김범수

분할 정복

Divide and Conquer

By POSCAT

Contents

분할 정복이란

- 분할 정복은 문제가 주어졌을 때, 문제를 나눌 수 없을 때까지 나누고 각 문제를 푼 후 결과를 합치면서 주어진 문제를 푸는 방법입니다.
- 분할 정복은 아래의 과정으로 이루어 집니다.
  1. Divide : 주어진 문제를 나눌 수 있다면, 2개 이상의 문제로 나눕니다.
  2. Conquer : 만약 나뉘어진 문제가 충분히 잘게 쪼개졌다면 문제를 풀고, 그렇지 않다면 문제를 더 나눕니다.
  3. Combine : 작은 문제의 정답을 이용하여 원래 문제의 정답을 구합니다.
- 분할 정복은 다음 슬라이드와 같이 재귀 함수의 형태로 구현하게 됩니다.

분할 정복 의사코드

Code Explanation

```
function solve(x):
    if x is solvable: // 문제 x를 더 쪼갤 수 없는 경우
        return solve(x)를 직접 푼 값
    else: // 문제를 더 쪼갤 수 있는 경우
        x1, x2 = split(x) // 문제 x를 x1과 x2로 분할한다.
        y1 = solve(x1), y2 = solve(x2) // 문제 x1과 문제 x2를 재귀적으로 해결합니다.
        return combine(y1, y2) // 문제 x1과 x2에서 얻은 결과를 합쳐 원래 문제를 해결합니다.
```

분할 정복을 이용한 정렬

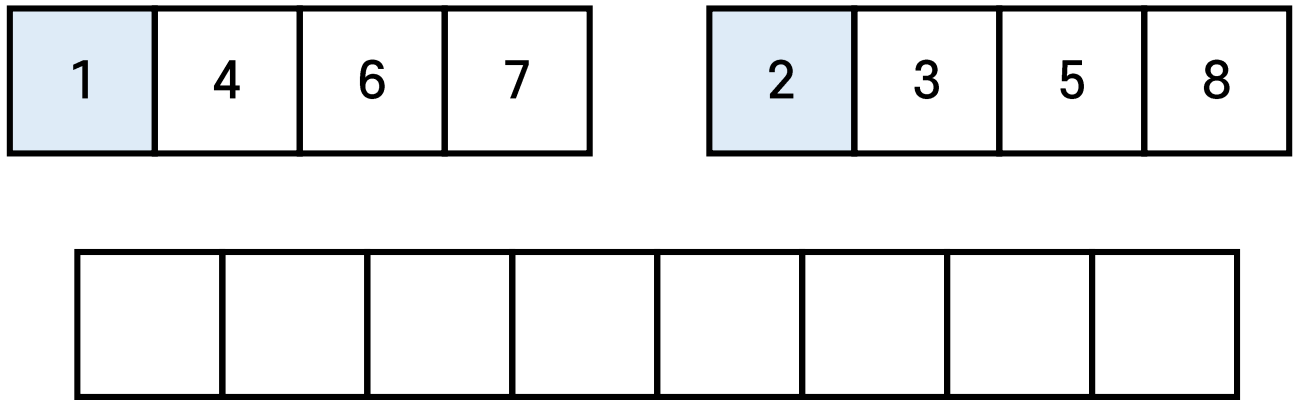
- 분할 정복을 실제로 코딩하려면 크게 2가지를 중점으로 구현해야 합니다.
  1. 문제를 언제까지 쪼개야 하고, 얼마나 잘게 쪼개졌을 때 간단하게 문제를 풀 수 있는가.
  2. 문제를 쪼개서 푼 결과를 어떻게 합쳐야 원래 문제를 풀 수 있는가.
- 위 2가지에 중점을 두면서 정렬을 분할 정복 알고리즘으로 풀어봅시다.
- 버블 소트 등 간단한 정렬 알고리즘은 길이가 n인 배열을 정렬하기 위해 $O(n^2)$의 시간복잡도를 요구합니다.
- 하지만 분할 정복을 이용하면 시간복잡도를 줄일 수 있습니다. 분할 정복에서 시간복잡도의 계산 방법은 뒤에서 나올 Master Theorem을 이용해 구할 수 있습니다.

분할 정복을 이용한 정렬

- 먼저 길이가 n인 배열을 정렬하기 위해 정렬을 수행하는 함수 solve(a, b)를 정의합니다.
- solve(a, b)는 a번째 수부터 b번째 수까지 정렬을 진행하는 함수입니다. 즉 우리의 목표는 solve(1, n)을 하는 것입니다.
- 우리는 배열을 2개로 쪼개서 각 배열을 정렬하고, 두 배열을 합쳐서 새로 정렬하는 방법을 이용할 것입니다.
- 그렇다면 배열은 언제까지 쪼갤 수 있을까요? 만약 배열을 길이가 1일 때까지 쪼갠다면 그 배열은 이미 정렬된 것이 당연하므로 문제를 이미 해결했다고 볼 수 있습니다. 즉 solve(a, a)는 아무 행동을 취하지 않아도 이미 해결되었다고 할 수 있습니다.
- 그렇다면 두 정렬된 배열을 어떻게 하나로 합칠 수 있을까요?
- 각 배열의 맨 앞 수부터 비교해가면서 새로운 배열에 작은 수부터 넣는 방식으로 구현할 수 있습니다.
- (1, 4, 6, 7)과 (2, 3, 5, 8) 두 정렬된 배열을 합치는 과정을 살펴보겠습니다.

