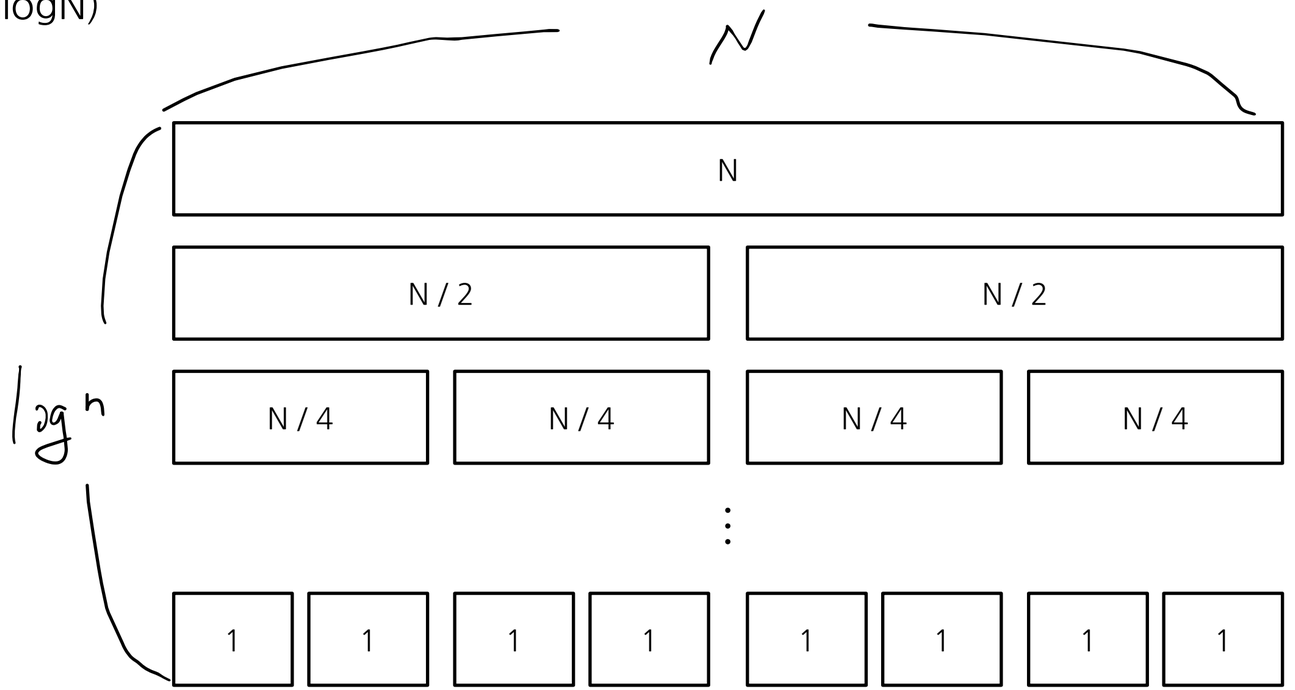
퀵 정렬



2023 Summer Algorithm Camp

퀵 정렬 - 시간 복잡도

- (best/average)매번 고르는 pivot이 왼쪽과 오른쪽 배열을 정확히(또는 평균적으로) 절반씩 나누는 경우:
- 호출 깊이(call depth)가 같은 sort 함수끼리 시간 복잡도 합은 O(N)
- 배열을 정확히 절반씩 나누기 때문에 호출 깊이는 $O(\log_2 N)$
- 시간 복잡도는 O(NlogN)

