Divide and Conquer
중급 2주차

서강대학교 전해성(seastar105)

목차

재귀 함수

재귀 함수는 보통 자기 자신을 함수 내에서 호출 하는 함수를 말합니다.

무한히 호출하는 것을 막기 위해서 호출을 더 이상 하지 않는 경우를 base case라고 정의한 뒤에 이런 경우엔 호출을 그만둡니다.

예시) 최대공약수를 구하는 유클리드 알고리즘

```
1  int gcd(int a, int b) {
2    if(b == 0) return a;
3    return gcd(b, a%b);
4  }
```

하노이 탑 이동 순서(백준 11729번)

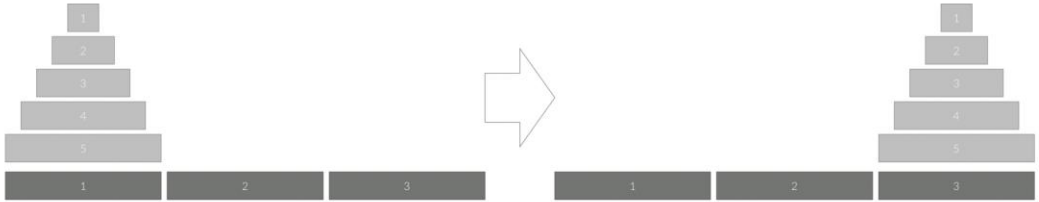
문제

세 개의 장대가 있고 첫 번째 장대에는 반경이 서로 다른 n개의 원판이 쌓여 있다. 각 원판은 반경이 큰 순서대로 쌓여있다. 이제 수도승들이 다음 규칙에 따라 첫 번째 장대에서 세 번째 장대로 옮기려 한다.

1. 한 번에 한 개의 원판만을 다른 탑으로 옮길 수 있다.
2. 쌓아 놓은 원판은 항상 위의 것이 아래의 것보다 작아야 한다.

이 작업을 수행하는데 필요한 이동 순서를 출력하는 프로그램을 작성하라. 단, 이동 횟수는 최소가 되어야 한다.

아래 그림은 원판이 5개인 경우의 예시이다.



입력

첫째 줄에 첫 번째 장대에 쌓인 원판의 개수 N (1 ≤ N ≤ 20)이 주어진다.

출력

첫째 줄에 옮긴 횟수 K를 출력한다.

두 번째 줄부터 수행 과정을 출력한다. 두 번째 줄부터 K개의 줄에 걸쳐 두 정수 A B를 빈칸을 사이에 두고 출력하는데, 이는 A번째 탑의 가장 위에 있는 원판을 B번째 탑의 가장 위로 옮긴다는 뜻이다.

하노이 탑 이동 순서(백준 11729번)

풀이

하노이의 탑 문제를 아래와 같은 그림으로 표현해봅시다.

src 탑에 있는 원판 n개를 dst로 옮기는데 med를 사용할 수 있는 상황입니다.
med를 경유지라고 할 수 있습니다.

하노이 탑 이동 순서(백준 11729번)

제일 아래 원판을 dst로 옮기기 위해선 dst가 비어 있어야 하고 이를 위해선 n-1개의 원판을 med로 옮겨야 합니다.

이는 src에 위치한 원판 n-1개를 dst를 경유지로 해서 med로 옮기는 상황입니다.

하노이 탑 이동 순서(백준 11729번)

이제 src에 있는 제일 큰 원판을 dst로 옮깁니다.

그리고 med에 있는 원판 n-1개를 src를 경유지로 해서 dst로 옮겨야 하는 상황입니다.

하노이 탑 이동 순서(백준 11729번)

이제 주어진 문제를 해결하는 함수를 rec이라고 합시다.

이 문제는 src에 n개의 원판이 존재할 때 med를 경유지로 해서 dst로 n개의 원판을 옮기는 문제입니다.
이 문제는 아래와 같은 과정으로 쪼개집니다.

rec(n, src, med, dst)
1. n-1개의 원판은 src에서 dst를 경유지로 해서 med로 옮긴다. -> rec(n-1, src, dst, med)
2. src에 존재하는 가장 큰 원판 하나를 dst로 옮긴다.
3. med에 존재하는 원판 n-1개를 src를 경유지로 해서 dst로 옮긴다. -> rec(n-1, med, src, dst)

하노이 탑 이동 순서(백준 11729번)

rec(n, src, med, dst)
1. n-1개의 원판은 src에서 dst를 경유지로 해서 med로 옮긴다. -> rec(n-1, src, dst, med)
2. src에 존재하는 가장 큰 원판 하나를 dst로 옮긴다.
3. med에 존재하는 원판 n-1개를 src를 경유지로 해서 dst로 옮긴다. -> rec(n-1, med, src, dst)

위 과정을 코드로 짜면 아래와 같습니다.

```
void rec(int n, int src, int dst, int med) {
  rec(n-1, src, med, dst);
  cout << src << ' ' << dst << '\n';
  rec(n-1, med, dst, src);
}
```

하노이 탑 이동 순서(백준 11729번)

재귀 함수는 base case를 잘 정의 하지 않으면 동작하지 않습니다.

방금 코드는 재귀를 끝낼 base case를 정의하지 않아서 스택이 터져서 메모리 초과를 받았습니다.

고친 코드 입니다.