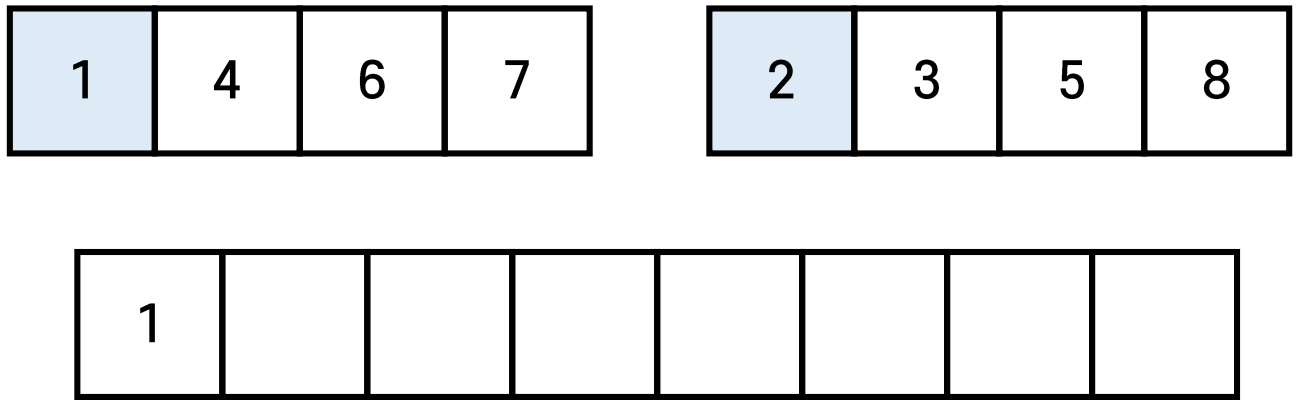
분할 정복을 이용한 정렬

- 먼저 두 배열의 가장 앞에 있는 두 원소를 비교합니다.



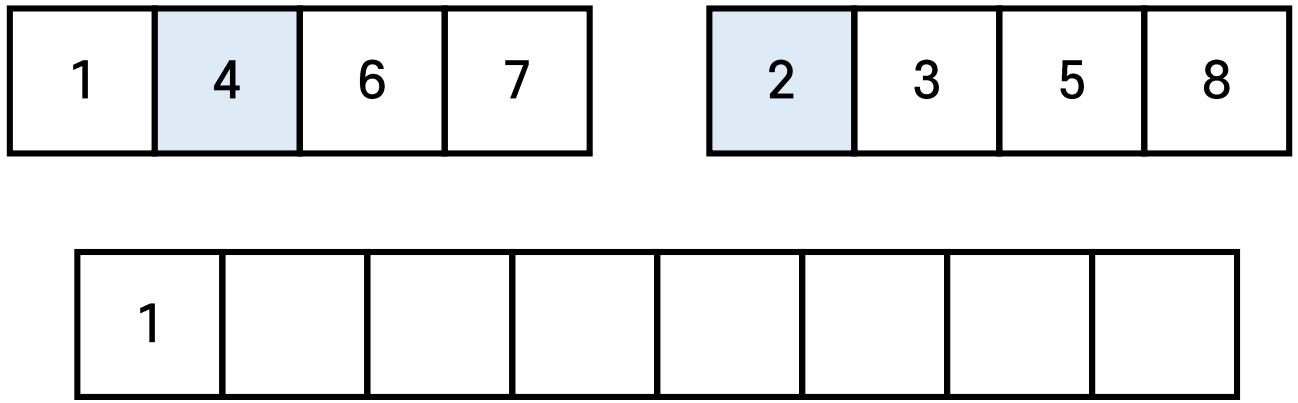
분할 정복을 이용한 정렬

- 1이 2보다 작으므로 새 배열이 1을 추가합니다.



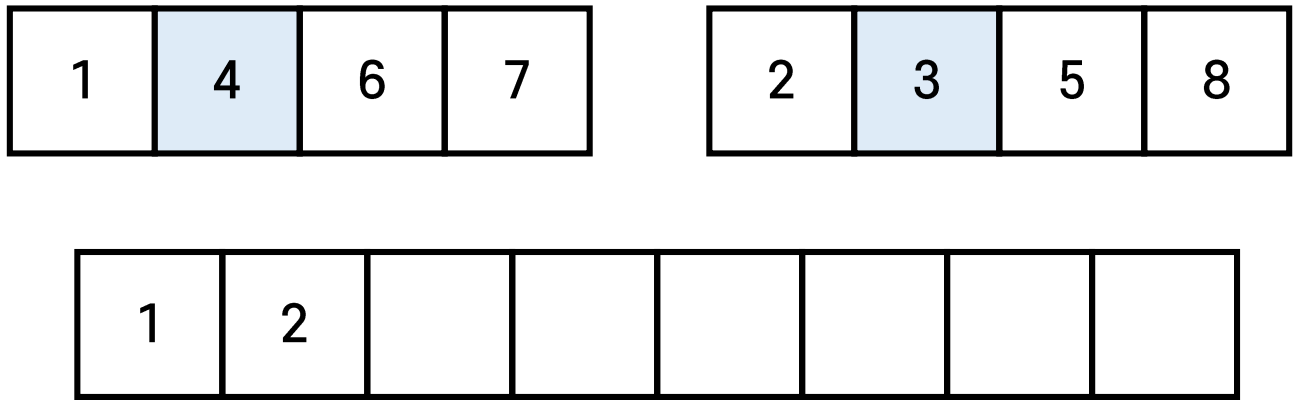
분할 정복을 이용한 정렬

- 이제 1번째 배열에서 다음 원소를 봅니다.