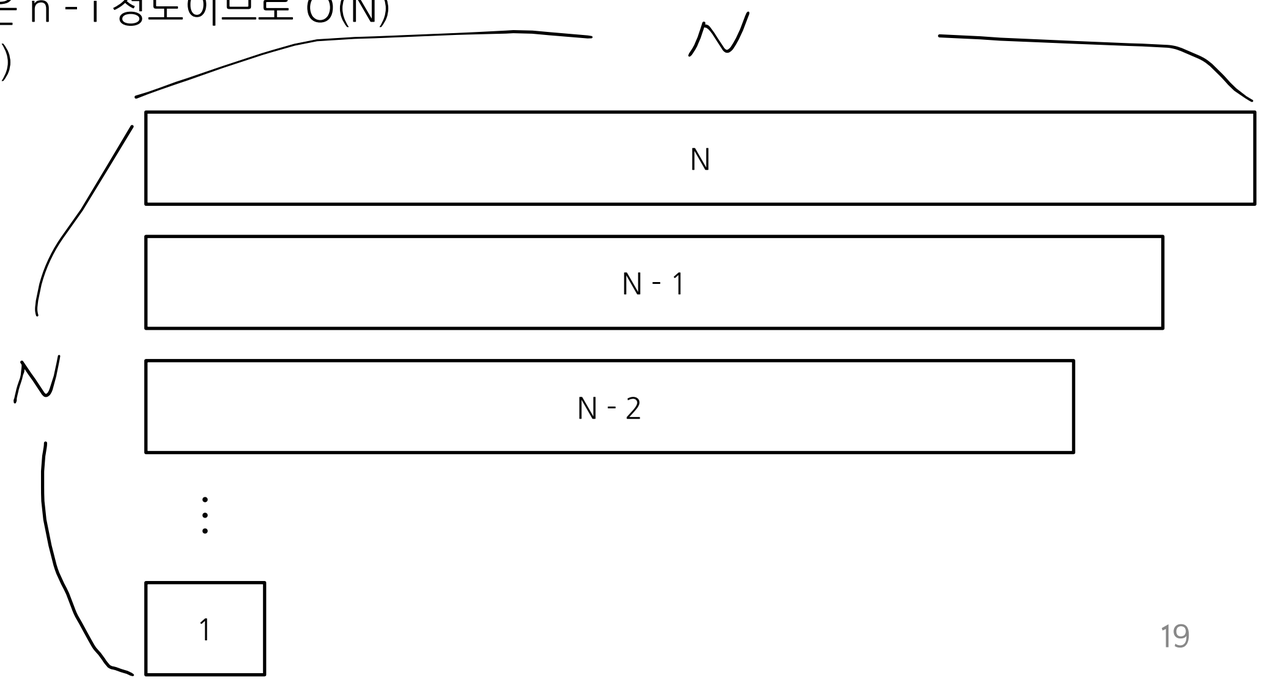


2023 Summer Algorithm Camp

퀵 정렬 - 시간 복잡도

- (worst)매번 고르는 pivot이 불균형하게 나누는 경우

- 배열의 길이가 1씩 감소하는 경우를 생각해 보자.
- [5, 4, 3, 2, 1]에서 pivot이 5이면, 5 [4 3 2 1] 과 같은 경우.
-
- 호출 깊이(call depth)가 i인 sort 함수에서 필요한 연산량은 n - i 정도이므로 O(N)
- 호출 깊이는 배열의 길이가 1이 될 때까지 반복되므로 O(N)
- 시간 복잡도는 $O(N^2)$



질문?

병합 정렬

- 배열이 주어지면 다음 작업을 수행하는 함수 sort를 정의하자.

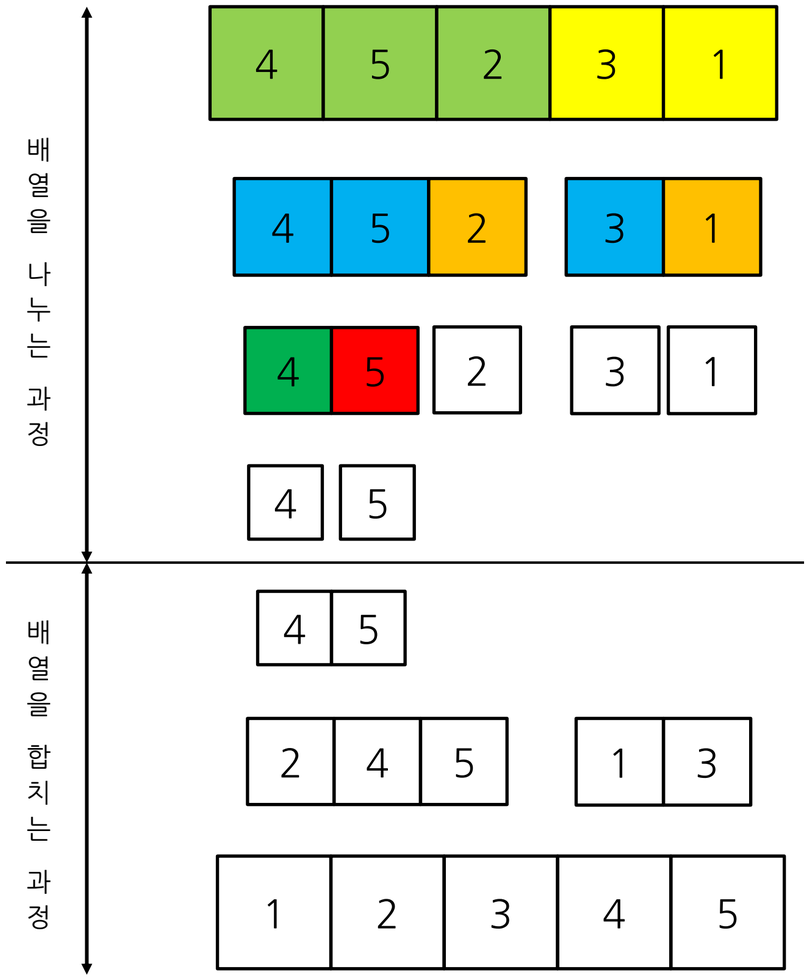
- sort(int A[])
- 배열 A의 길이가 0 또는 1이면, 이미 정렬된 배열이므로 함수를 종료한다.