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

void rec(int n, int src, int dst, int med) {
  if(n == 1) {
    cout << src << ' ' << dst << '\n';
    return ;
  }
  rec(n-1, src, med, dst);
  cout << src << ' ' << dst << '\n';
  rec(n-1, med, dst, src);
}

int main() {
  int N; cin >> N;
  cout << (1<<N) - 1 << '\n';
  rec(N, 1, 3, 2);
  return 0;
}
```

하노이 탑 이동 순서(백준 11729번)

시간 복잡도는 어떻게 될까요?
N개를 옮기는 데에 걸리는 시간을 T(N)이라고 놓고 재귀적으로 표현해봅시다.

$$T(N) = 2T(N-1) + 1$$

N개를 옮기는 과정은 N-1개를 두 번 옮기고 제일 큰 원판을 한 번 옮깁니다.

위를 계산 하면 $T(N) = 2^N - 1$이 나옵니다.

분할 정복

분할정복의 문제 해결 방식은 다음과 같습니다.

> 주어진 문제를 둘 이상의 부분 문제로 나눕니다.
>
> 각 부분 문제들에 대한 답을 재귀를 통해 구합니다.
>
> 부분 문제의 답을 통해서 전체 문제의 답을 구합니다.

문제를 분할하다가 문제의 크기가 충분히 작아서 빠르게 답을 구할 수 있는 경우가 되면 해당 문제의 답을 구합니다.

MergeSort와 QuickSort

정렬 문제는 분할정복을 통해 빠르게 해결하는 대표적인 문제입니다.

Merge Sort와 Quick Sort가 분할정복을 사용하는 알고리즘입니다.

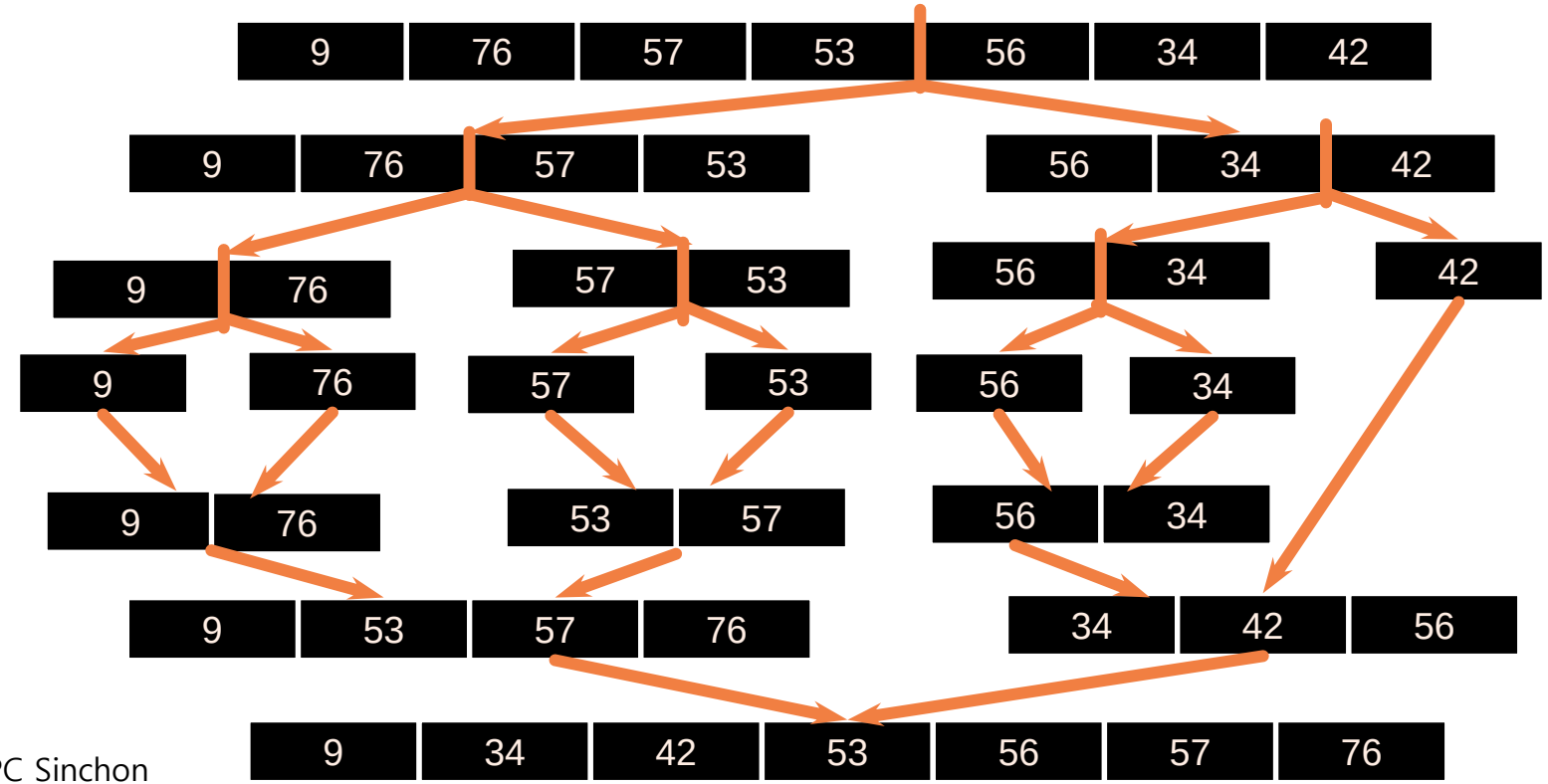
Merge Sort는 주어진 배열을 앞의 절반과 뒤의 절반으로 나눈 뒤에 나눠진 배열들을 정렬하도록 합니다.

그리고 정렬된 부분 배열을 통해 전체 배열의 정렬 결과를 구하는 merge 과정이 있습니다.