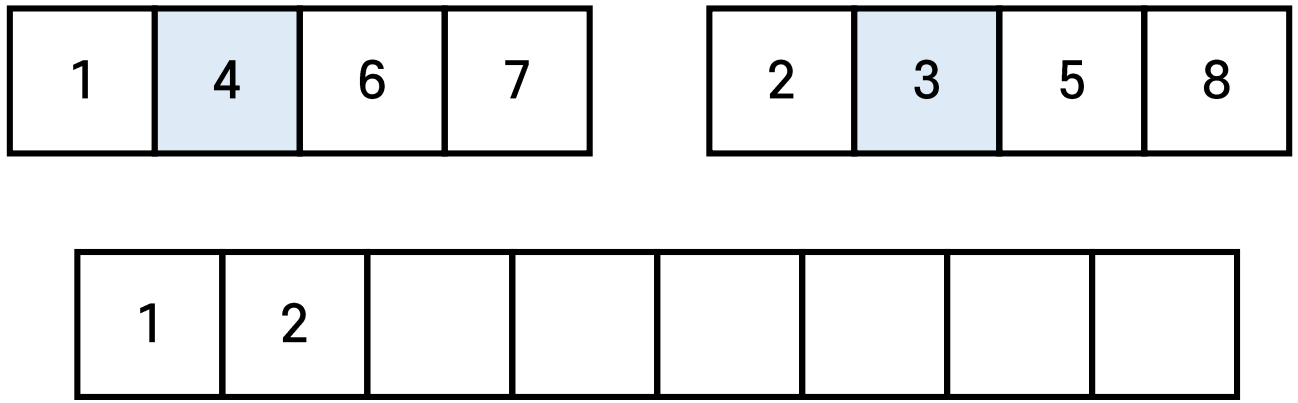
분할 정복을 이용한 정렬

- 2와 4를 비교하면 2가 더 작으므로 새 배열이 2를 추가합니다. 그 후 2번째 배열에서 다음 원소를 봅니다.



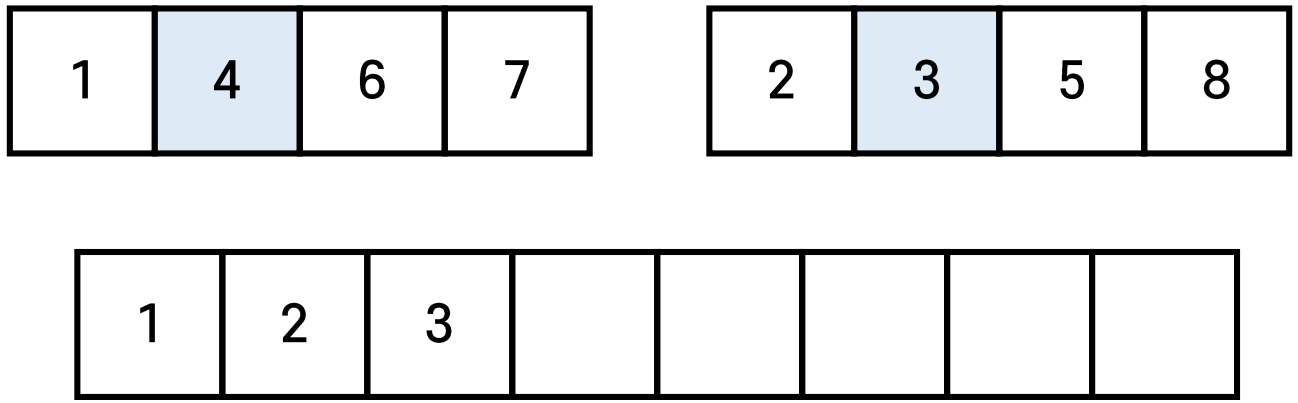
분할 정복을 이용한 정렬

- 이 과정을 계속 반복하면 됩니다.



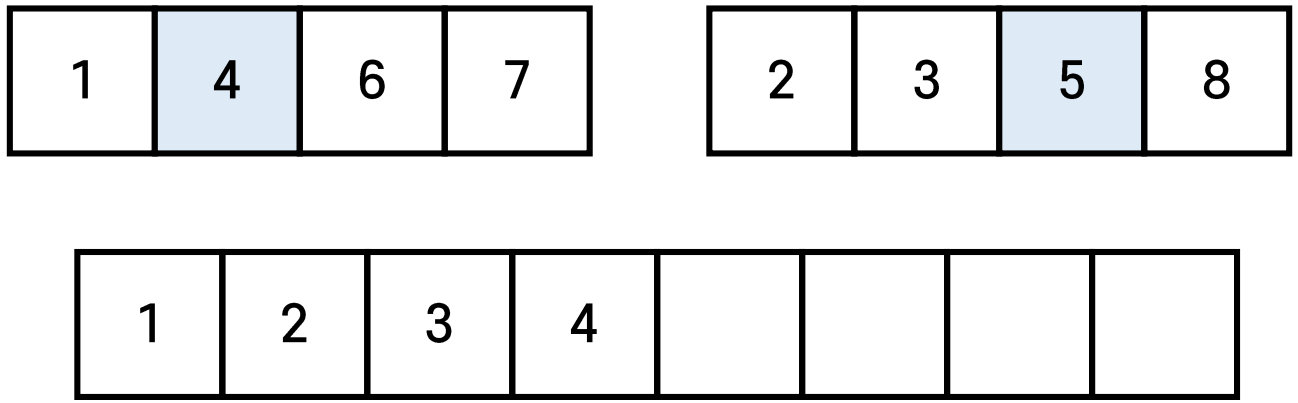
분할 정복을 이용한 정렬

- 이 과정을 계속 반복하면 됩니다.



