02. Sorting

02. 정렬 (숭실대학교 박찬솔)

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

목차

정렬

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

- 오름차순 : a b c ... , $a \le b \le c \le \dots$
- 내림차순 : a b c ... , $a \ge b \ge c \ge \dots$

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.

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.

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.

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.)
- $1 + \dots + N - 1 = O(N^2)$

- (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

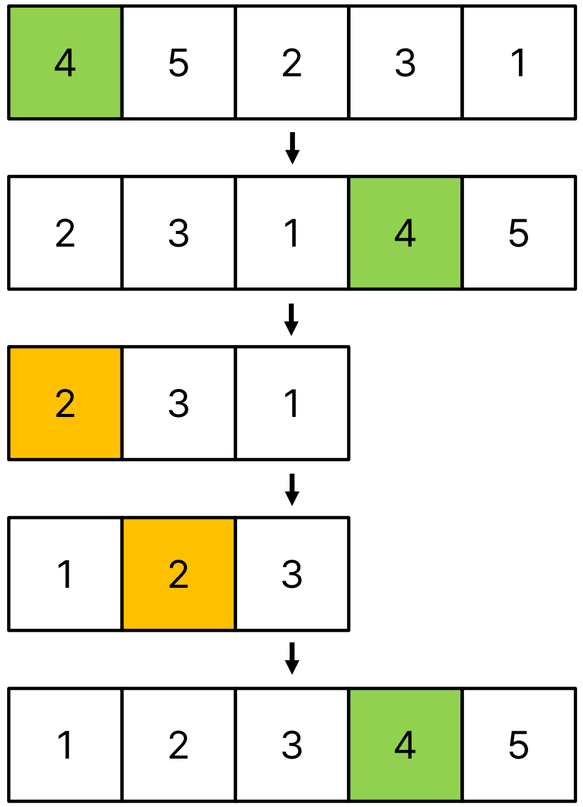
- 배열이 주어지면 다음 작업을 수행하는 함수 sort를 정의하자.

- 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.

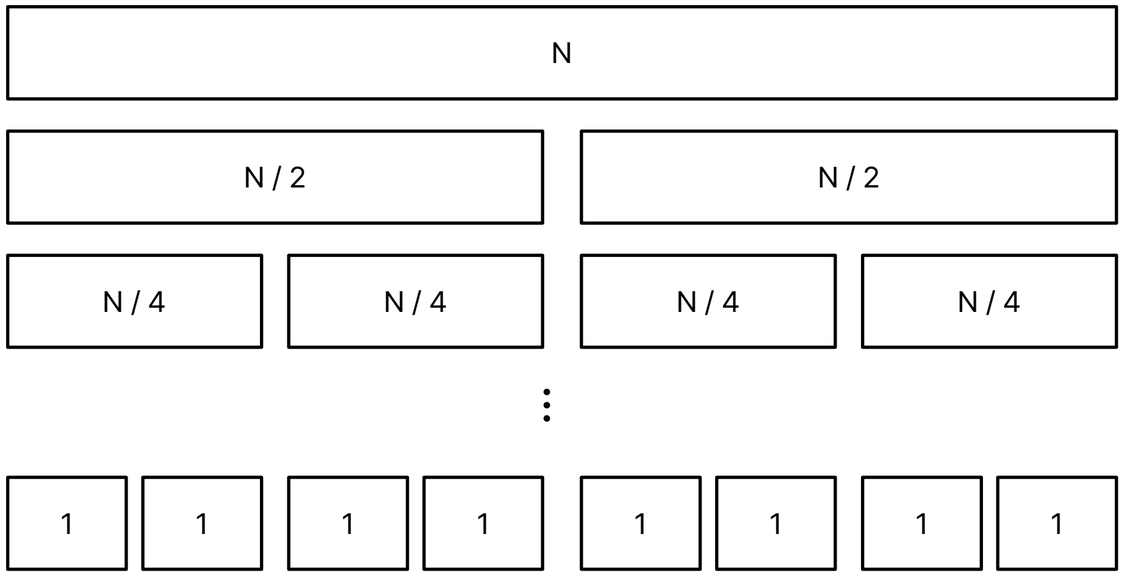
- pivot을 기준으로 왼쪽에 있는 배열에 대해서 sort를 다시 호출한다. (왼쪽 배열을 다시 정렬)
- pivot을 기준으로 오른쪽에 있는 배열에 대해서 sort를 다시 호출한다. (오른쪽 배열을 다시 정렬)