Quick Sort는 partition이라는 과정을 통해서 정렬을 진행합니다.

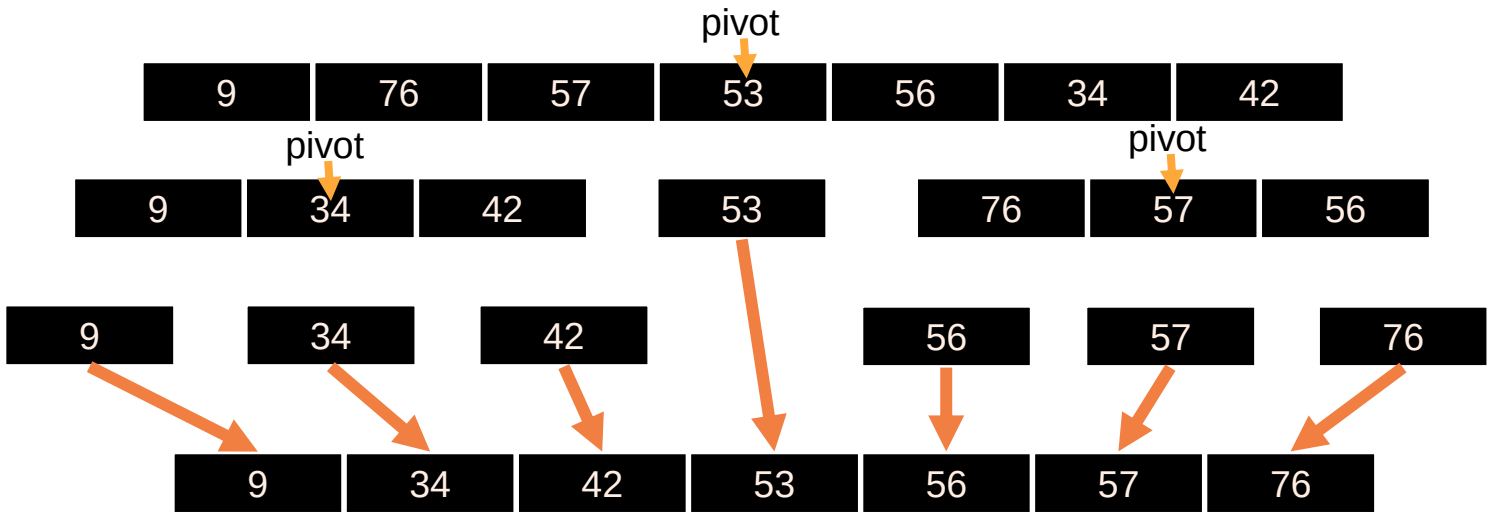
pivot이 될 원소를 하나 정해서 전체 배열을 pivot보다 작은 원소는 pivot의 왼쪽, 큰 원소는 오른쪽에 둡니다. 이런 과정을 통해 정렬을 진행합니다.

MergeSort와 QuickSort



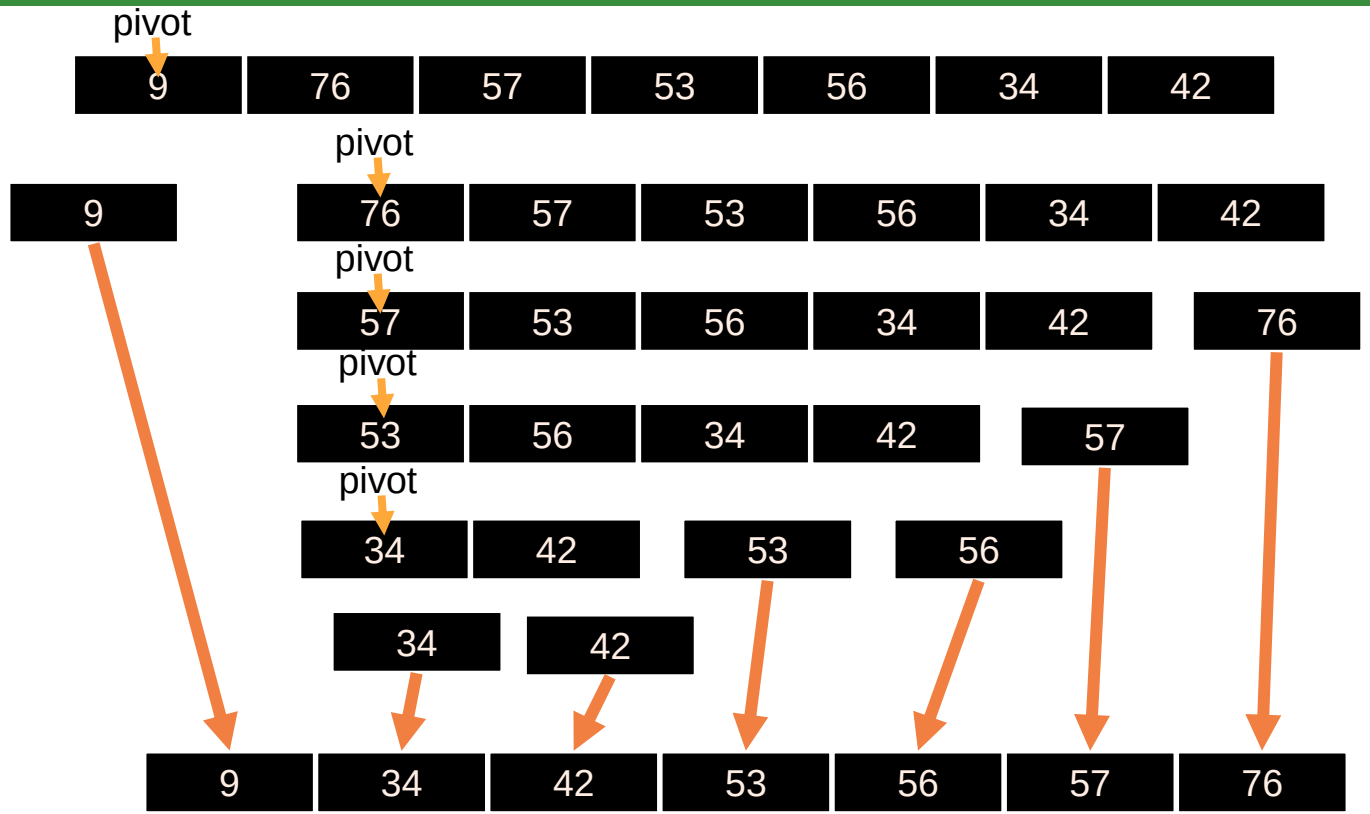
MergeSort와 QuickSort

Nice Pivot



MergeSort와 QuickSort

Bad Pivot



MergeSort와 QuickSort

Merge Sort의 경우 항상 문제를 절반으로 쪼갠다.
Quick Sort는 pivot으로 삼을 원소의 값에 따라서 잘 쪼개질 수도 있고 딱 하나만 줄어들 수도 있다.

-> 문제를 잘 분할하는 것이 중요하다.

Merge Sort는 분할을 진행하고 부분 문제를 풀어준 뒤에 다시 합쳐주는 별도의 과정이 필요하다.
Quick Sort는 partition 과정을 통해서 별도의 병합 과정 없이 전체를 정렬된 상태로 만들어준다.

-> 같은 일을 하더라도 부분 문제들을 병합하는 과정은 설계방식 따라 달라질 수도 있다.

분할정복을 언제 사용할까?

1. Merge Sort와 같이 큰 문제를 동일한 형태의 작은 문제들로 "잘" 쪼갤 수 있을 때
2. 문제의 크기가 충분히 작을 때 쉽게 해결할 수 있을 때
3. 쪼개진 문제들을 통해서 전체 문제를 풀 수 있을 때

해결해야할 문제가 위와 같은 특성들을 가지고 있다면 논리를 각 단계로 쪼개서 설계할 수 있다.

연속합 (백준 1912번)

연속합 성공

☆

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1 초 (추가 시간 없음)	128 MB	84310	26951	18625	30.982%

문제

n개의 정수로 이루어진 임의의 수열이 주어진다. 우리는 이 중 연속된 몇 개의 수를 선택해서 구할 수 있는 합 중 가장 큰 합을 구하려고 한다. 단, 수는 한 개 이상 선택해야 한다.