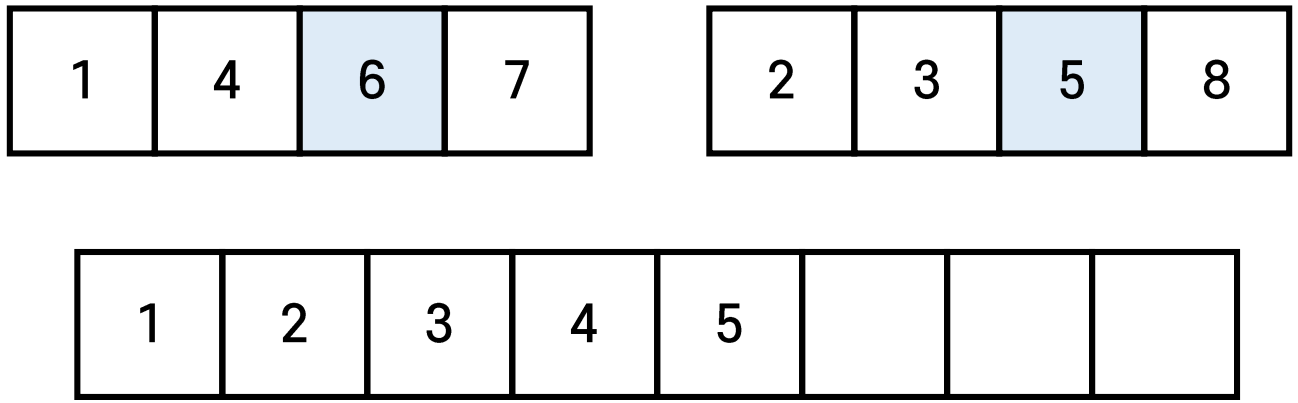
분할 정복을 이용한 정렬

- 이 과정을 계속 반복하면 됩니다.



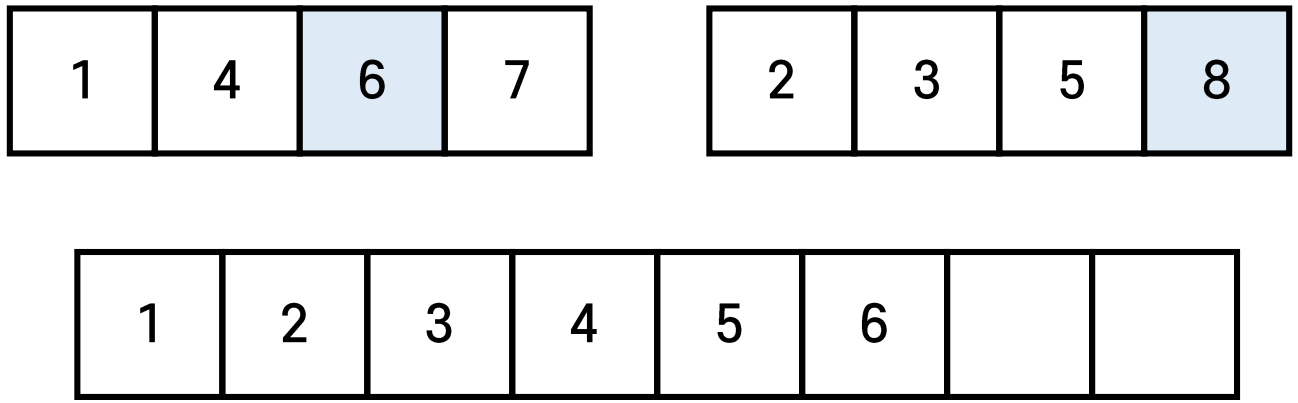
분할 정복을 이용한 정렬

- 이 과정을 계속 반복하면 됩니다.



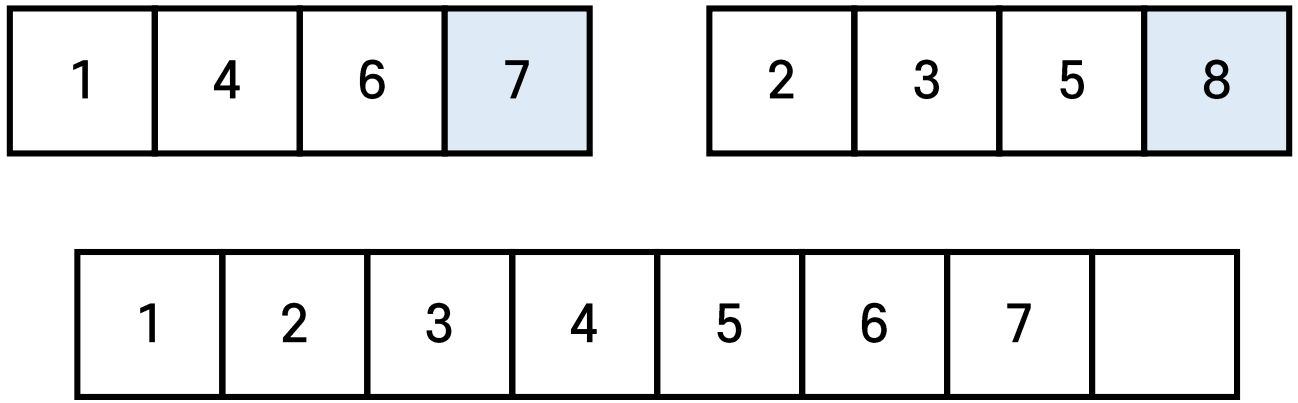
분할 정복을 이용한 정렬

- 이 과정을 계속 반복하면 됩니다.