- 배열 A를 다음 두 배열로 나눈다.
- 왼쪽 절반을 L, 나머지 오른쪽 절반을 R이라고 하자.
- sort(L)을 호출한다. (왼쪽 배열을 정렬)
- sort(R)을 호출한다. (오른쪽 배열을 정렬)

- 정렬된 두 배열 L과 R을 합쳐서 정렬된 배열 A를 반환한다.

2023 Summer Algorithm Camp

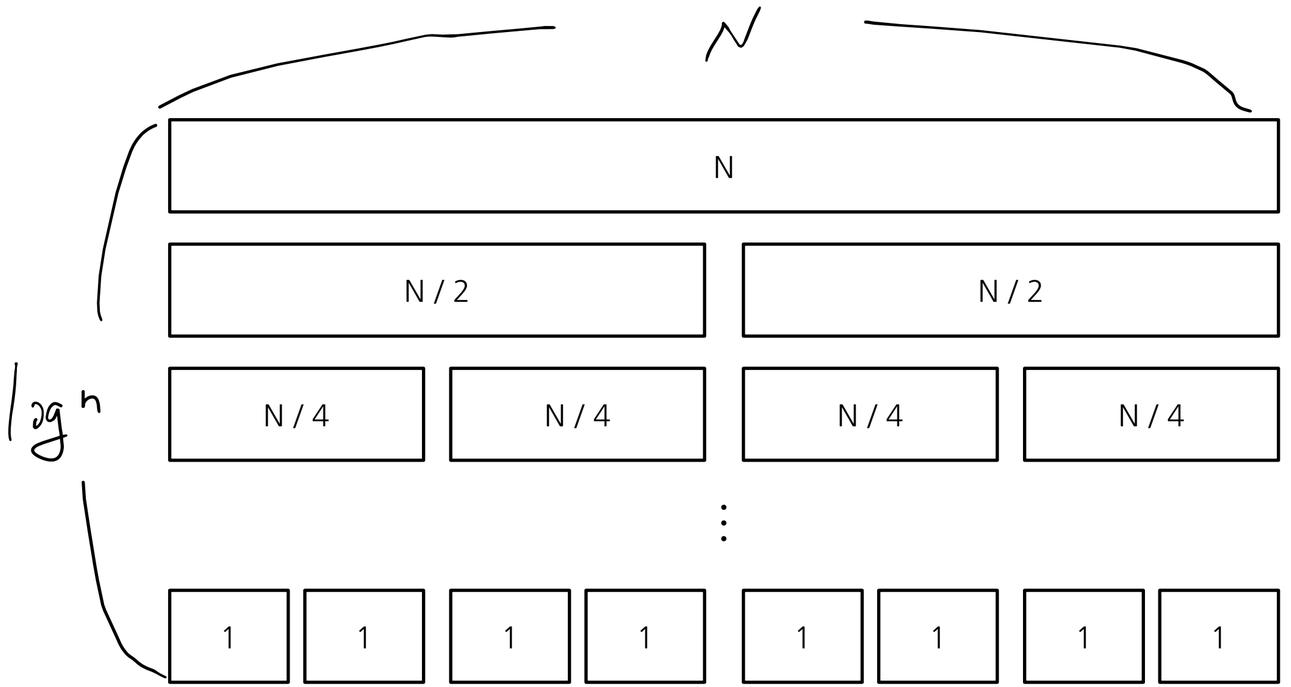
병합 정렬



2023 Summer Algorithm Camp

병합 정렬 - 시간 복잡도

- best/average/worst - 퀵 정렬의 best 경우와 동일
- 시간 복잡도는 O(NlogN)



질문?

비교 함수

bool compare(T a, T b);

- a가 b보다 “**무조건**” 앞에 나와야 한다면 true를 반환한다.
- 그렇지 않으면, false를 반환한다.
- 비교 함수는 **Strict Weak Ordering**을 만족해야 한다.

비교 함수 - Strict Weak Ordering

- 이항 관계(binary relation) **R(a, b)**에 대해서
- a가 b보다 **반드시 앞에 나와야 한다**면 **참**(T), 그렇지 않으면 거짓(F)이라고 하자.

- a가 b보다 앞에 나와야 한다면,
- R(**a, b**)는 **참**, R(**b, a**)는 **거짓**이다.
- 이 경우에는 a와 b를 비교할 수 있다고 한다. (**비교성**/comparability)

- a가 b와 **동등**equivalent하다면,
- R(a, b), R(b, a)는 **둘 다 거짓**이다.
- 이 경우에는 a와 b를 비교할 수 없다고 한다. (**비비교성**/incomparability)

비교 함수 - Strict Weak Ordering

- strict weak ordering은 다음 조건을 **모두** 만족해야 한다.