예를 들어서 10, -4, 3, 1, 5, 6, -35, 12, 21, -1 이라는 수열이 주어졌다고 하자. 여기서 정답은 12+21인 33이 정답이 된다.

입력

첫째 줄에 정수 n(1 ≤ n ≤ 100,000)이 주어지고 둘째 줄에는 n개의 정수로 이루어진 수열이 주어진다. 수는 -1,000보다 크거나 같고, 1,000보다 작거나 같은 정수이다.

출력

연속합 (백준 1912번)

이 문제는 kadane algorithm이라고 O(N)에 푸는 최적 알고리즘이 존재합니다.

여기선 O(NlogN)에 푸는 분할정복 풀이를 알아봅시다.

주어진 수열이 a라고 할 때, 답이 될 수 있는 후보들은 모든 $i, j (i \leq j)$에 대해서 $\sum_i^j a_k$입니다.

a =	57	-58	56	34	-42	9
i=1, j=2	57	53				
i=1, j=4	57	53	-56	34		
i=2, j=5	53	-56	34	-42		
i=3, j=5	-56	34	-42			

연속합 (백준 1912번)

수열이 고정된 상태에서 (i, j)의 부분합을 구하는 건 O(1)에 가능합니다.

가능한 (i,j) 쌍은 $O(n^2)$만큼 있기 때문에 전부 구하는 건 이만큼 걸립니다.

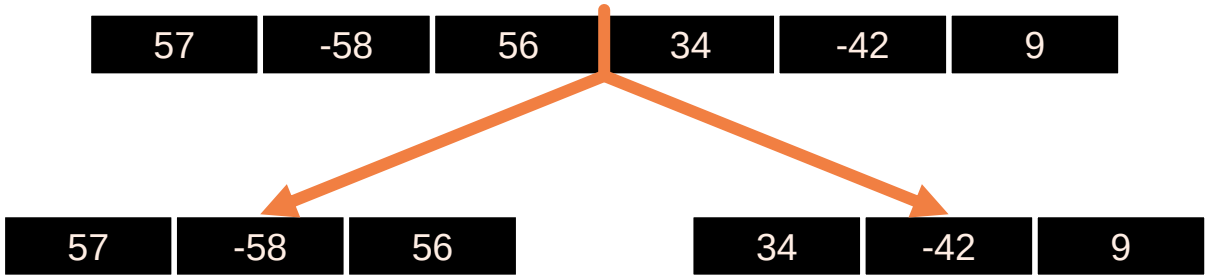
이것보다 빠르게 하기 위해 문제를 쪼개봅시다.

연속합 (백준 1912번)

우리가 해결해야 되는 문제는 어떤 수열이 주어지면 해당 수열에서 연속합 중에서 가장 큰 것을 구하는 것입니다.

이 문제는 수열을 반으로 쪼개고 쪼갠 수열에 대해서도 똑같이 적용될 수 있는 문제입니다.

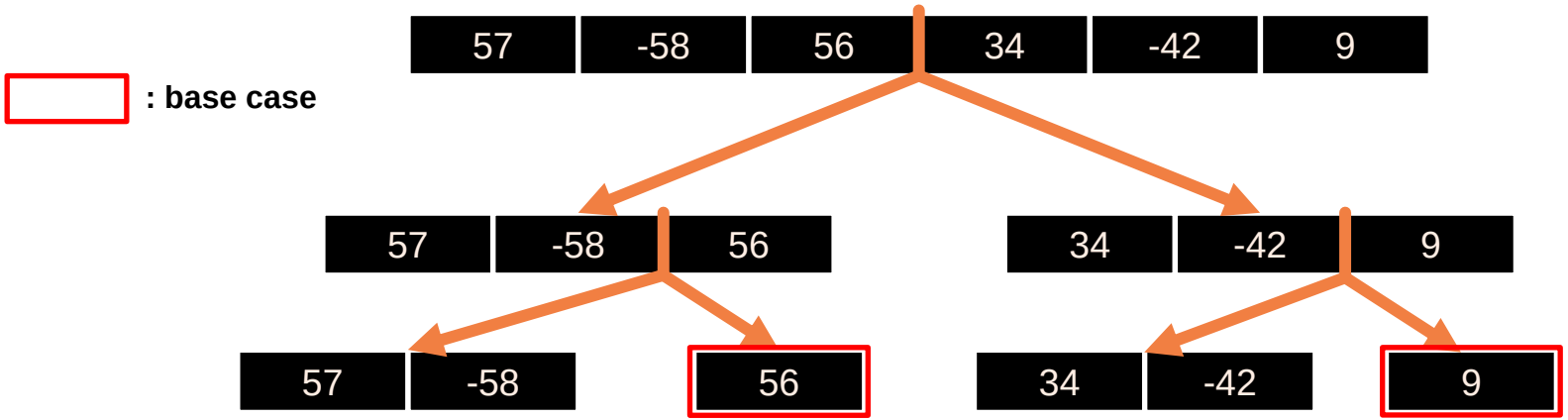
따라서 수열을 반으로 쪼개면서 부분 문제들로 쪼개는 것이 가능합니다.



연속합 (백준 1912번)

전체 문제를 크기가 절반인 두 개의 부분 문제로 분할했습니다.