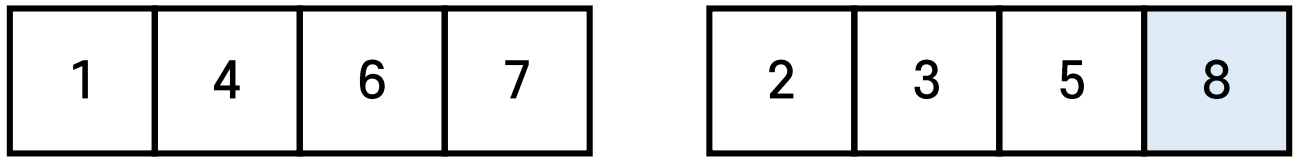
분할 정복을 이용한 정렬

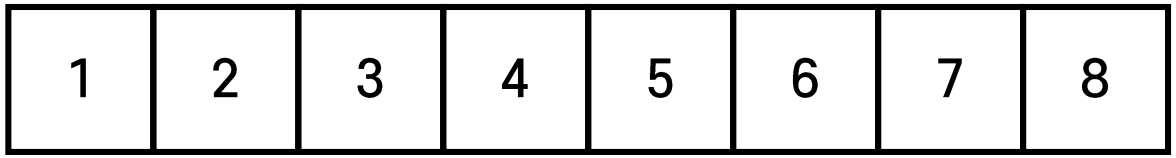
- 이 과정을 계속 반복하면 됩니다.



분할 정복을 이용한 정렬

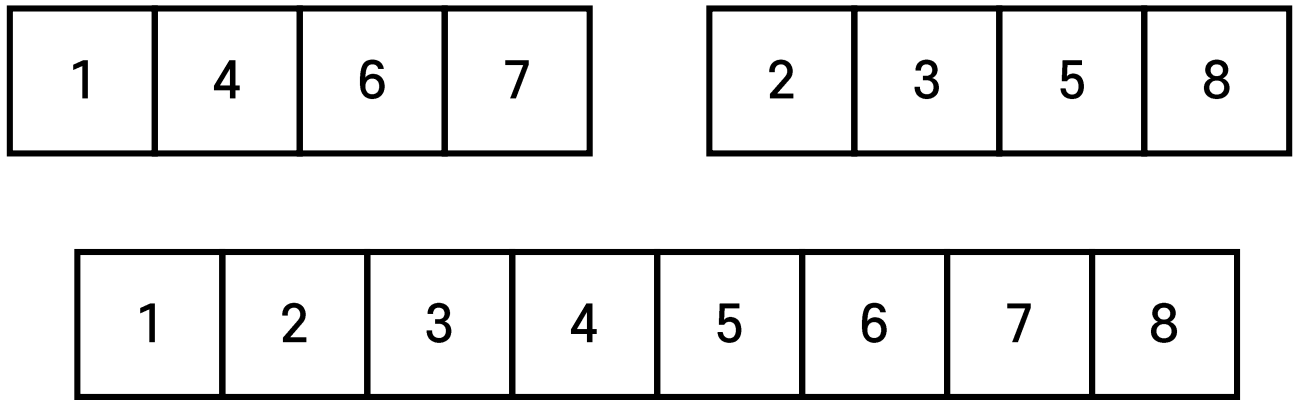
- 1번째 배열을 모두 처리했으므로 2번째 배열에서 남은 원소를 새 배열에 넣어줍니다.





분할 정복을 이용한 정렬

- 이렇게 2개의 정렬된 배열을 $O(n)$ 안에 하나의 정렬된 배열로 합칠 수 있습니다.



분할 정복을 이용한 정렬

Code Explanation

```
void my_sort(vector <int> &v){ // vector v를 정렬하는 함수
    if(v.size() == 1) return; // 만약 배열의 길이가 1이라면, 이미 정렬된 상태이므로 바로 종료
    int mid = v.size() / 2;
    vector <int> v1(v.begin(), v.begin() + mid), v2(v.begin() + mid, v.end()); // 주어진 배열을 2개의 배열로 쪼갠다.
    my_sort(v1), my_sort(v2); // 쪼개진 두 배열을 각각 정렬한다.
    merge(v, v1, v2); // 두 정렬된 배열을 하나의 정렬된 배열로 합친다.
}

// merge의 구현은 생략한다.
```

Master Theorem

- Master Theorem은 분할 정복뿐만이 아니라 재귀적으로 계산이 이뤄지는 알고리즘의 시간 복잡도를 계산하는 데 사용하는 방법입니다.
- 먼저 주어진 문제를 $T(n)$라 하고, 점화식이 아래와 같이 정의될 때

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- 알고리즘의 시간복잡도는 아래와 같이 구할 수 있습니다.

$$T(n) = \begin{cases} O\big(g(n)\big)\ if\ g(n) > f(n) \\ O(g(n)logn)\ if\ g(n) = f(n) \\ O\big(f(n)\big)\ if\ g(n) < f(n) \end{cases}$$

$$where\ g(n) = nlog_b a$$

Master Theorem

- 이 때, f(n) = g(n)은 시간 복잡도가 동일하는 것을 의미합니다. 즉 f(n) = n^2, g(n) = n^2 + 2n이더라도, f(n)과 g(n)의 시간복잡도는 f(n) = g(n) = O(n^2)이므로 f(n) = g(n)이 됩니다.

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ 의 의미는 크기가 n인 문제를 n/b 크기의 문제 a개로 나누어 푼 뒤, 이 문제들의 결과를 합치는 데 f(n)의 시간이 필요하다는 의미입니다.

- 앞서 분할 정복을 이용한 정렬 문제에서는 길이 n짜리 배열을 n/2짜리 배열 2개로 나누고 정렬한 뒤 O(n)동안 합쳤으므로

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

의 점화식으로 나타낼 수 있습니다.

