2025 겨울방학 알고리즘 스터디

분할 정복 / 백트래킹

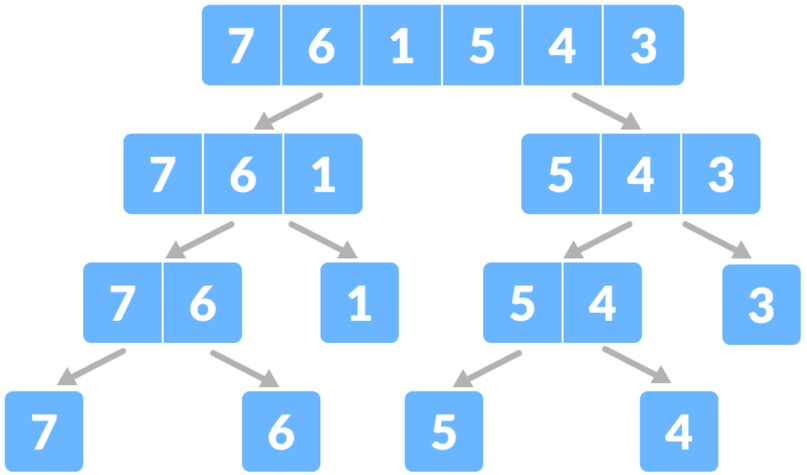
컴퓨터 공학과 20230546 서보경

목차

1. 분할 정복 이란?
2. 분할 정복 응용 – 빠른 거듭 제곱
3. 백트래킹 이란?
4. 다음 순열

분할 정복 이란?

분할 정복(Divide and Conquer)은 문제를 작은 부분으로 나누어 해결하는 알고리즘 기법이다. 가장 큰 특징은 문제를 분할(Divide) 하고, 해결된 부분 문제를 정복(Conquer) 하여 최종 결과를 얻는다는 점이다. 일반적으로 재귀적으로 동작하며, 더 이상 나눌 수 없을 때 기저 사례(Base Case)를 이용해 직접 해결한다.
대표적인 예시로 병합 정렬(Merge Sort), 퀵 정렬(Quick Sort), 이분 탐색(Binary Search) 등이 있으며, 문제를 효율적으로 줄여나가기 때문에 시간 복잡도를 개선하는 데 유리하다.



대표적인 분할 정복의 예시인 병합 정렬

분할 정복

86	60	28	65	48	86	71

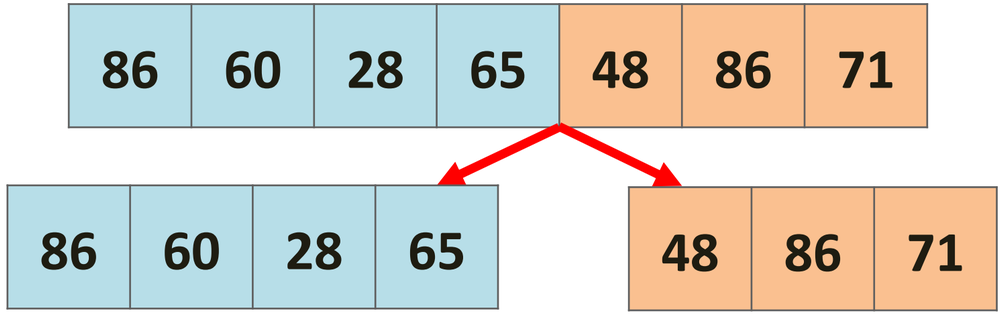
자료 구조 수업에서도 배운 대표적 정렬 기법인 병합 정렬을 보자.

병합 정렬은 대표적인 분할 정복의 예시 중 하나이다.

분할 과정은 배열이 크기 1(기저 조건)을 가질 때 까지 계속 분할해 나가는 작업이다.

정복 과정은 배열이 기저 조건에 도달 했을 때 왼쪽과 오른쪽을 순서에 맞춰 합치는 과정이다.
이때, 두 배열의 원소가 다르다면 각 배열의 포인터를 움직여 가면서 작은 원소를 우선으로 합쳐준다.

분할 정복

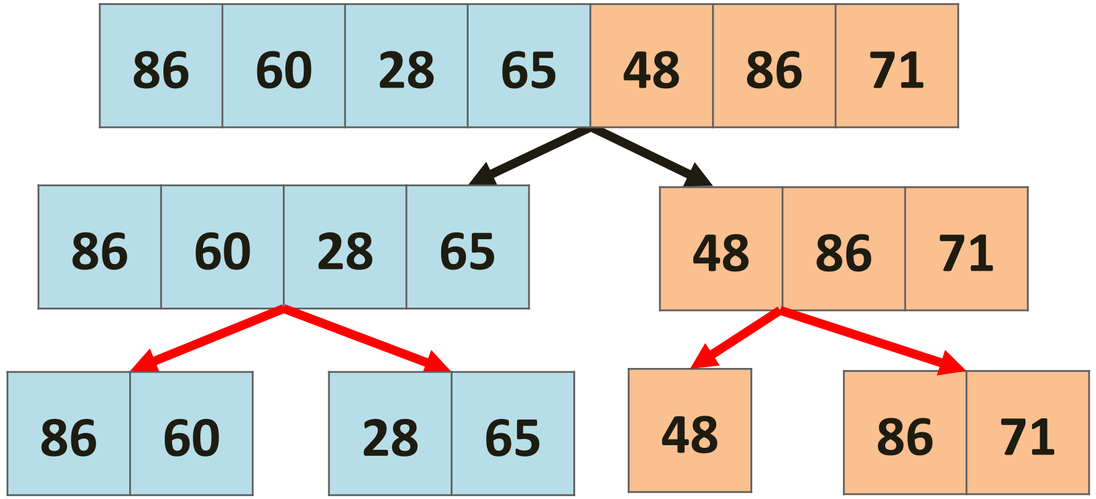


먼저, 배열을 반으로 나눈다.

이러면 크기가 4인 배열과 3인 배열로 분할 된다.

이러한 분할 작업을 크기가 1일 때 까지 반복한다. (분할 정복의 “분할”)

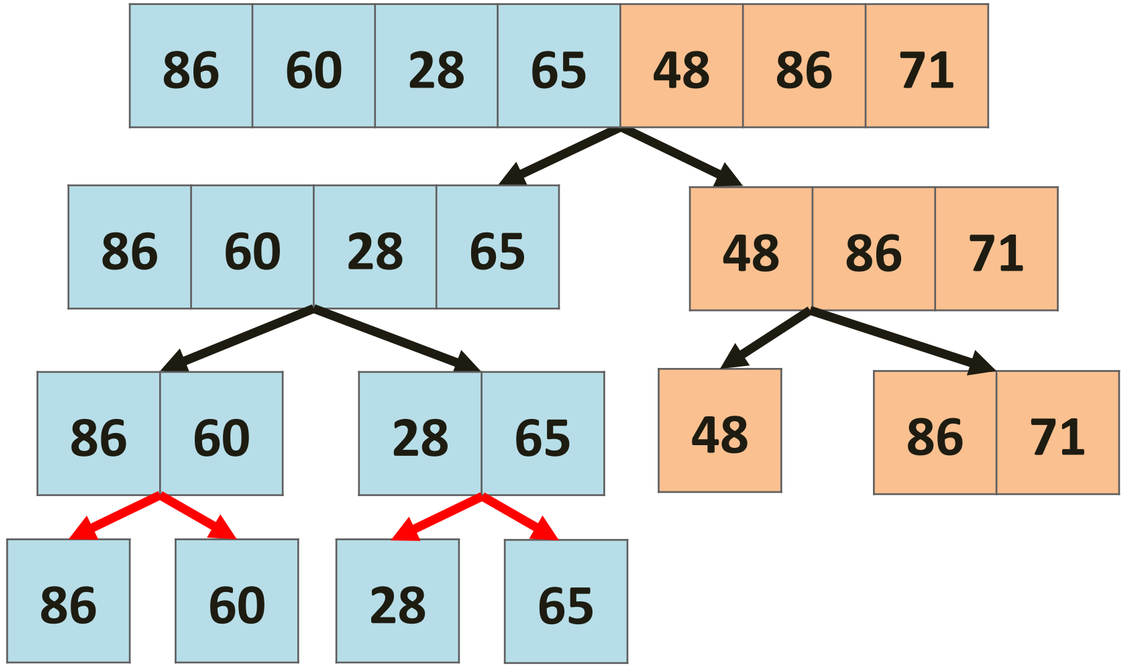
분할 정복



왼쪽 배열의 크기는 현재 4이다. 아직 기저 조건에 도달 하지 않았으므로 또 반으로 분할한다.