base case를 생각해봅시다. 수열의 크기가 1이라면 존재하는 연속합이 정확히 하나이기 때문에 답이 바로 나옵니다.
따라서, 수열의 크기가 1일 경우를 base case라고 합시다.

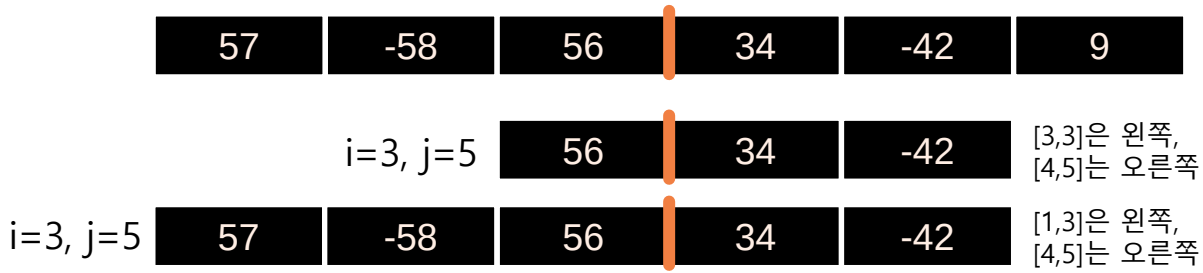


연속합 (백준 1912번)

base case도 정의 했으니 부분 문제들로부터 전체 문제의 답을 구하는 방법만 찾으면 알고리즘이 완성됩니다.

전체 수열의 길이가 N이라면 왼쪽의 N/2짜리 수열에 있는 연속합과 오른쪽의 N/2짜리 수열에 있는 연속합을 부분문제로 정의했습니다.

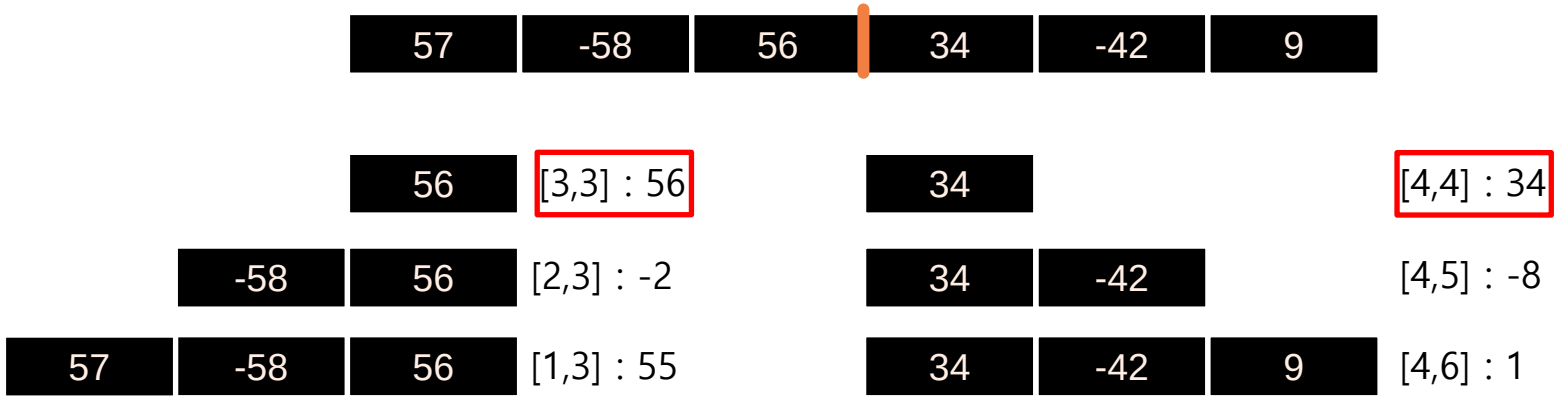
우리가 알아봐야할 후보들은 왼쪽 절반과 오른쪽 절반에 걸쳐 있는 연속합입니다.



왼쪽과 오른쪽에 걸쳐서 존재하는 연속합은 중간을 기준으로 왼쪽 절반의 것과 오른쪽 절반의 것으로 구분됩니다.

연속합 (백준 1912번)

이를 통해서 걸쳐서 존재하는 연속합 중에서 최댓값은 중간에서부터 왼쪽으로 뻗어 있는 연속합 중 최댓값, 중간에서부터 오른쪽으로 뻗어 있는 연속합 중 최댓값을 합한 값입니다.



왼쪽으로 뻗는 합 중 최대는 56, 오른쪽으로 뻗는 합 중 최대는 34고 이 둘을 합친 게 중간에 걸쳐서 존재하는 연속합 중 최대입니다.

연속합 (백준 1912번)

이걸로 분할정복 풀이가 완성됐습니다. 정리하죠.

1. 왼쪽 절반에 대한 답(left_mx)과 오른쪽 절반에 대한 답(right_mx)을 구합니다.

2. 중간을 기준으로 왼쪽에 이어지는 연속합 중 최대(left_sum)과 오른쪽으로 이어지는 연속합 중에서 최대(right_sum)를 구합니다.

3. 구한 답 세가지 중에서 최대를 리턴합니다. max(left_mx, right_mx, left_sum + right_sum)

base case는 수열의 길이가 1일 때로 하나 있는 원소가 답입니다.

연속합 (백준 1912번) - 시간복잡도

크기가 N인 문제를 푸는 데에 걸리는 시간을 T(N)이라고 합시다.