Master Theorem

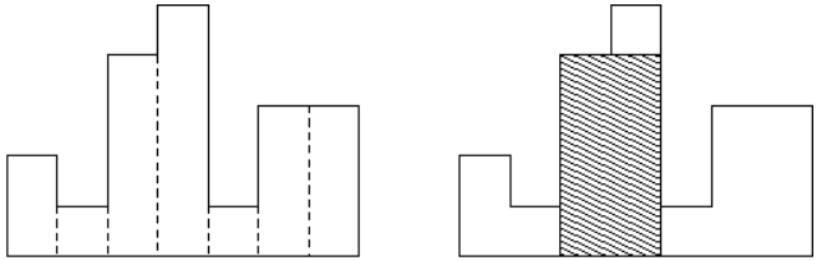
- $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$ 에서 $a = 2, b = 2, f(n) = O(n), g(n) = nlog_b a = nlog_2 2 = O(n)$ 이므로 $f(n) = g(n)$입니다.
- 따라서 Master Theorem을 이용하면 분할 정복을 이용한 정렬의 시간 복잡도는 $O(g(n)logn) = O(nlogn)$이 됩니다.
- 실제로 c++에서 사용하는 정렬 알고리즘의 시간 복잡도도 $O(nlogn)$입니다. 일반적인 정렬 알고리즘은 $O(nlogn)$보다 빠를 수 없음이 이미 증명되어 있으므로, 일반적인 상황에서는 c++의 sort()함수를 사용하시면 됩니다.

히스토그램 가장 큰 직사각형

- 분할 정복을 이용해서 다음 문제를 풀어봅시다. 물론 직사각형의 각 변은 히스토그램의 변과 평행합니다.

문제

히스토그램은 직사각형 여러 개가 아래쪽으로 정렬되어 있는 도형이다. 각 직사각형은 같은 너비를 가지고 있지만, 높이는 서로 다를 수도 있다. 예를 들어, 왼쪽 그림은 높이가 2, 1, 4, 5, 1, 3, 3이고 너비가 1인 직사각형으로 이루어진 히스토그램이다.



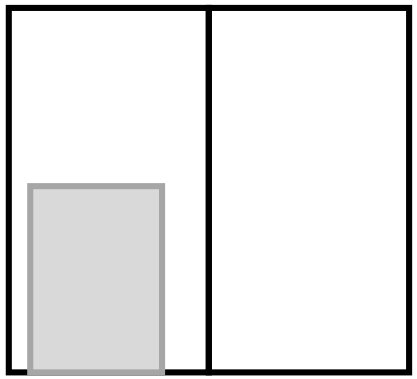
히스토그램에서 가장 넓이가 큰 직사각형을 구하는 프로그램을 작성하시오.

히스토그램 가장 큰 직사각형

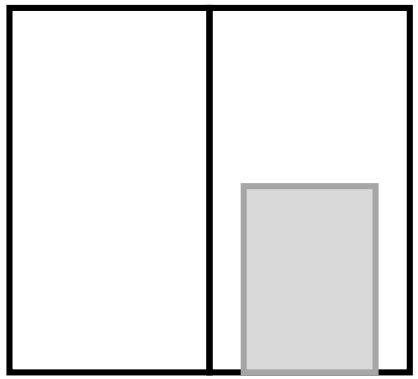
- 먼저 solve(a, b)를 a번 히스토그램부터 b번 히스토그램을 이용해서 만들 수 있는 가장 큰 직사각형의 넓이를 구하는 문제로 정의합니다. 그렇다면 n개의 히스토그램이 주어질 때, 우리의 목표는 solve(1, n)을 구하는 것이 될 것입니다.
- 먼저 solve(a, a), 즉 1개의 히스토그램에서 가장 큰 직사각형의 넓이는 간단하게 a번째 히스토그램의 넓이가 될 것입니다.
- 그렇다면 n개의 히스토그램을 2개로 쪼갠 후 얻은 각 답을 어떻게 합칠 수 있을까요?

히스토그램 가장 큰 직사각형

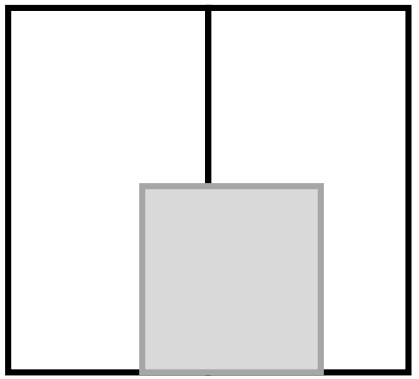
- 먼저 히스토그램을 두 부분으로 나눈다면 가장 큰 직사각형의 위치는 아래처럼 세 경우로 나눌 수 있습니다.
- 이 때 case 1, 2처럼 왼쪽이나 오른쪽 부분에만 있는 경우에는 각 부분문제에서의 답을 이용하면 되므로 큰 문제가 없습니다. 하지만 case 3처럼 왼쪽 부분과 오른쪽 부분에 걸쳐있는 경우에는 직접 계산을 해주어야 합니다.
- case 3의 경우는 greedy를 이용해서 구할 수 있습니다.