오른쪽 배열의 크기는 현재 3이다. 아직 기저 조건에 도달 하지 않았으므로 또 반으로 분할한다.

분할 정복

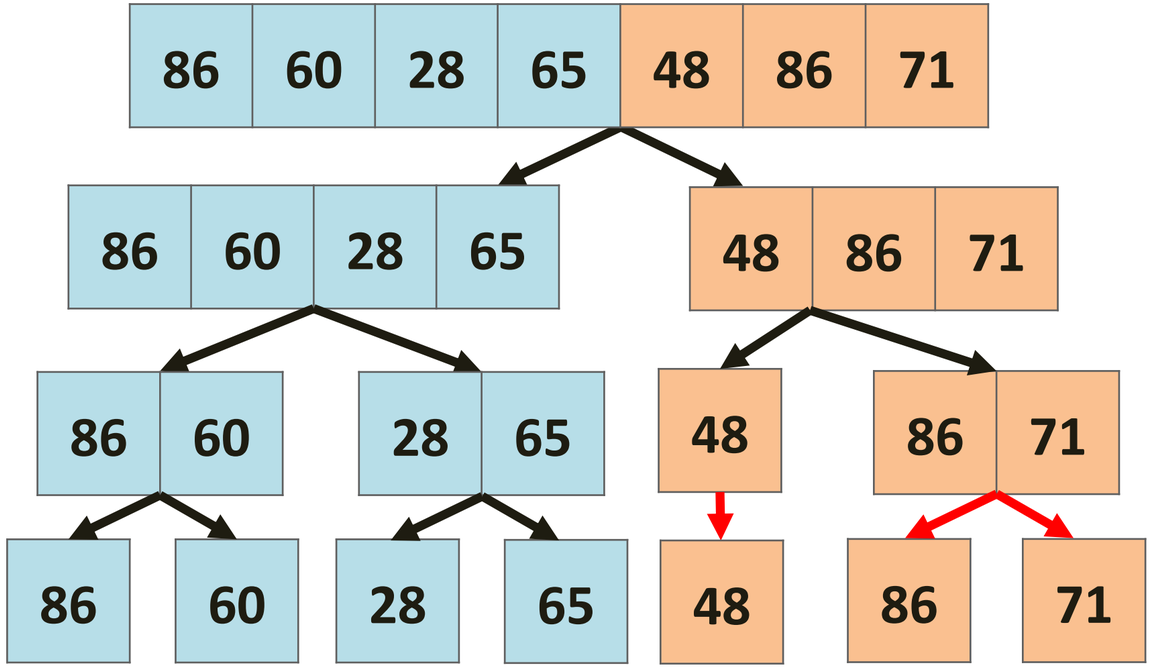


먼저 파란색 배열을 보자.

파란색의 배열들은 둘 다 크기가 2이다.

이는 아직 기저조건이 도달하지 않았음을 의미하므로 다시 분할을 해준다.

분할 정복



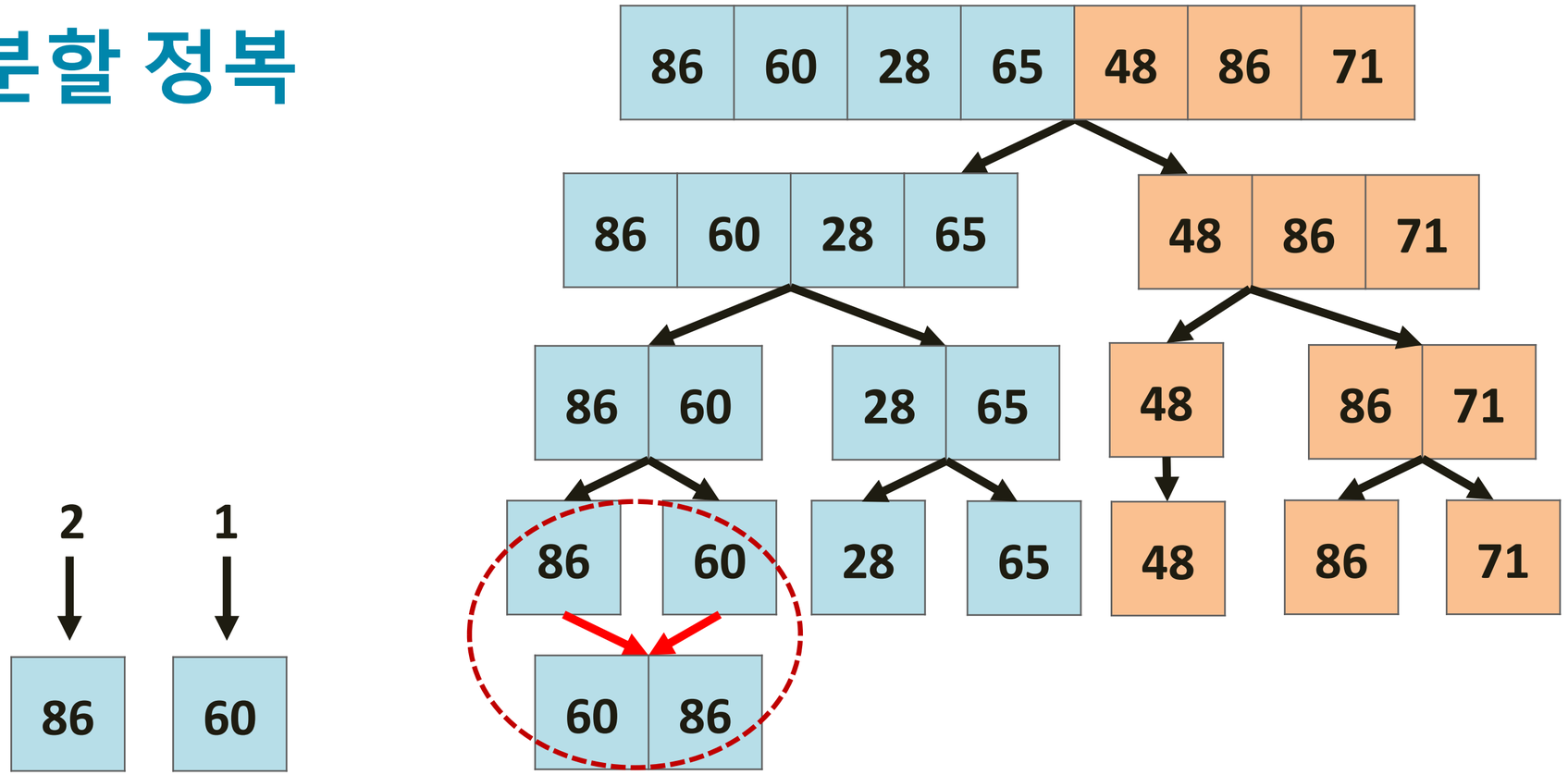
다음으로 주황색 배열을 보자.

주황색 배열의 왼쪽 배열([48])은 이미 기저조건에 도달한 상태이다.

그러나 오른쪽 배열은 아직 기저조건에 도달하지 않았다. 고로 분할을 해준다.

분할 후, 왼쪽과 오른쪽 값을 합쳐서 정복을 하는 논리이기 때문에 왼쪽 배열은 대기한다.