1. **비반사성** (irreflexivity) : 모든 a에 대하여 R(a, a)는 거짓
2. **비대칭성** (asymmetry) : 모든 a, b에 대하여 R(a, b)가 참이면 R(b, a)는 거짓
3. **전이성** (transitivity) : 모든 a, b, c에 대하여 R(a, b), R(b, c)가 참이면 R(a, c)는 참
4. **비비교성의 전이성** (transitivity of incomparability) : 모든 a, b, c에 대하여 R(a, b), R(b, a), R(b, c), R(c, b)가 거짓이면, R(a,c), R(c, a)는 거짓
   * 동등성의 전이성 (transitivity of equivalence)

비교 함수 - Strict Weak Ordering

- **비반사성** (irreflexivity)
- 모든 a에 대하여 R(a, a)는 거짓

- 같은 원소가 두 개 있다면 어떤 것이 앞에 와야 하는지 순서를 정할 수 있을까?
- 순서를 정할 수 없기 때문에 R(a, a)는 거짓이어야 한다.

- 따라서, 오름차순의 비교 함수로 $\leq$를 사용할 수 없다.

비교 함수 - Strict Weak Ordering

- **비대칭성** (asymmetry)
- 모든 a, b에 대하여 R(a, b)가 참이면 R(b, a)는 거짓

- a가 b보다 앞에 와야 하는데, b도 a보다 앞에 와야 한다고 하면 어떨까?
- 이런 상황에서 a와 b의 순서를 정할 수 없으므로, R(a, b)가 참이라면 R(b, a)는 거짓이어야 한다.

비교 함수 - Strict Weak Ordering

- **전이성** (transitivity)
- 모든 a, b, c에 대하여 R(a, b), R(b, c)가 참이면 R(a, c)는 참

- a가 b보다 앞에 오고, b가 c보다 앞에 와야 한다면, a … b … c와 같은 형태일 것이다.
- 따라서, R(a, c)도 참이어야 한다.

비교 함수 - Strict Weak Ordering

- **비비교성의 전이성** (transitivity of incomparability) * 동등성의 전이성 (transitivity of equivalence)
- 모든 a, b, c에 대하여 R(a, b), R(b, a), R(b, c), R(c, b)가 거짓이면, R(a,c), R(c, a)는 거짓

- R(a, b), R(b, a) 둘 다 거짓이라는 것은 a와 b가 비교할 수 없다(동등함)는 것을 의미한다.
- R(b, c), R(c, b)도 둘 다 거짓이면 b와 c도 비교할 수 없다.(동등함)
- 따라서, a와 c도 비교할 수 없어야 한다. (동등해야 함)

질문?

STL sort

```
#include <algorithm>
```

- std::sort(first, last);
- std::sort(first, last, comp);

- [first, last)를 감소하지 않는 순서로 정렬한다.

- comp: 비교 함수
  - 비교 함수의 원형 : bool compare(T a, T b);
  - a가 b보다 작다(앞에 와야 한다)면 true를 반환하는 함수

- 시간 복잡도 : O(NlogN)

STL sort

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    vector<int> vec = {4, 5, 2, 3, 1};
    sort(vec.begin(), vec.end());
    for (int i : vec) cout << i << " "; // 1 2 3 4 5
}
```

STL sort

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool compare(int a, int b) {
    return a > b;
}

int main() {
    vector<int> vec = {4, 5, 2, 3, 1};
    sort(vec.begin(), vec.end(), compare);
    for (int i : vec) cout << i << " "; // 5 4 3 2 1
}
```