수열의 왼쪽 절반을 부분 문제로 하나, 오른쪽 절반을 부분 문제로 하나 만들고 그것을 풀었습니다.

중간에 걸쳐 있는 연속합 중의 최대를 구하는 데 전체 수열을 한번 훑었습니다. O(N)

$$T(N) = 2T(N/2) + O(N)$$

이 식은 머지 소트의 것과 동일합니다. 따라서, T(N) = O(NlogN).

곱셈 (백준 1629번)

곱셈 성공

★

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
0.5 초 (추가 시간 없음)	128 MB	41928	10810	7954	25.439%

문제

자연수 A를 B번 곱한 수를 알고 싶다. 단 구하려는 수가 매우 커질 수 있으므로 이를 C로 나눈 나머지를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 A, B, C가 빈 칸을 사이에 두고 순서대로 주어진다. A, B, C는 모두 2,147,483,647 이하의 자연수이다.

출력

첫째 줄에 A를 B번 곱한 수를 C로 나눈 나머지를 출력한다.

곱셈 (백준 1629번)

나머지 연산에서는 다음이 성립합니다.

$$(A \bmod C)(B \bmod C) \equiv AB \bmod C$$

따라서 A를 B번 곱하면 답을 구할 수 있습니다.

-> B가 굉장히 크기 때문에 시간초과

곱셈 (백준 1629번)

$$A^B = \underbrace{AAA\cdots AAA}_{B}$$
$$= \underbrace{A\cdots AA\cdots A}_{B/2 \qquad B/2}$$

A를 B번 곱하는 것은 A를 B/2번 곱한 것 두개를 곱해도 결과가 같습니다.

-> 분할정복이 가능하다

B가 2로 나누어 떨어지지 않는다면? B에서 1을 빼주고 계산한 결과에 A를 한 번 곱해주면 됩니다.

곱셈 (백준 1629번)

base case를 정의합시다.

B가 0이라면 문제의 답이 항상 1이라는 것을 바로 알 수 있습니다.
-> base case

A의 B승을 C로 나눈 결과를 돌려주는 함수를 ipow라고 합시다.