분할 정복

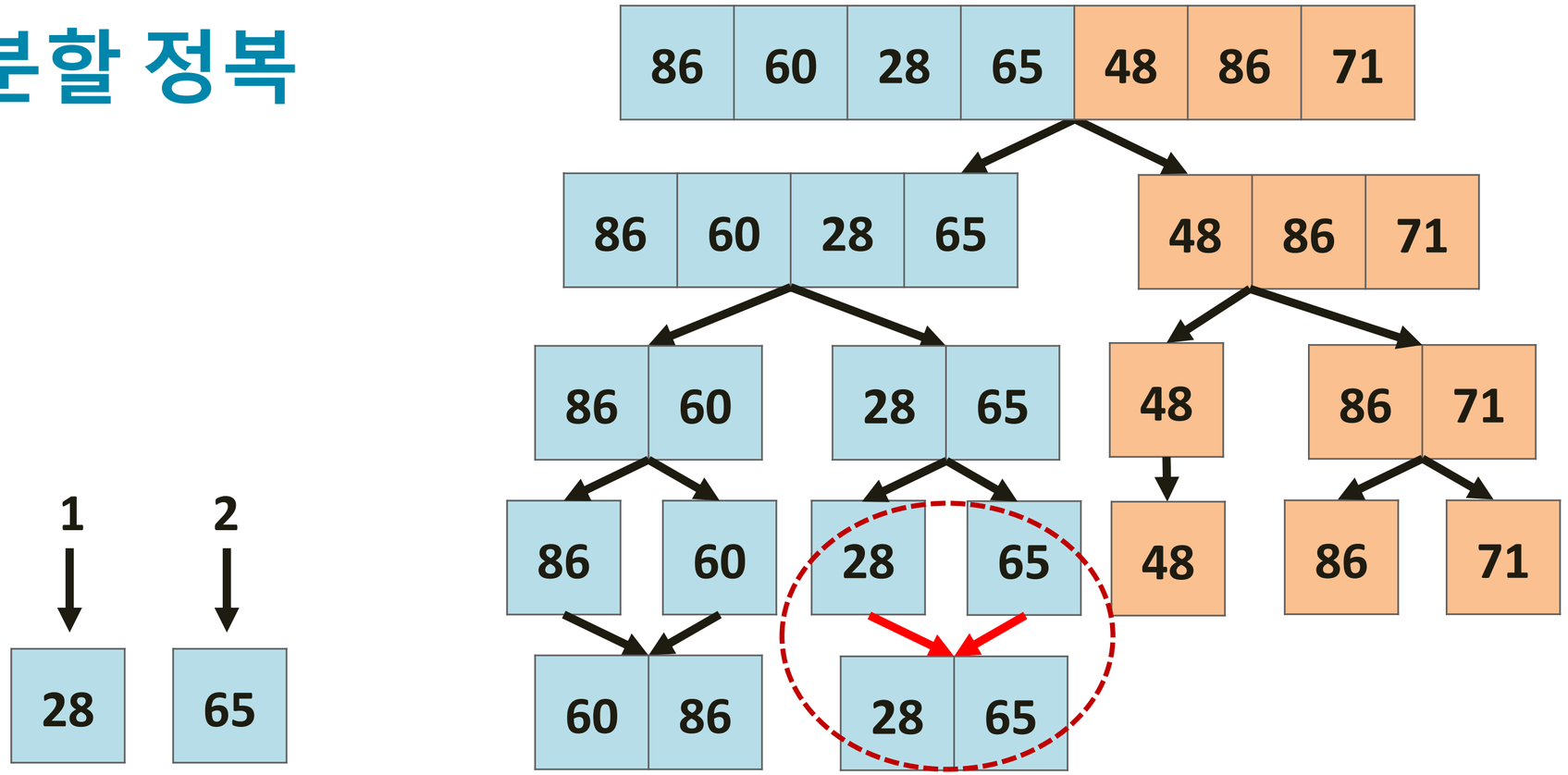


기저에 도달한 86과 60을 보자.

두 원소 전부 기저에 도달하였기 때문에 정복을 할 수 있는 조건이 되었다.

60이 86보다 앞서기 때문에 [60,86]의 형태로 정복하여 배열을 반환한다.

분할 정복

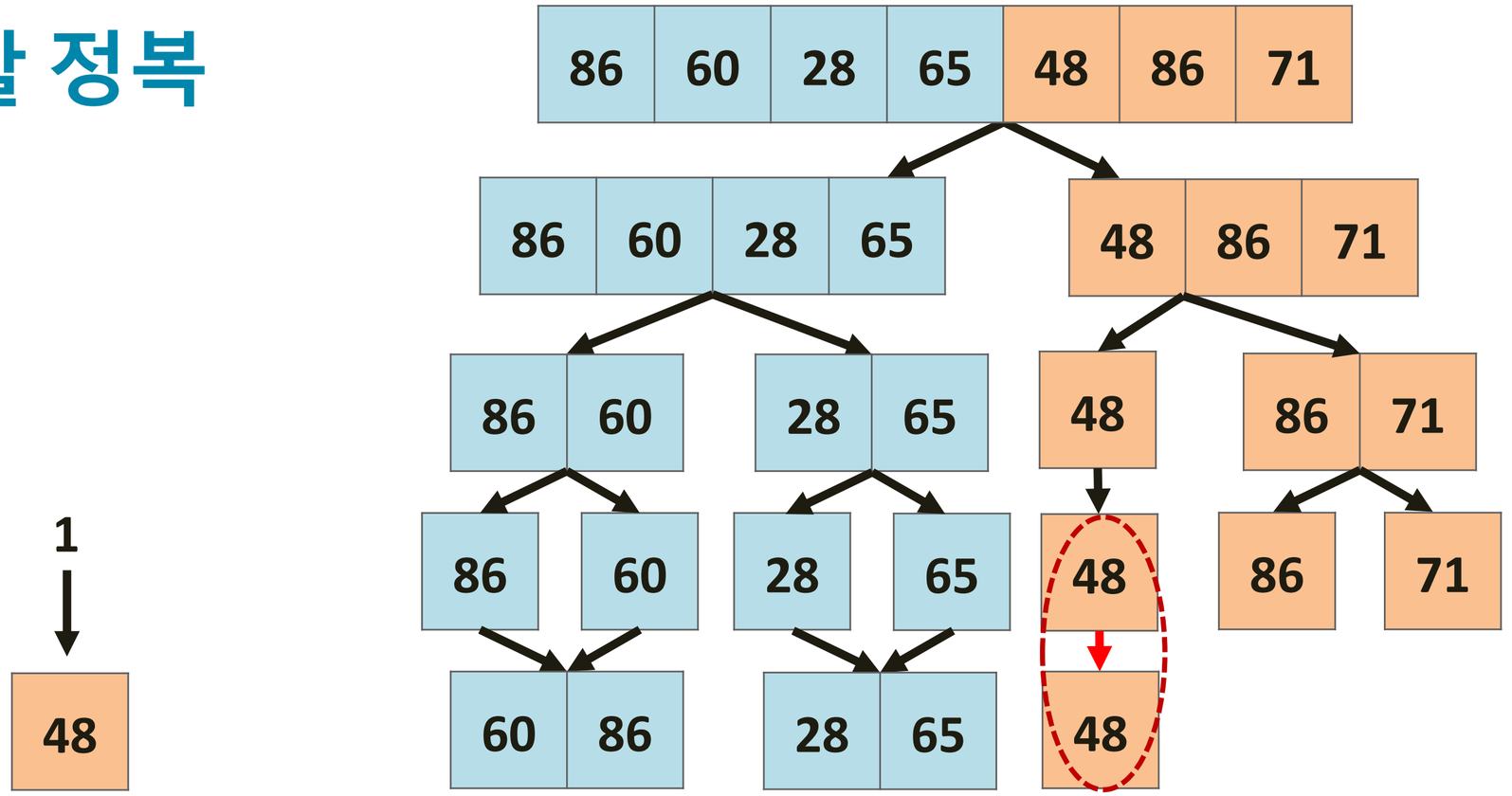


기저에 도달한 28과 65을 보자.

두 원소 전부 기저에 도달하였기 때문에 정복을 할 수 있는 조건이 되었다.

28이 65보다 앞서기 때문에 [28,65]의 형태로 정복하여 배열을 반환한다.