STL sort

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    vector<int> vec = {4, 5, 2, 3, 1};
    sort(vec.begin(), vec.end(), greater<>());
    for (int i : vec) cout << i << " "; // 5 4 3 2 1
}
```

질문?

2023 Summer Algorithm Camp

정렬 연습 문제

- BOJ 2750 (수 정렬하기)
  - 적당한 $O(N^2)$ 정렬 알고리즘 직접 구현해서 풀어보기

- BOJ 2751 (수 정렬하기 2)
  - std::sort 연습 문제

240

질문?

선형 자료 구조

- 선형 자료 구조 : 하나의 데이터 뒤에 **하나의 데이터**만 올 수 있는 데이터 구조
- ex) 배열 등

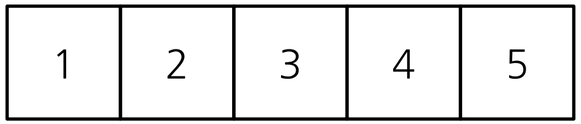
- 비선형 자료 구조 : 하나의 데이터 뒤에 **여러 데이터**가 올 수 있는 데이터 구조
- ex) 트리, 그래프 등

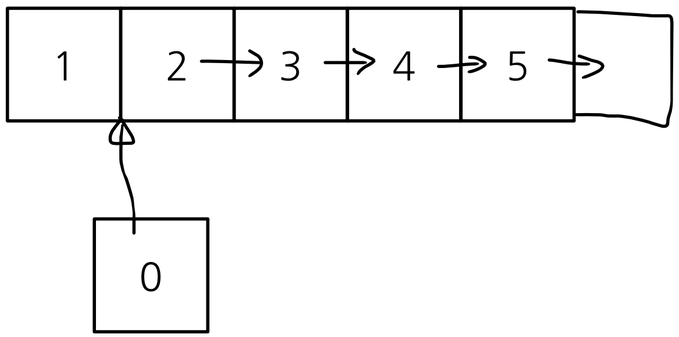
배열

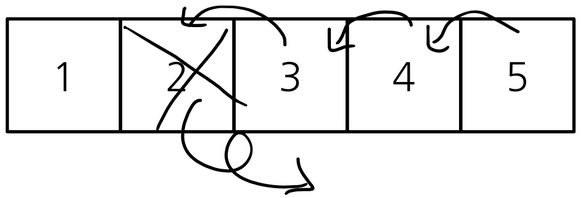
- 자료가 **물리적**으로 연속되어 저장되는 자료구조

- 시간 복잡도
  - 임의 위치 접근 : O(1)
  - 원소 삽입/삭제 : O(N)

- std::vector, std::array







연결 리스트

- 자료가 **논리적**으로 연속되어 저장되는 자료구조
  - 메모리상에서 연속하지 않는다.

- 시간 복잡도
  - 임의 위치 접근 : O(N) * 접근하려는 노드의 주소를 미리 알면 O(1)에 접근할 수 있다.
  - 원소 삽입/삭제 : O(1)

- std::list

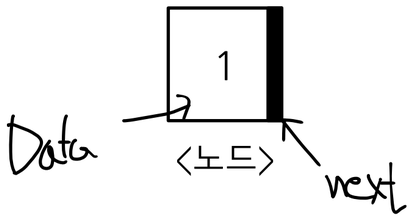
2023 Summer Algorithm Camp

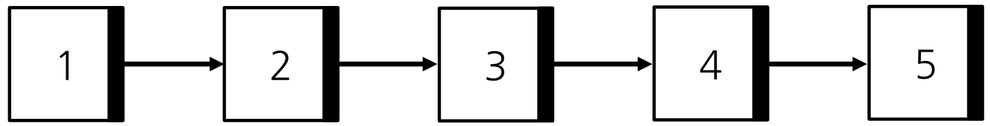
연결 리스트

- 각 자료를 노드Node라고 한다.
- 노드는 **데이터**와 **자신과 인접한 노드의 주소**를 저장한다.

- **다음 노드의 주소** 또는 이전 노드의 주소만을 저장하면 *Singly Linked List*
- 다음 노드와 이전 노드의 주소를 **모두 저장**하면 *Doubly Linked List*

- 연결 리스트는 가장 처음 또는 끝 노드를 저장한다.
- 임의 위치에 접근하려면 가장 처음 또는 끝 노드부터 시작해서 인접한 노드의 주소를 따라서 쭉 이동한다.
- 임의 위치 접근 : O(N)



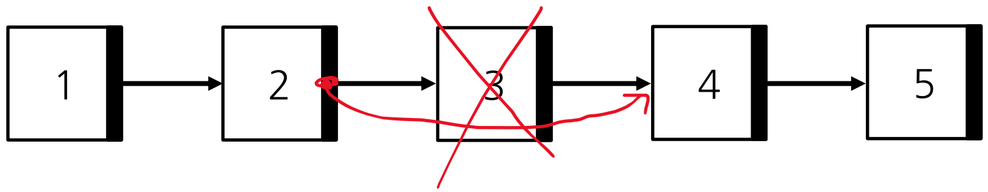


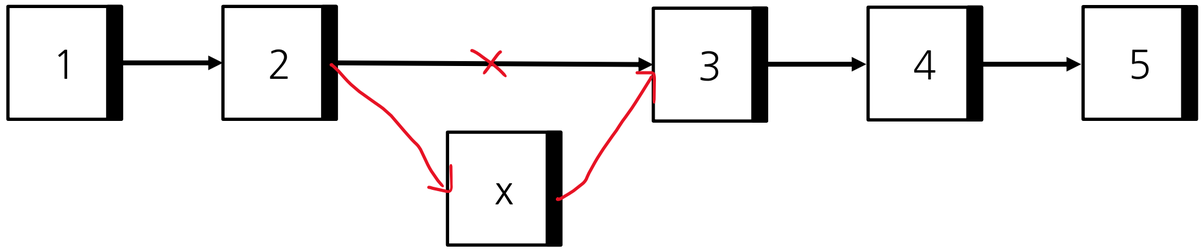
2023 Summer Algorithm Camp

연결 리스트

- 연결 리스트에 원소를 삽입하거나 제거해야 할 때, 배열보다 처리하기 쉽다.