```
def ipow(A, B):
    if B is zero:
        return 1
    x <- ipow(A, B/2)
    x <- x * x
    if B is odd:
        x <- x * a
    return x
```

곱셈 (백준 1629번) – 시간복잡도

ipow의 시간복잡도는 B의 크기에 영향을 받습니다.
B=N이라 놓고 ipow(A, N)의 시간복잡도를 T(N)이라고 합시다.

$$T(N) = T(N/2) + O(1)$$

이는 $T(N) = O(\lg N)$입니다.

Counting Inversions (백준 10090번)

Counting Inversions

성공 출처 다국어

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1 초	256 MB	1752	706	484	40.468%

문제

A permutation of integers from 1 to n is a sequence a_1, a_2, ..., a_n, such that each integer from 1 to n is appeared in the sequence exactly once.

Two integers in a permutation form an inversion, when the bigger one is before the smaller one.

As an example, in the permutation 4 2 7 1 5 6 3, there are 10 inversions in total. They are the following pairs: 4–2, 4–1, 4–3, 2–1, 7–1, 7–5, 7–6, 7–3, 5–3, 6–3.

Write program invcnt that computes the number of the inversions in a given permutation.

입력

The value for the number n is written on the first line of the standard input. The permutation is written on the second line: n numbers, delimited by spaces. ($2 \leq n \leq 1000000$)

출력

Write the count of inversions on the standard output.

Counting Inversions (백준 10090번)

수열 a의 inversion이란 수열 i<j면서 a_i>a_j인 (i,j)를 말합니다.

(i,j) 쌍은 $\frac{n(n-1)}{2}$ 개 존재하기 때문에 전부 검사하는 방법은 $O(n^2)$으로 시간초과

-> 분할정복으로 시간을 줄이자

Counting Inversions (백준 10090번)

문제를 다시 정의하자.

길이 n인 순열 a가 주어졌을 때 a의 inversion 개수를 계산하는 함수 inv_count(a)

a에 존재하는 inversion (i,j)를 세가지 종류로 나누자.

1. i, j 모두 a의 왼쪽 절반에 존재하는 경우 (i, j <= n/2)
2. i, j 모두 a의 오른쪽 절반에 존재하는 경우 (i,j > n/2)
3. i는 왼쪽, j는 오른쪽 절반에 존재하는 경우

Counting Inversions (백준 10090번)

1. i, j 둘 다 왼쪽 절반인 [1, 3]에 존재하는 inversion은 (1, 2)입니다.
2. i, j 둘 다 오른쪽 절반인 [4, 7]에 존재하는 inversion은 (5, 7), (6, 7)입니다.
3. i는 왼쪽, j는 오른쪽에 위치한 inversion은 (1, 4), (1, 7), (2, 4), (3, 4), (3, 5), (3, 6), (3, 7)입니다.

이것으로 순열에 있는 전체 inversion을 구할 수 있습니다.

Counting Inversions (백준 10090번)

왼쪽에 존재하는 inversion과 오른쪽에 존재하는 inversion은 순열을 반으로 쪼개서 재귀적으로 구하면 됩니다.

base case는 순열의 길이가 1일 때로 inversion이 없습니다.

i는 왼쪽에 존재하고 j는 오른쪽에 존재하는 inversion을 빠르게 구할 방법을 찾으면 분할정복을 통해 문제를 빠르게 해결할 수 있습니다.

Counting Inversions (백준 10090번)

아이디어
inversion만 세지 말고 정렬도 같이 하자.
그리고 Merge Sort에서 두 정렬된 배열을 합쳐주는 과정을 inversion을 세는 데에 사용하자.

왼쪽 절반의 인덱스들은 오른쪽 절반의 인덱스보다 작다는 것은 보장되어 있습니다. (i < j)

그러면 이제 오른쪽에 위치한 원소들마다 왼쪽의 원소 중에서 자신보다 큰 원소를 세주면 inversion입니다.

-> naive : $O(n^2)$

Counting Inversions (백준 10090번)

왼쪽 절반을 l, 오른쪽 절반을 r이라고 합시다.
Merge 과정은 다음과 같은 과정을 반복해서 원래 수열 a를 정렬합니다.

1. l[i], r[j]를 비교해서 더 작은 것을 a[k]에 놓습니다.
2. l[i]를 놓은 뒤에는 i,k를 1 늘리고, r[j]를 놓았다면 j,k를 1 늘립니다.
3. 위 과정을 반복한 뒤 l, r 중 남은 것은 그대로 a에 붙입니다.

Counting Inversions (백준 10090번)

Merge 과정에서 l[i] > r[j]인 경우를 생각합시다.

이 때 l은 정렬된 상태이기 때문에 l[i], l[i+1], … , l[n/2]는 전부 r[j]보다 큽니다.

따라서 r[j]로 인해 발생하는 걸쳐진 inversion의 수는 n/2 – l + 1이라는 것을 알 수 있습니다.

반대로 l[i] < r[j]인 경우엔 l[1], l[2], … , l[i]까지 전부 r[j]보다 작습니다.

따라서 r[j]로 인해 발생하는 inversion은 없습니다.

이것으로 O(n)에 양쪽에 걸쳐 존재하는 inversion의 수를 셀 수 있습니다.

Counting Inversions (백준 10090번) - 시간복잡도

inversion을 세는 것과 동시에 정렬을 하기 때문에 적어도 merge sort만큼은 시간이 걸립니다.

그리고 merge를 하면서 동시에 inversion을 세기 때문에 merge sort와 시간복잡도는 동일합니다.

카라츠바 곱셈

PS에서 정수 곱셈은 상수 연산이라고 생각합니다. 이는 컴퓨터가 32비트 정수, 64비트 정수 연산을 지원하기 때문입니다.

매우 큰 정수는 그렇지 않습니다.

n자리 정수 두개를 곱한다고 할 때 우리가 일반적으로 사용하는 곱셈 알고리즘은 $O(n^2)$입니다.