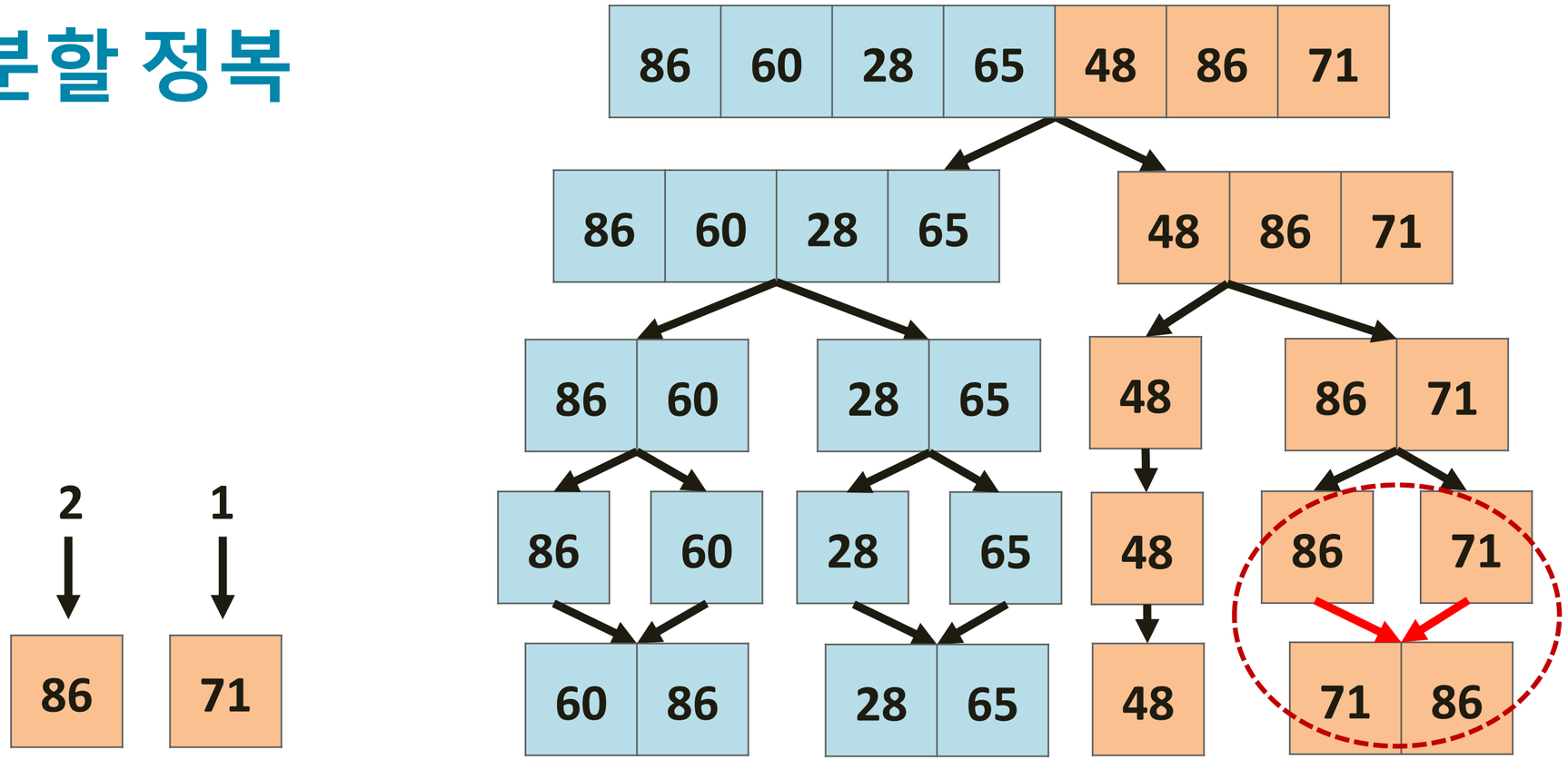
분할 정복



기저에 도달한 48원소는 오른쪽 배열 [86,71]가 정복이 끝나길 기다리고 있다.
([48,86,71]에서 분할되어 왼쪽은 [48], 오른쪽은 [86,71] 이기때문에)

고로 아무것도 하지 않는다.

분할 정복

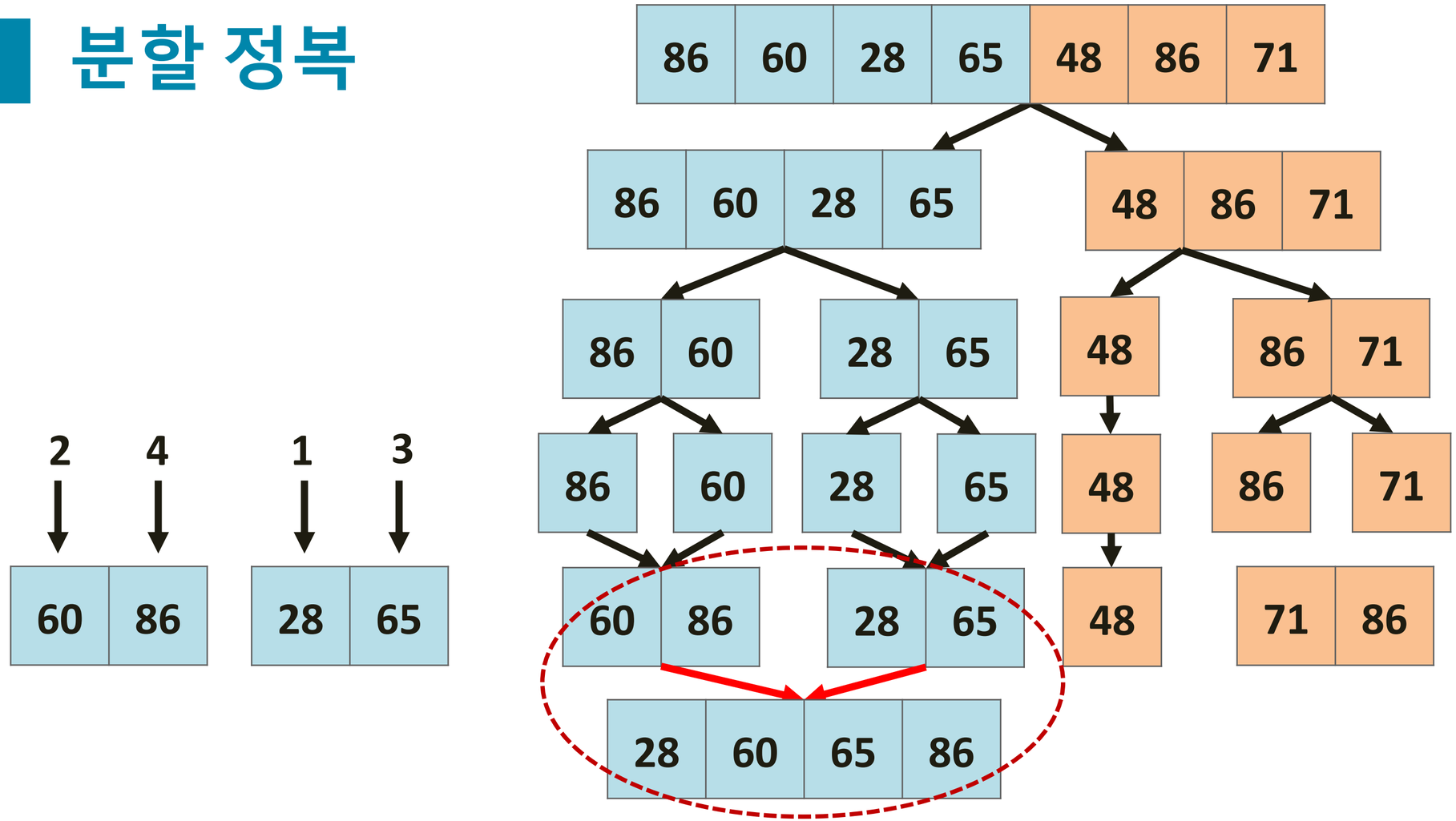


기저에 도달한 71와 86을 보자.

두 원소 전부 기저에 도달하였기 때문에 정복을 할 수 있는 조건이 되었다.

71가 86보다 앞서기 때문에 [71,86]의 형태로 정복하여 배열을 반환한다.