- 인접한 노드의 주소 정보만 바꿔주면 된다.
- 삽입하거나 제거할 노드의 주소만 알고 있다면 O(1)에 처리할 수 있다.





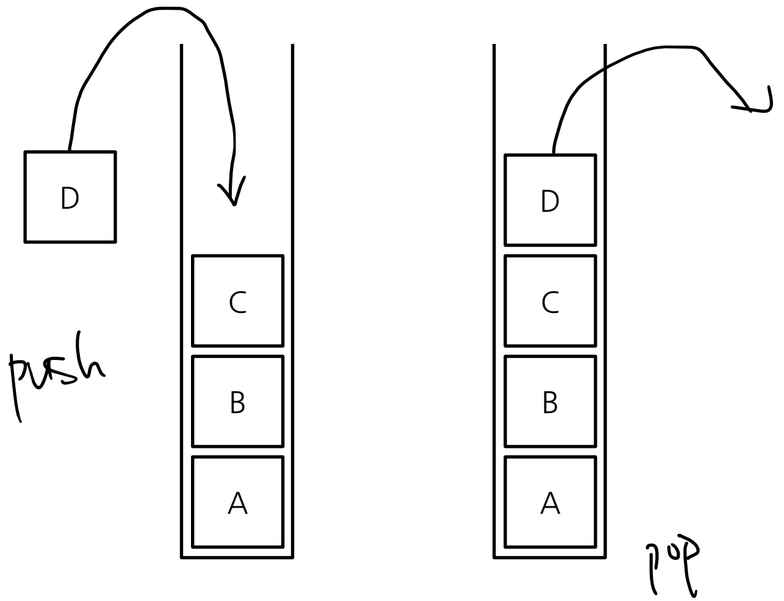
질문?

스택

- Last In First Out (LIFO) 자료구조
- 나중에 들어온 자료가 먼저 나온다.

- 할 수 있는 연산
- **push**(x) : x를 스택에 넣는다.
- **pop**() : 스택에서 가장 마지막에 추가된 원소를 뺀다.

- std::stack



스택

- 어떻게 구현할 수 있을까?
- 동적 배열, 연결 리스트 둘 다 가능하다.

- **동적 배열**
  - push(x) : 배열의 길이를 1 늘리고, 마지막 원소를 x로 한다.
  - pop() : 배열의 길이를 1 줄인다.

- **연결 리스트**
- 스택에서 가장 위에 있는 노드 하나를 저장한다. 이 노드를 Root라고 하자.
- 각 노드는 자신보다 이전에 들어온 노드의 주소를 저장한다.
  - push(x) : 기존의 Root를 가리키는 노드를 새로 만든다. 새로 만든 노드를 Root로 한다.
  - pop() : 기존의 Root가 가리키던 노드를 Root로 한다.

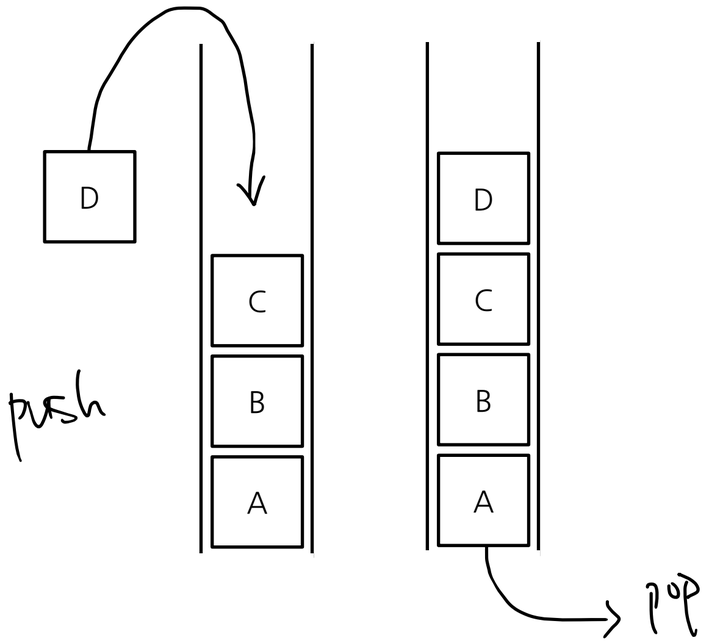
- 시간 복잡도
  - push, pop 둘 다 O(1)

큐

- First In First Out (FIFO) 자료구조
- 먼저 들어온 자료가 먼저 나온다.

- 할 수 있는 연산
- **push**(x) : x를 큐에 넣는다.
- **pop**() : 큐에서 가장 먼저 들어온 원소를 뺀다.

- std::queue



큐

- 어떻게 구현할 수 있을까?
- 효율적인 구현은 연결 리스트로 가능하다.

- **연결 리스트**
- 큐에서 가장 앞에 있는 노드와 가장 끝에 있는 노드를 저장한다.
- 각 노드는 자신보다 나중에 들어온 노드의 주소를 저장한다.
  - push(x) : 큐의 가장 끝에 있는 노드를 가리키는 노드를 새로 만든다. 새로 만든 노드는 큐의 가장 끝에 넣는다.
  - pop() : 원래 큐에서 가장 앞에 있는 노드가 가리키던 노드를 가장 앞에 있는 노드로 바꾼다.

- 시간 복잡도
  - push, pop 둘 다 O(1)

덱

- 덱은 FIFO, LIFO를 모두 지원하는 자료구조이다.

- 할 수 있는 연산
- **push_back**(x) : 덱의 가장 뒤에 x를 넣는다.
- **push_front**(x) : 덱의 가장 앞에 x를 넣는다.
- **pop_back**() : 덱의 가장 뒤 원소를 제거한다.
- **pop_front**() : 덱의 가장 앞 원소를 제거한다.

- 구현은 Doubly Linked List를 사용하고, 맨 앞/맨 뒤 원소를 저장해두면 된다.
  - 모든 연산은 O(1)에 동작한다.

- std::deque

질문?

240

STL stack