case 1



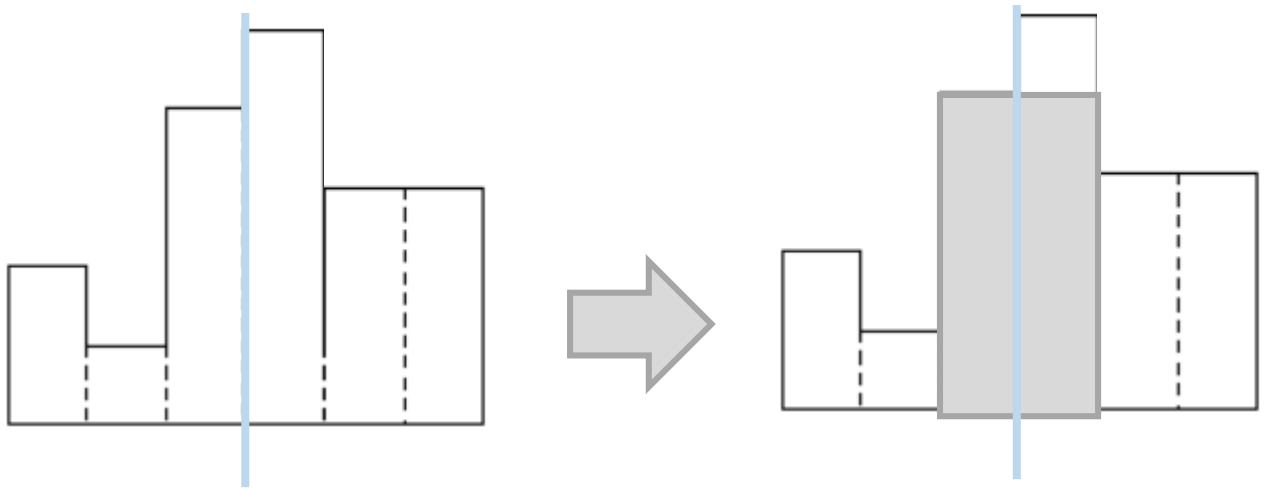
case 2



case 3

히스토그램 가장 큰 직사각형

- 먼저 아래의 히스토그램을 파란 선을 기준으로 쪼개서 계산했다고 가정하겠습니다. 그렇다면 먼저 파란 선을 기준으로 왼쪽 히스토그램 1개와 오른쪽 히스토그램 1개를 이용해서 가장 큰 직사각형의 넓이를 구합니다.

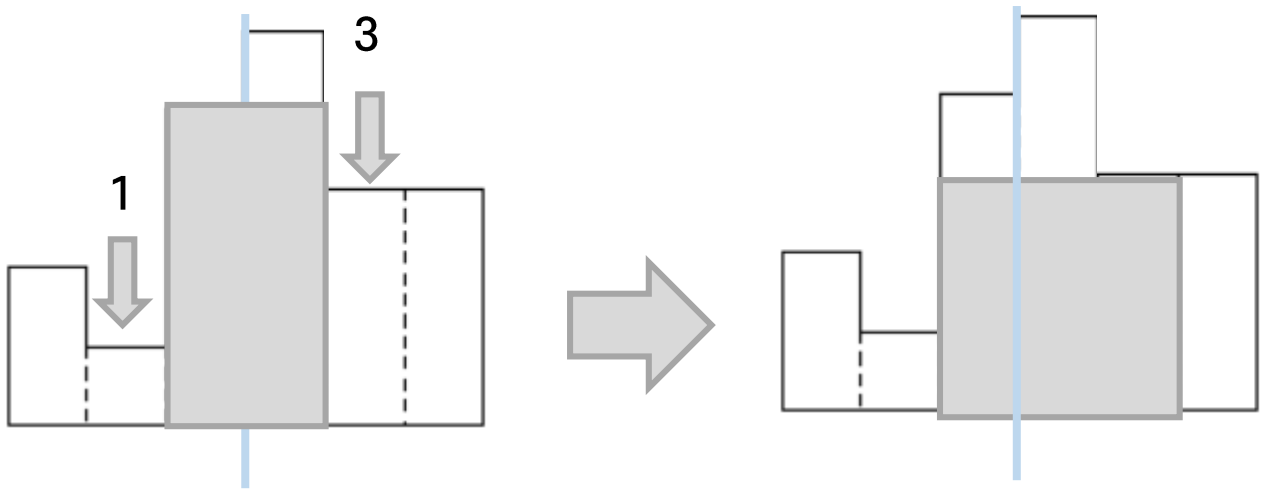


area = 4 * 2 = 8

max area = 8

히스토그램 가장 큰 직사각형

- 그 후 왼쪽과 오른쪽 중 한 방향으로 직사각형의 가로 길이를 1만큼 늘릴 것입니다. 왼쪽 히스토그램과 오른쪽 히스토그램 중 높이가 높은 방향으로 1만큼 늘립니다. 만약 직사각형의 높이를 낮춰야 한다면 낮춥니다.

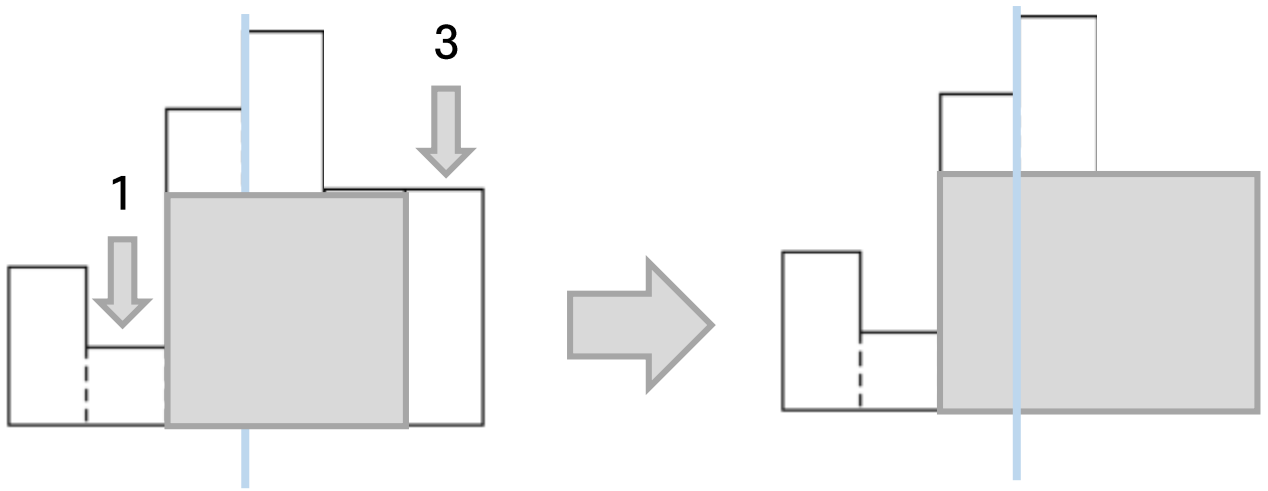


area = 3 * 3 = 9

max area = 9

히스토그램 가장 큰 직사각형

- 이 과정을 직사각형의 가로 길이가 최대한이 될 때까지 반복합니다.