분할 정복

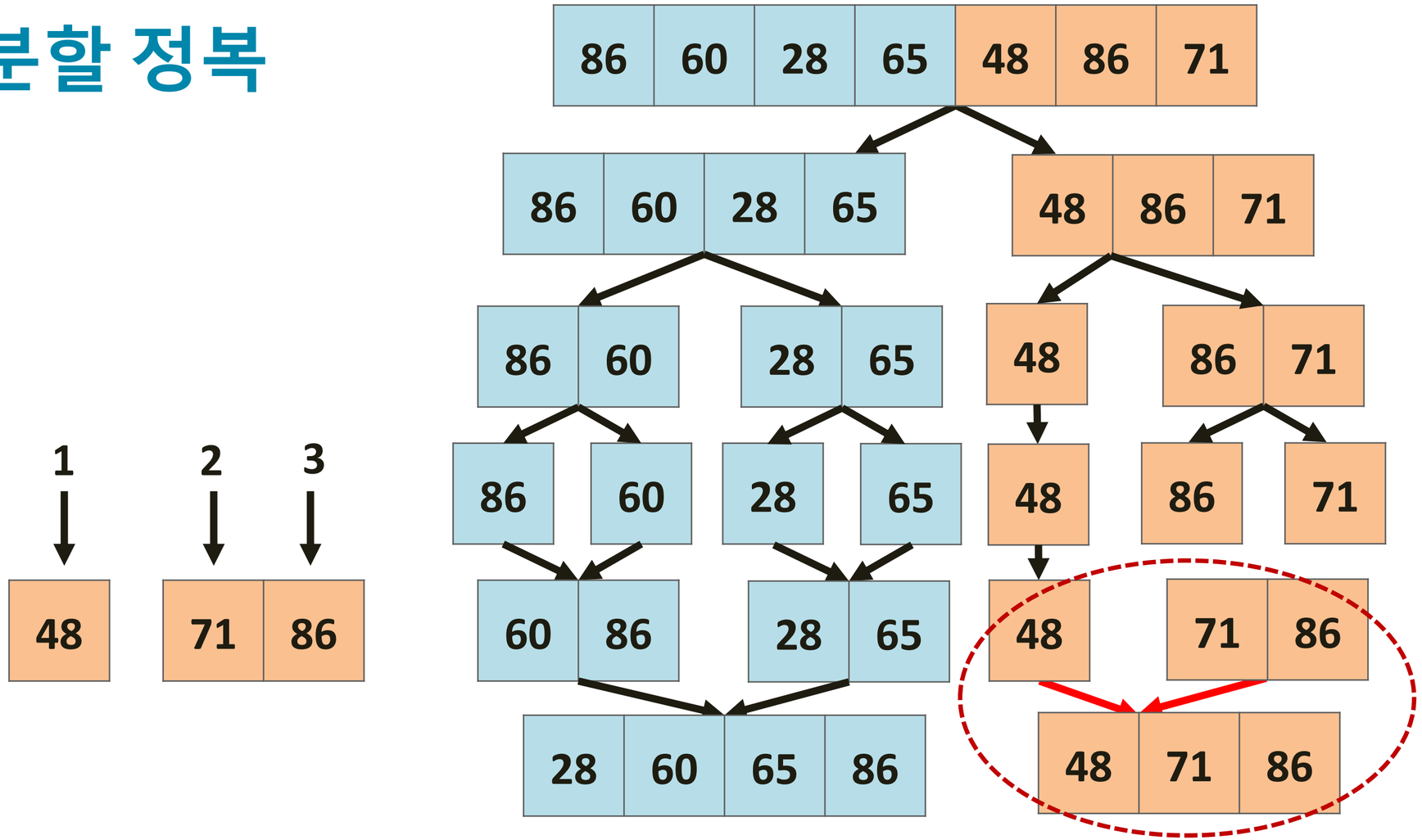


정복을 한 두 파란색 배열을 보자.

각각 [60,86] [28,65]이다. 앞서 언급한 것 처럼 포인터 연산을 통해 두 배열의 작은 원소부터 합쳐야 한다.

고로 [28, 60, 65, 86]으로 합쳐진다(정복된다).

분할 정복

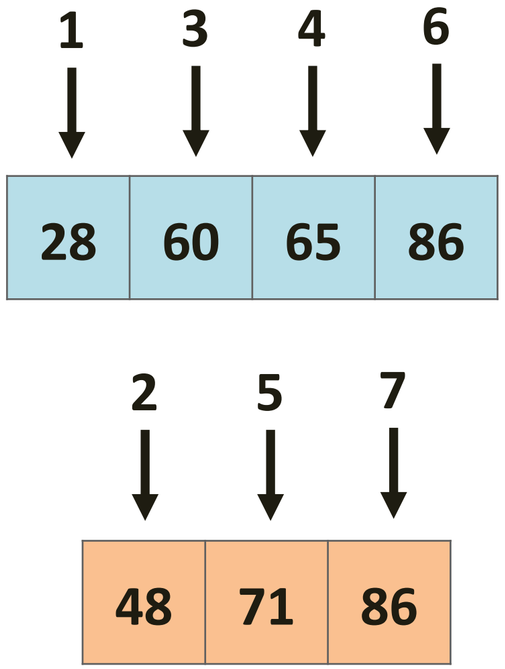


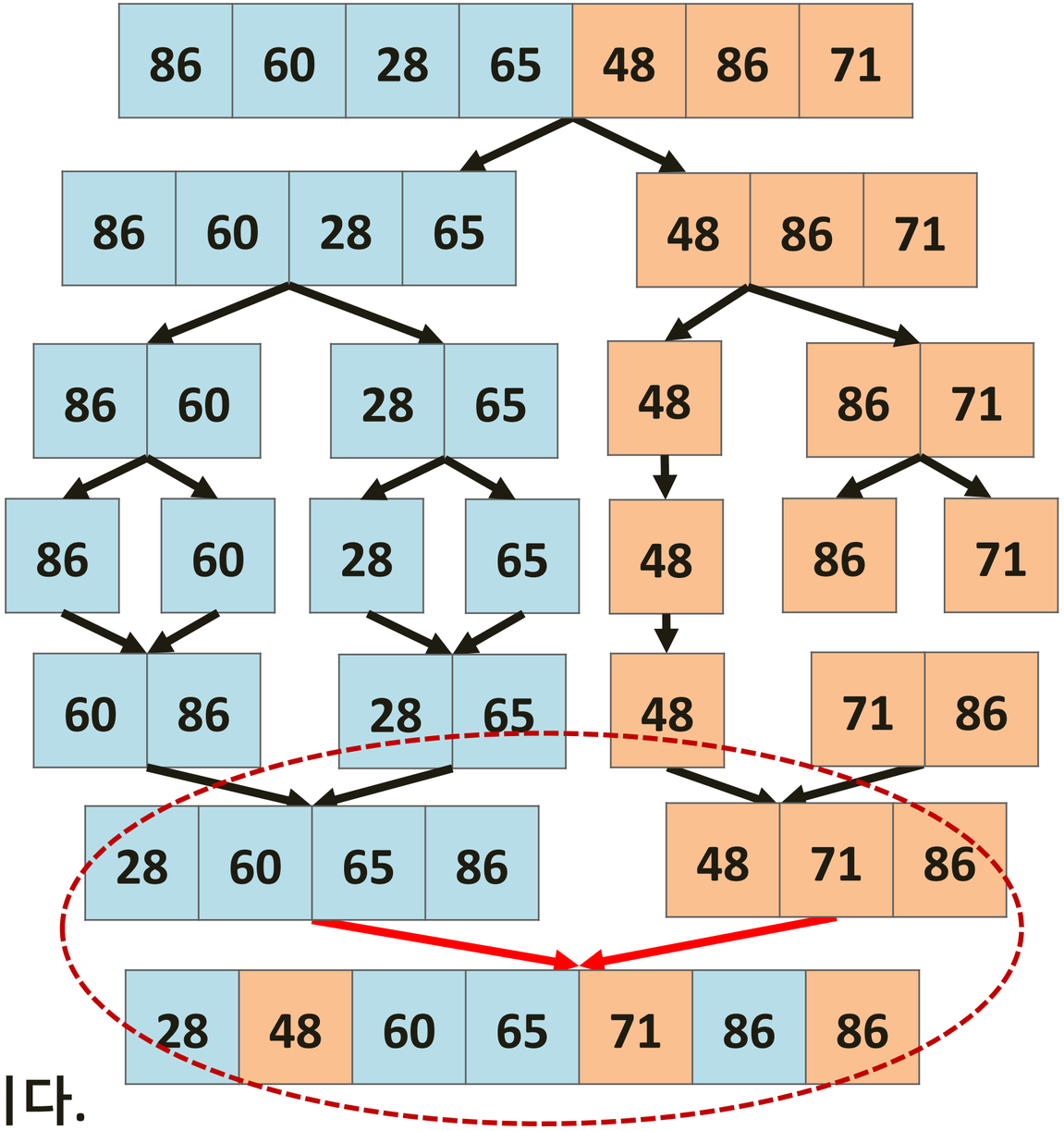
다시 주황색 배열을 보자.