```
#include <stack>

- std::stack<T>

- void push(T a);
- T top();
- void pop();

- bool empty();
- unsigned int size();
```

STL stack

```cpp
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack<int> st;
    st.push(1);
    st.push(2);
    st.push(3);
    st.push(4);
    while (!st.empty()) {
        cout << st.top() << " "; // 4 3 2 1
        st.pop();
    }
}
```

STL queue

```
#include <queue>

- std::queue<T>

- void push(T a);
- T front();
- void pop();

- bool empty();
- unsigned int size();
```

STL queue

```cpp
#include <iostream>
#include <queue>

using namespace std;

int main() {
    queue<int> Q;
    Q.push(1);
    Q.push(2);
    Q.push(3);
    Q.push(4);
    while (!Q.empty()) {
        cout << Q.front() << " "; // 1 2 3 4
        Q.pop();
    }
}
```

STL deque

```
#include <deque>

- std::deque<T>

- void push_front(T a);
- void push_back(T a);
- T front();
- T back();
- void pop_front();
- void pop_back();

- bool empty();
- unsigned int size();

- T at(unsigned int pos);
```

STL deque

```cpp
#include <iostream>
#include <deque>

using namespace std;

int main() {
    deque<int> deq;
    deq.push_back(1);
    deq.push_front(2);
    deq.push_front(3);
    deq.push_back(4);

    deque<int> deq2 = deq; // copy

    while (!deq.empty()) {
        cout << deq.front() << " "; // 3 2 1 4
        deq.pop_front();
    }
    cout << "\n";
    while (!deq2.empty()) {
        cout << deq2.back() << " "; // 4 1 2 3
        deq2.pop_back();
    }
}
```

질문?

예시 문제

- BOJ 9012 (괄호)

- 괄호 문자열이 주어지면 올바른 괄호 문자열인지 판별하는 문제

- 올바른 괄호 문자열의 정의:
  - **()**는 올바른 괄호 문자열이다.
  - 올바른 괄호 문자열 x가 있다면, **(x)**도 올바른 괄호 문자열이다.
  - 올바른 괄호 문자열 x, y가 있다면, **xy**도 올바른 괄호 문자열이다.

예시 문제

- (()())((()))
- (()())((()))
- (()())((()))

- 닫는 괄호)가 등장할 때 **매칭하지 않은 가장 최근의 여는 괄호**(를 매칭하면 된다.
- (()())((()))

예시 문제

- 현재 열려 있는매칭하지 않은 여는 괄호의 개수를 세는 것만으로도 충분하다.
- 닫는 괄호가 나오면 여는 괄호의 개수를 1만큼 감소시킨다.

- (()())((()))

(	(	)	(	)	)	(	(	(	)	)	)
1	2	1	2	1	0	1	2	3	2	1	0

- 마지막 괄호까지 처리했을 때, 열려 있는 괄호의 개수가 0 초과거나
- 계산 도중 열려 있는 괄호의 개수가 0 미만이 되면 올바른 괄호 문자열이 아니다.

240

예시 문제