퀵 정렬



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)

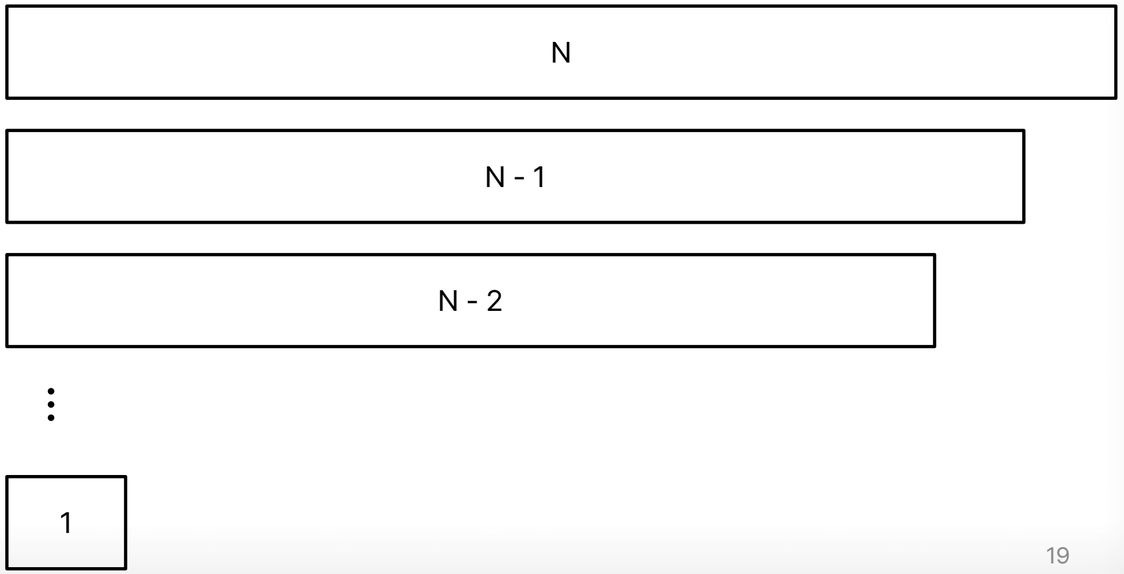


퀵 정렬 – 시간 복잡도

- (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

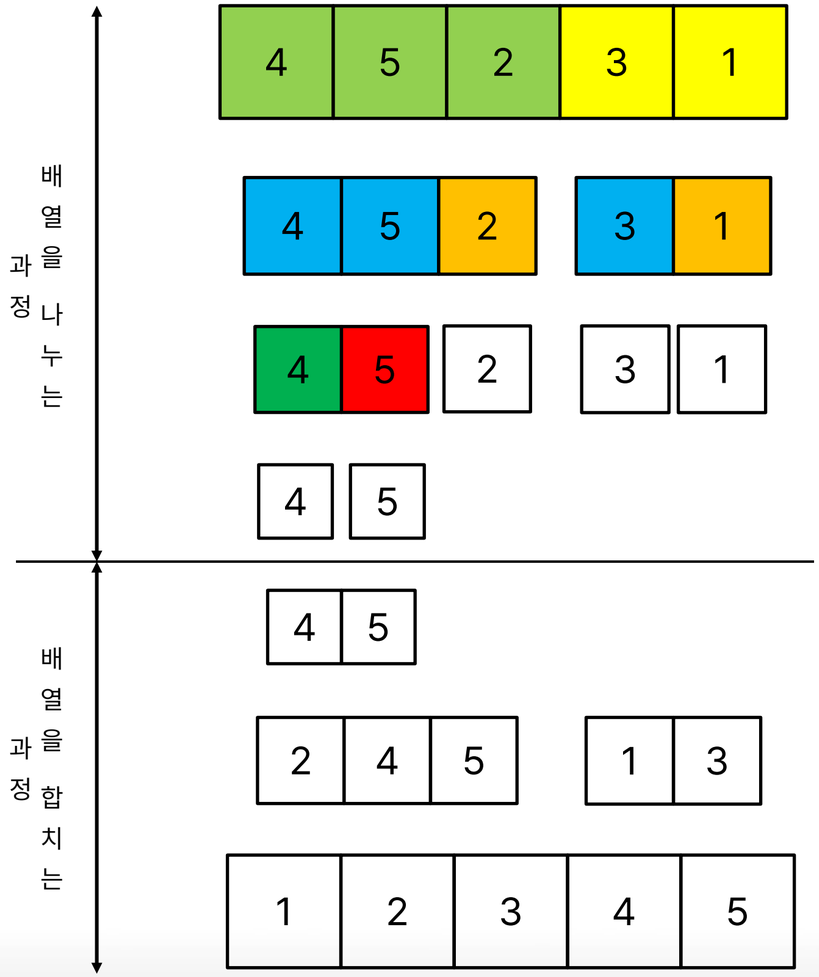
- 배열이 주어지면 다음 작업을 수행하는 함수 sort를 정의하자.