각각 [48], [71,86]이다. 포인터 연산을 통해 작은 원소부터 합쳐준다.

고로 [48,71,86]으로 합쳐진다(정복된다).

분할 정복





이제 전체 배열을 합칠 차례이다.

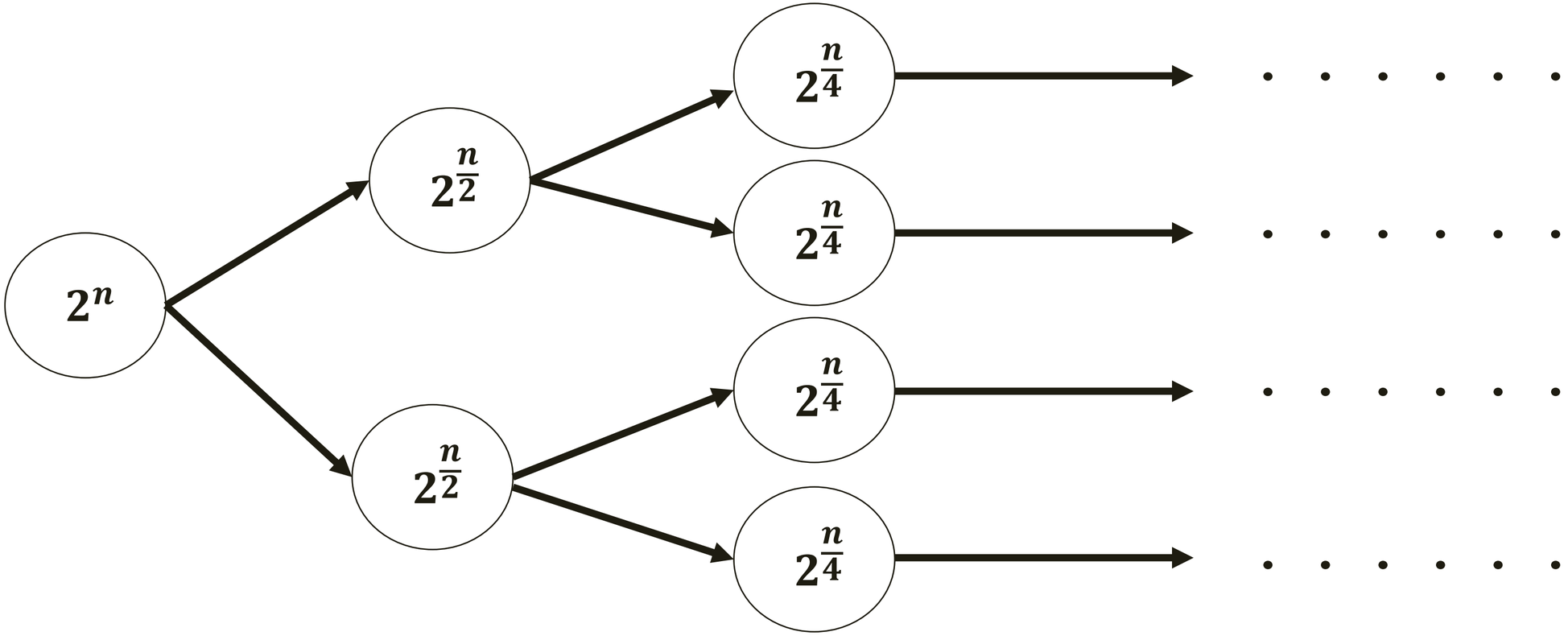
각각 [28,60,65,86] [48,71,86]이므로 [28,48,60,65,71,86,86]으로 합쳐진다.

분할 정복을 이용한 거듭 제곱

$2^{1000000000}$을 어떻게 구할 수 있을까?

(1) 2를 1000000000번 곱한다. -> 매우 느리다. O(N)

(2) 분할정복을 이용해서 빠르게 거듭 제곱을 한다. O(log N)
-> 이게 어떻게 가능한 걸까?



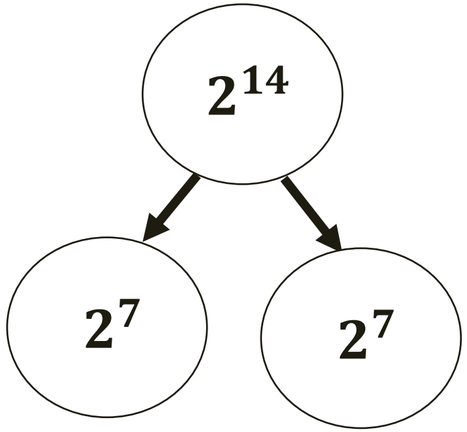
분할 정복을 이용한 거듭 제곱

2^{14}

조금 더 현실적인 예제를 가져와 봤다.

2^{14}을 분할정복으로 구해보자!

분할 정복을 이용한 거듭 제곱



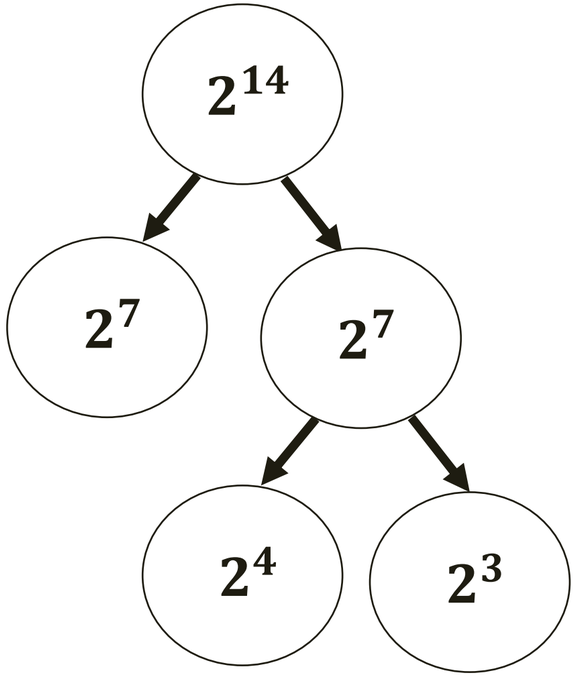
먼저 분할 정복을 위해 2^{14}을 반으로 나누자.

깔끔하게 2^{7}두개로 나누어 진다.

중요한 점은 재분할 후에 모든 원소에 대해서 분할을 실시 하지 않는다는 점이다.

즉, 2^{7} 가 2개가 있을 때 한쪽만 기저 조건까지 보낸 후에 그 값을 제곱하면 된다.
두 분할에 대해서 전부 보내도 되지만, 시간적으로 손해가 된다. (이미 계산된 값을 재계산)

분할 정복을 이용한 거듭 제곱

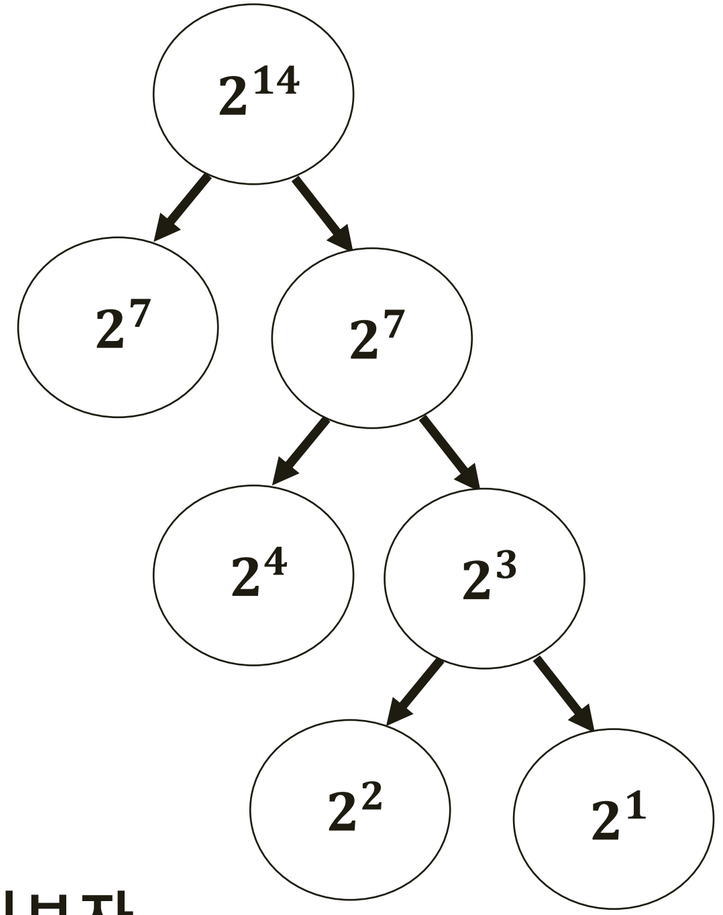


오른쪽 2^7원소를 고르겠다. 반으로 나눠보자.