```cpp
#include <iostream>

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    int T; cin >> T;
    while (T--) {
        string s; cin >> s;
        int cnt = 0;
        int no = 0;
        for (char c : s) {
            if (c == '(') cnt++;
            else if (cnt == 0) no = 1;
            else cnt--;
        }
        if (cnt > 0) no = 1;
        cout << (no ? "NO" : "YES") << "\n";
    }
}
```

질문?

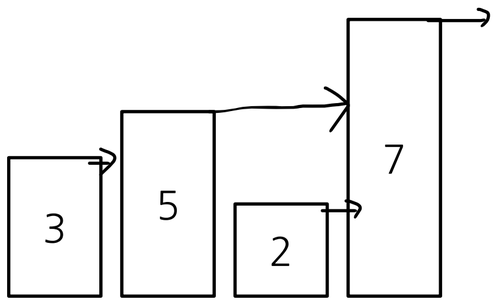
예시 문제

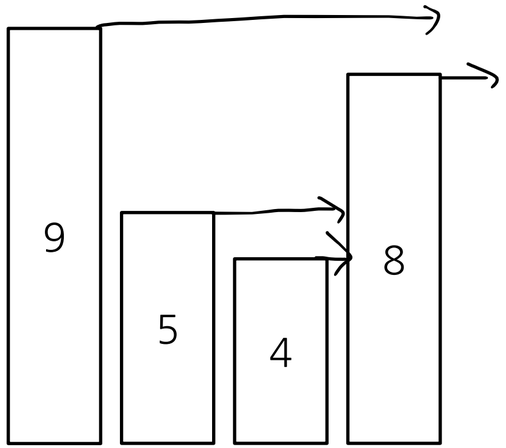
- BOJ 17298 (오큰수)

- 자신보다 오른쪽에 있는 큰 수 중에 가장 왼쪽에 있는 수를 구하는 문제

- A = [3, 5, 2, 7]이면 문제의 답은 [5, 7, 7, -1]
- A = [9, 5, 4, 8]이면 문제의 답은 [-1, 8, 8, -1]

- 막대 그래프로 나타내 보면, 자신의 막대에서 보이는 막대가 어떤 것인지 구하는 문제

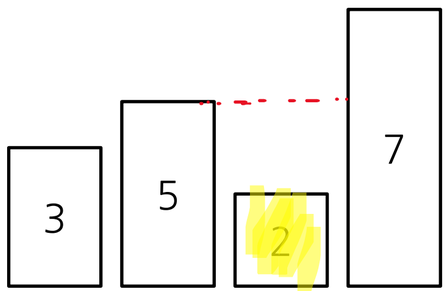


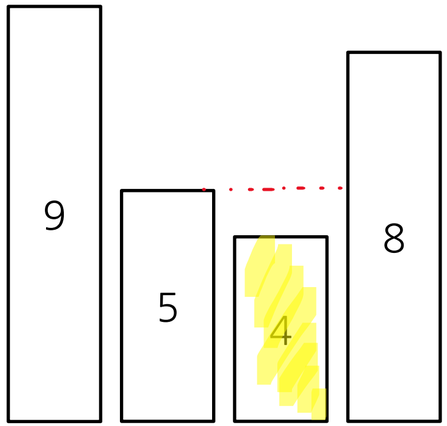


예시 문제

- $O(N^2)$에 문제를 해결하면 시간 초과를 받으므로 다른 풀이를 생각해 보자.

- 관찰
  - 오른쪽부터 왼쪽으로 막대를 하나씩 본다고 하자.
  - 자신보다 작은 막대는 자신의 막대에 가리기 때문에 자신보다 앞에 있는 막대에서는 절대 볼 수 없다.
  - 이런 성질을 잘 활용하여 답이 될 수 있는 막대들만 스택으로 관리해 보자.

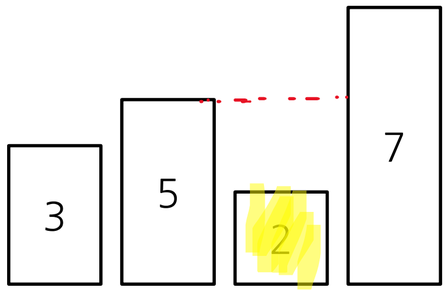


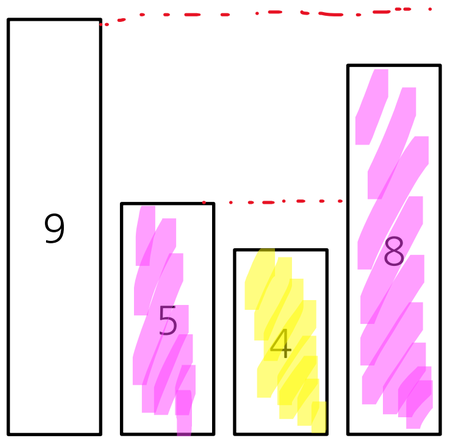


예시 문제

- 풀이
  - i = N .. 1 순서대로 A[i]를 본다.
  - A[i]를 넣을 때, Stack의 내용물이 A[i]보다 작거나 같다면 계속 Stack의 내용물을 제거한다.
    - 이 원소들은 A[i]를 가리지 않을뿐더러, A[i]보다 앞에 있는 원소에서 절대 이 원소들을 볼 수 없다. (A[i]에 의해 완전히 가리기 때문에)
  - Stack의 가장 위에 있는 원소가 해당 원소에 대한 답이 된다. (비어 있다면 -1이 답)
  - Stack에 A[i]를 넣는다.

- 이 과정에서 Stack은 오름차순을 유지하게 된다.
  - **Monotone Stack**





240

예시 문제

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    int n; cin >> n;
    vector<int> A(n);
    for (int &i : A) cin >> i;
    stack<int> S;
    vector<int> ans(n);
    for (int i = n - 1; i >= 0; i--) {
        while (!S.empty() && S.top() <= A[i]) S.pop();
        ans[i] = S.empty() ? -1 : S.top();
        S.push(A[i]);
    }
    for (int i : ans) cout << i << " ";
}
```

질문?

문제

- **필수 문제**
- BOJ 2750 (수 정렬하기)
- BOJ 2751 (수 정렬하기 2)
- BOJ 10828 (스택)
- BOJ 10845 (큐)
- BOJ 10866 (덱)

- **심화 문제**
- BOJ 9012 (괄호)
- BOJ 17298 (오큰수)
- BOJ 2003 (수들의 합 2)
- BOJ 11003 (최솟값 찾기)