- 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)

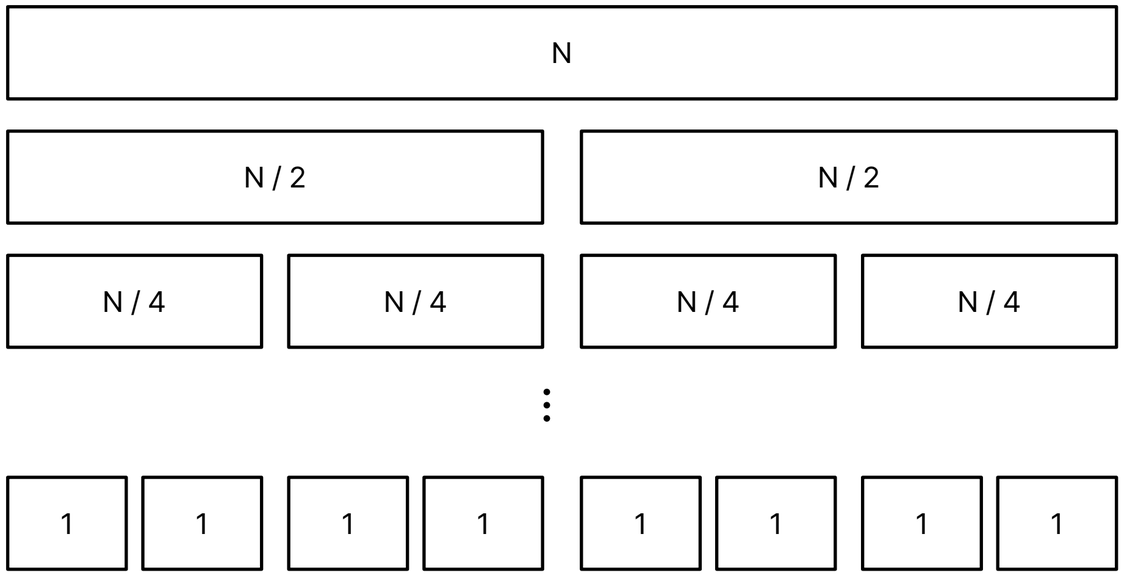
- 정렬된 두 배열 L과 R을 합쳐서 정렬된 배열 A를 반환한다.

병합 정렬



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

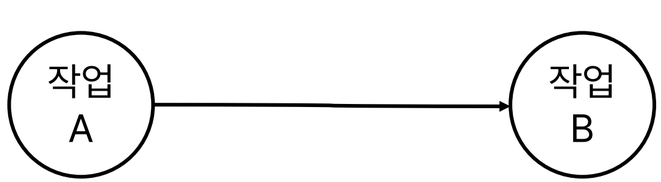
- 같은 원소가 두 개 있다면 어떤 것이 앞에 와야 하는지 순서를 정할 수 있을까?
- 순서를 정할 수 없기 때문에 R(a, a)는 거짓이어야 한다.

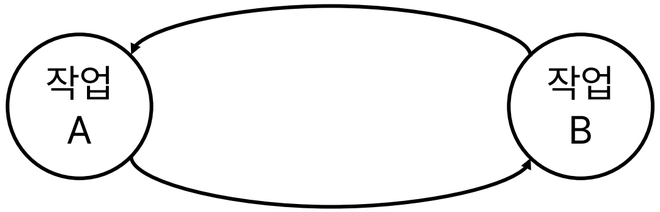
- 따라서, 오름차순의 비교 함수로 ≤를 사용할 수 없다.

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?

문제

- **필수 문제**
- BOJ 2750 (수 정렬하기)
- BOJ 2751 (수 정렬하기 2)
- BOJ 1026 (보물)
- BOJ 11650 (좌표 정렬하기)
- BOJ 10867 (중복 빼고 정렬하기)

- **심화 문제**
- BOJ 10989 (수 정렬하기 3)
  - 모르겠으면 계수 정렬counting sort에 대해 알아보세요.