2^4과 2^3로 나누어 진다. 깔끔하게 나누어 지지 않는다.

하지만 2^4 == 2^3 x 2에 의해 2^3을 골라도 문제가 없다. 일단 작은 것을 골라보자.

분할 정복을 이용한 거듭 제곱

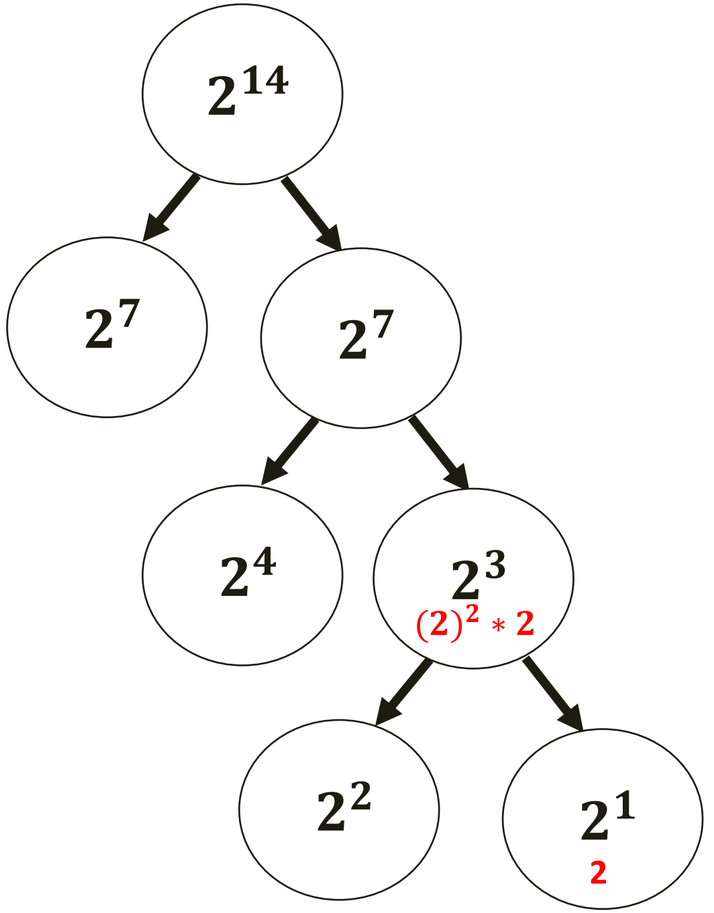


오른쪽 2^3원소를 고르겠다. 반으로 나눠보자.

2^2과 2^1로 나누어 진다. 깔끔하게 나누어 지지 않는다.

하지만 2^2 == 2^1 x 2에 의해 2^1을 골라도 문제가 없다. 이 때, 2^1은 기저조건이다. 이제 정복하자!

분할 정복을 이용한 거듭 제곱

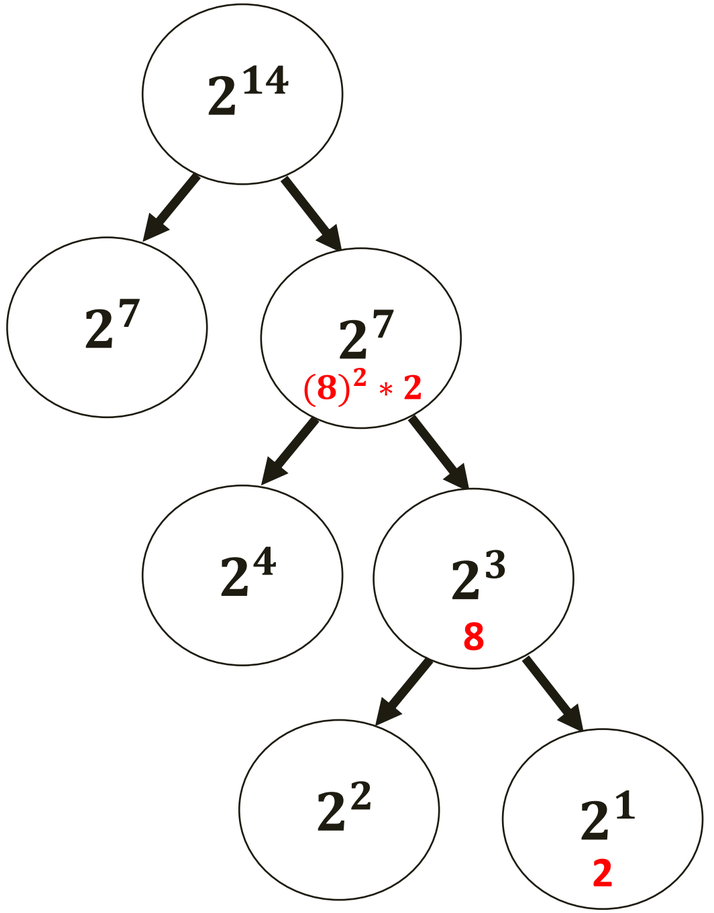


2^{1}은 자명하게 2이다.

이 값은 2^{3}으로 부터 분리 되어 있는 상태이다. 또한 왼쪽 원소가 오른쪽 원소가 같지 않다.

고로 $(2)^{2}$을 해주고 각 원소가 다르기 때문에 2를 곱한 8이 2^{3}의 결과이다.

분할 정복을 이용한 거듭 제곱

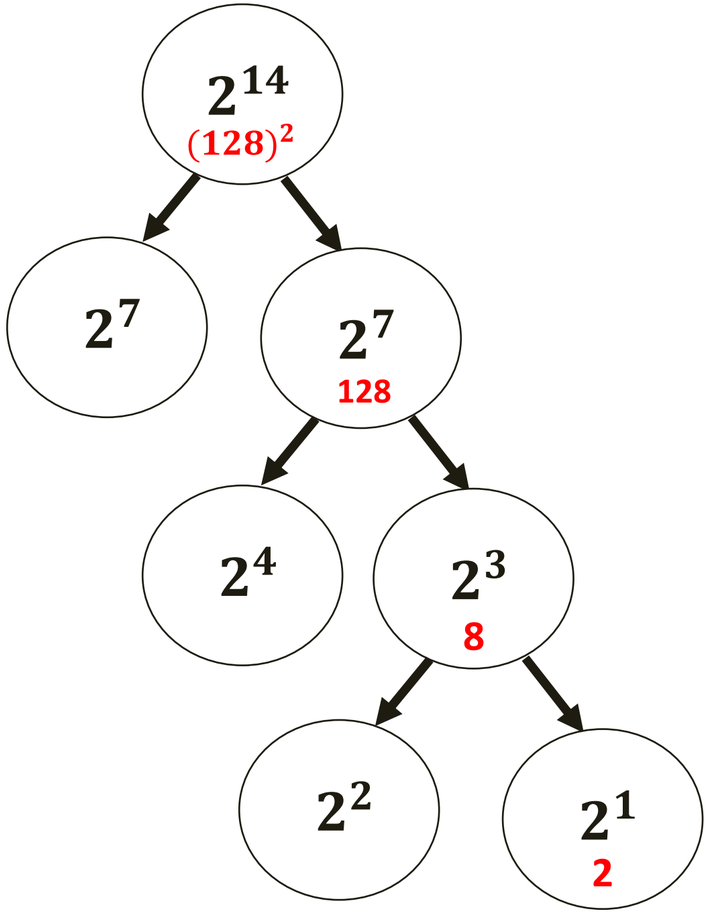


2^3은 8이다.