```
   1234
X  1111
   1234
  12340
 123400
1234000
1370974
```

카라츠바 곱셈

분할정복을 이용하면 곱셈을 빠르게 하는 것이 가능합니다.

문제를 정의합시다. N자리 정수 a, b를 곱한 결과를 우리는 알고 싶습니다.
이를 수행하는 함수를 mult_fast(a,b)라고 합시다.

a, b를 각각 상위 N/2자리, 하위 N/2자리로 나눈 것을 $(a_1, a_2), (b_1, b_2)$라고 합시다.

$$a = a_1 \times 10^{N/2} + a_2$$
$$b = b_1 \times 10^{N/2} + b_2$$

$$\begin{aligned} a \times b &= (a_1 \times 10^{N/2} + a_2)(b_1 \times 10^{N/2} + b_2) \\ &= a_1 b_1 \times 10^N + (a_2 b_1 + a_1 b_2) \times 10^{N/2} + a_2 b_2 \end{aligned}$$

카라츠바 곱셈

$$a \times b = a_1 b_1 \times 10^N + (a_2 b_1 + a_1 b_2) \times 10^{N/2} + a_2 b_2$$

위 식 대로 a와 b의 곱셈을 구한다 쳐도 $a_1 b_1, a_1 b_2, a_2 b_1, a_2 b_2$로 N/2자리 정수 곱셈을 4번 수행해야 합니다.

N자리 정수 곱셈을 하는 데에 걸리는 시간이 T(N)이라고 합시다.

T(N) = 4T(N/2) + O(N)

여전히 $O(N^2)$입니다.

카라츠바 곱셈

$$a \times b = a_1 b_1 \times 10^N + (a_2 b_1 + a_1 b_2) \times 10^{N/2} + a_2 b_2$$

위 식에서 아래와 같이 치환을 합시다.

$$\begin{aligned} c_1 &= a_1 b_1 \\ c_2 &= a_2 b_1 + a_1 b_2 \\ c_3 &= a_2 b_2 \end{aligned}$$

c_1, c_3는 N/2자리 정수 곱셈 한번씩으로 구할 수 있습니다.

이제 c_2를 빠르게 구해야 합니다.

카라츠바 곱셈

$$(a_1 + a_2) \times (b_1 + b_2) = a_1 b_1 + a_1 b_2 + a_2 b_1 + a_2 b_2$$
$$= c_1 + c_2 + c_3$$

c_2는 (a_1+a_2)와 $(b_1 + b_2)$를 곱한 것에서 c_1, c_3를 빼면 구할 수 있습니다.

덧셈과 뺄셈은 $O(N)$의 연산이기 때문에 시간복잡도에는 영향을 끼치지 않습니다.

카라츠바 곱셈 - 시간복잡도

시간복잡도를 계산해봅시다.

N자리 정수 a,b를 곱하려고 합니다. 이를 위해서 각 수를 N/2자리 정수 $(a_1, a_2), (b_1, b_2)$로 나눴습니다.

그리고 N/2자리 정수 두개의 곱셈을 총 세 번 실행합니다.

$$T(N) = 3T(N/2) + O(N)$$

N/2자리 정수 곱셈을 네 번보다는 줄였는데 시간복잡도의 계산이 어렵습니다.

마스터 정리

이제까지 수행시간을 T(N)이라고 놓고 이를 점화식처럼 표시했습니다.

마스터 정리는 이렇게 점화식처럼 표현했을 때 몇가지 경우에 대해서 쉽게 T(N)을 구하는 정리입니다.

아래와 같은 형식의 점화식이 나왔을 때 사용이 가능합니다.

$$T(n) = aT(\frac{n}{b}) + f(n) \quad (a >= 1, b > 1)$$

여기서 f(n)은 asymptotically positive function이어야 합니다.

말이 어려운데 어떤 N을 기준으로 N보다 크거나 같은 모든 n에 대해서 f(n)이 0보다 크다는 뜻입니다.

마스터 정리

$$T(n) = aT(\frac{n}{b}) + f(n) \quad (a >= 1, b > 1)$$

우리가 보통 만나게 될 점화식에선 f(n)을 아래처럼 바꿔도 적용이 가능합니다.

$$T(n) = aT(\frac{n}{b}) + \Theta(n^k \log^p n) \quad (a >= 1, b > 1)$$

이 점화식은 $log_b a$와 k의 관계에 따라서 크게 세가지 경우로 나눠서 생각할 수 있습니다.

마스터 정리

$$T(n) = aT(\frac{n}{b}) + \Theta(n^k \log^p n) \quad (a >= 1, b > 1)$$

Case 1. $log_b a > k$ $\quad T(n) = \Theta(n^{log_b a})$

Case 2. $log_b a = k$

- Case 2-1. $p > -1$ $\quad T(n) = \Theta(n^k log^{p+1} n)$
- Case 2-2. $p = -1$ $\quad T(n) = \Theta(n^k loglogn)$
- Case 2-3. $p < -1$ $\quad T(n) = \Theta(n^k)$

Case 3. $log_b a < k$

- Case 3-1. $p \geq 0$ $\quad T(n) = \Theta(n^k log^p n)$
- Case 3-2. $p < 0$ $\quad T(n) = \Theta(n^k)$

마스터 정리 - 예시

$$T(n) = aT(\frac{n}{b}) + \Theta(n^k \log^p n) \quad (a >= 1, b > 1)$$

Merge Sort와 Inversion Counting

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

a=2, b=2, k=1, p=0인 경우 입니다.
$log_2 2 = 1$이므로 Case 2이고, p=0이기 때문에 Case 2-1입니다.
따라서, $T(n) = \Theta\left(n^k log^{p+1} n\right) = \Theta(n^1 log^1 n)$

마스터 정리 - 예시

$$T(n) = aT(\frac{n}{b}) + \Theta(n^k \log^p n) \quad (a >= 1, b > 1)$$

분할정복을 이용한 거듭제곱

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$$

a=1, b=2, k=0, p=0인 경우 입니다.
$log_2 1 = 0$이므로 Case 2이고, p=0이기 때문에 Case 2-1입니다.
따라서, $T(n) = \Theta\left(n^k log^{p+1} n\right) = \Theta(n^0 log^1 n)$

마스터 정리 - 예시

$$T(n) = aT(\frac{n}{b}) + \Theta(n^k \log^p n) \quad (a >= 1, b > 1)$$

카라츠바 곱셈

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$$

a=3, b=2, k=1, p=0인 경우 입니다.
$log_2 3 > 1$이므로 Case 1입니다.
따라서, $T(n) = \Theta(n^{log_b a}) = \Theta(n^{log_2 3})$

나이브한 정수 곱셈의 시간복잡도보다 줄어들었습니다.

Problem Set

- 필수 문제
  - 1662 압축
  - 1629 곱셈
  - 1912 연속합
  - 10090 Counting Inversions

Problem Set

- 연습 문제
  - 1780 종이의 개수
  - 2630 색종이 만들기
  - 4779 칸토어 집합
  - 2447 별찍기 – 10
  - 17829 222-풀링
  - 14601 샤워실 바닥 깔기(Large)
  - 18222 투에-모스 문자열
  - 10830 행렬제곱
  - 15712 등비수열
  - 21870 시철이가 사랑한 GCD
  - 4256 트리
  - 14727 퍼즐 자르기
  - 2104 부분배열 고르기
  - 11004 K번째 수(Quick Selection)
  - 2261 가장 가까운 두 점

“Any Question?”