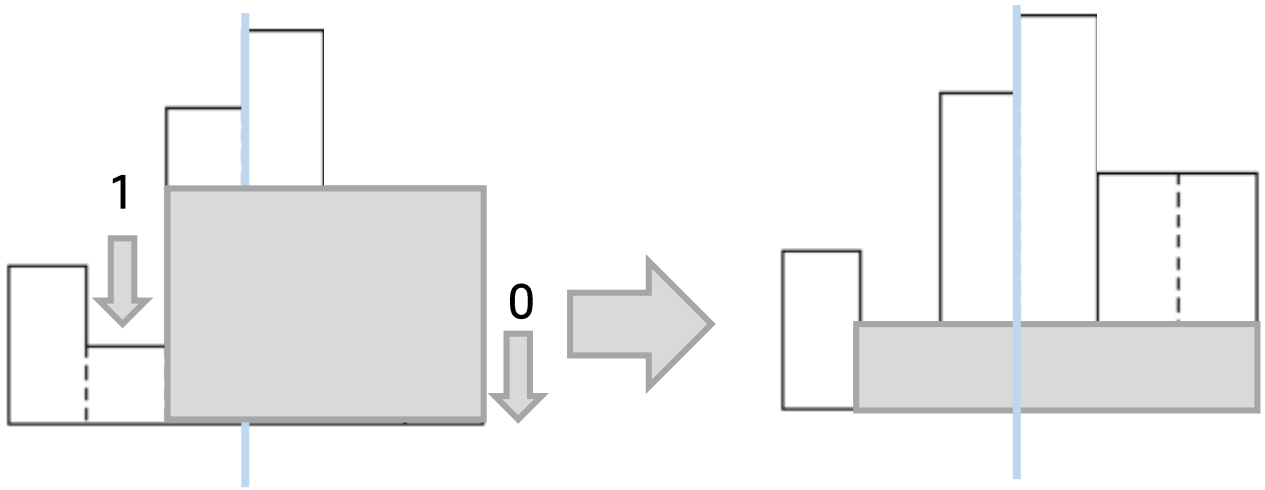


area = 3 * 4 = 12

max area = 12

히스토그램 가장 큰 직사각형

- 이 과정을 직사각형의 가로 길이가 최대한이 될 때까지 반복합니다.

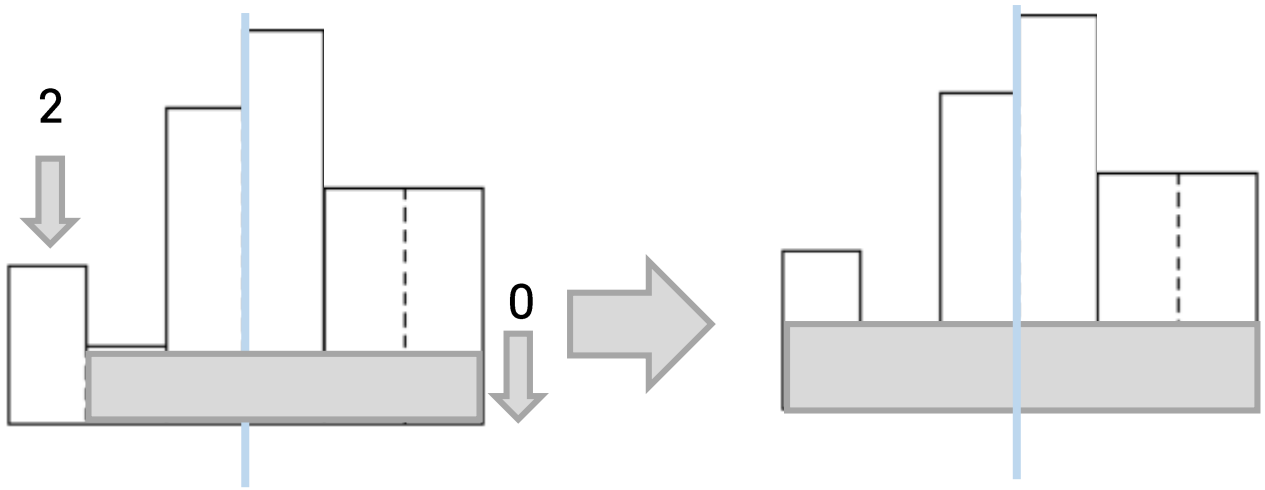


area = 1 * 5 = 5

max area = 12

히스토그램 가장 큰 직사각형

- 이 과정을 직사각형의 가로 길이가 최대한이 될 때까지 반복합니다.



area = 1 * 6 = 6

max area = 12

히스토그램 가장 큰 직사각형

- 이렇게 왼쪽과 오른쪽에 사각형이 걸쳐 있을 때, 가능한 최대 넓이는 12임을 알 수 있습니다. 왼쪽과 오른쪽에만 사각형이 있는 경우에는 재귀적으로 구할 수 있습니다. 따라서 주어진 히스토그램에서 가장 큰 직사각형의 넓이는 이 세 case의 최대 넓이 중 가장 큰 값이 됩니다.

관련 문제

- 쿼드트리 https://www.acmicpc.net/problem/1992
- 종이의 개수 https://www.acmicpc.net/problem/1780
- 행렬 제곱 https://www.acmicpc.net/problem/10830
- 히스토그램에서 가장 큰 직사각형 https://www.acmicpc.net/problem/6549