이 값은 2^7으로 부터 분리 되어 있는 상태이다. 또한 왼쪽 원소가 오른쪽 원소가 같지 않다.

고로 $(8)^2$을 해주고 각 원소가 다르기 때문에 2를 곱한 128이 (2^7)의 결과이다.

분할 정복을 이용한 거듭 제곱

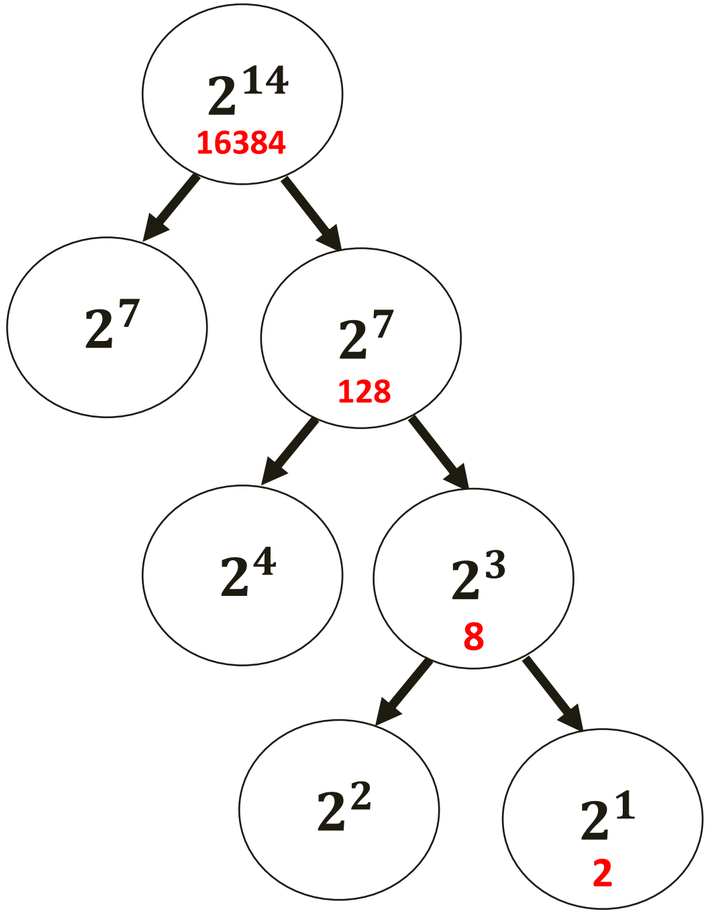


2^7은 128이다.

이 값은 2^{14}으로 부터 분리 되어 있는 상태이다. 왼쪽 원소가 오른쪽 원소가 같다!

고로 $(128)^2$을 해준 값인 16384가 (2^{14})의 결과이다.

분할 정복을 이용한 거듭 제곱



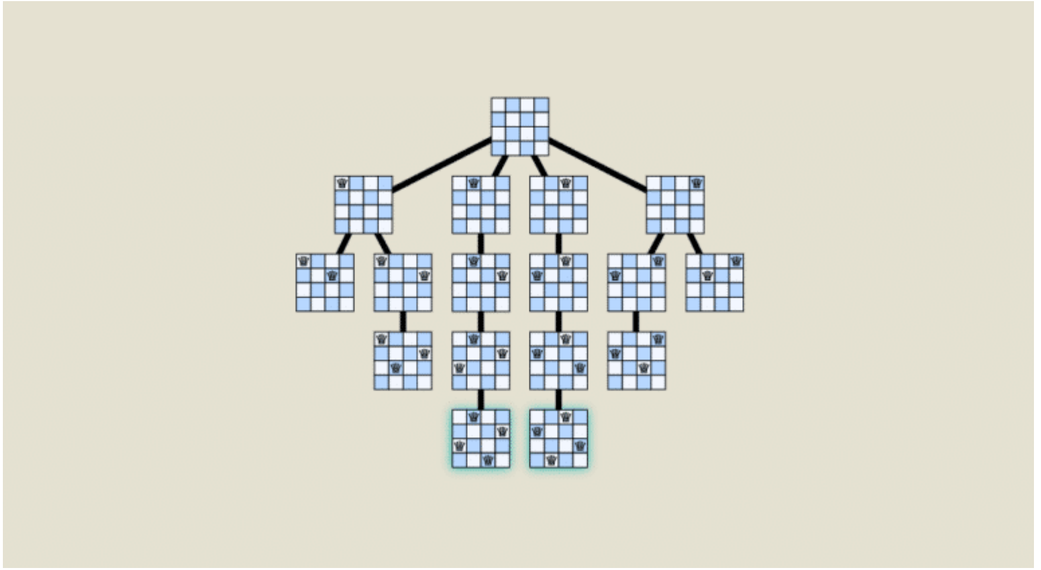
주목할만한 점은 아래와 같다.

1 – 기저의 값 (x^y에서 기저는 x) 을 통해 다른 거듭 제곱의 값을 유추할 수 있다.

2 – 중복된 값의 계산을 막기 위해 분할 후에 지수 차수가 낮은 놈을 보낸다. 이것이 가능한 이유는 분할 된 두 값의 지수가 서로 같거나 1만큼 차이 나기 때문이다. 고로 정복 후 추후에 곱해주면 아무 문제가 없다.
(사실상 지수가 홀수여야 지수 분할 값이 다르기 때문에 홀수일때 추가조건을 따져주면 된다.)

백트래킹 이란?

백트래킹(Backtracking)은 해를 찾기 위해 가능한 모든 경우를 탐색하는 알고리즘 기법이다. 가장 큰 특징은 현재 상태에서 유망하지 않은 경로를 조기에 가지치기(Pruning)하여 불필요한 탐색을 줄이는 점이다. 일반적으로 재귀적으로 동작하며, 해결해야 할 문제의 모든 경우를 고려하되, 조건을 만족하지 않는 경우 더 깊이 탐색하지 않고 돌아간다.
대표적인 예시로 N-Queen 문제, 순열(Permutation) 생성, 부분 집합(Subsets) 구하기, 수열 문제(DFS 기반 탐색) 등이 있으며, 가능한 경우를 효율적으로 탐색하는 데 유리하다.



백트래킹으로 풀 수 있는
대표적인 문제 - N-QUEEN

백트래킹

자연수 N과 M이 주어졌을 때, 아래 조건을 만족하는 길이가 M인 수열을 모두 구하는 프로그램을 작성하시오.

- 1부터 N까지 자연수 중에서 중복 없이 M개를 고른 수열

입력

첫째 줄에 자연수 N과 M이 주어진다. (1 ≤ M ≤ N ≤ 8)

출력

한 줄에 하나씩 문제의 조건을 만족하는 수열을 출력한다. 중복되는 수열을 여러 번 출력하면 안되며, 각 수열은 공백으로 구분해서 출력해야 한다.

수열은 사전 순으로 증가하는 순서로 출력해야 한다.

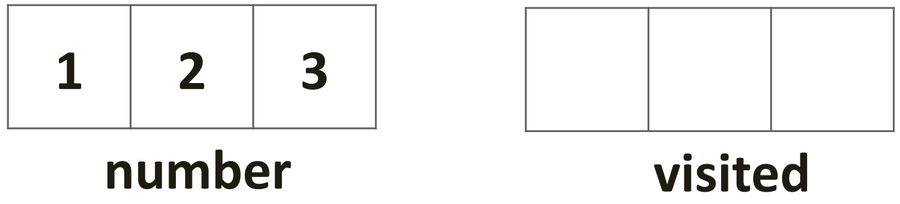
특히 백트래킹은 순열 생성 문제에서 진가를 발휘하게 된다.

간단하게 백트래킹으로 순열을 생성하는 법에 대해서 설명을 하겠다.

3개의 자연수 중에서 중복 없이 2개를 고른 수열을 오름차순으로 구하는 방법을 알아보자.

백트래킹

현재 depth : 0



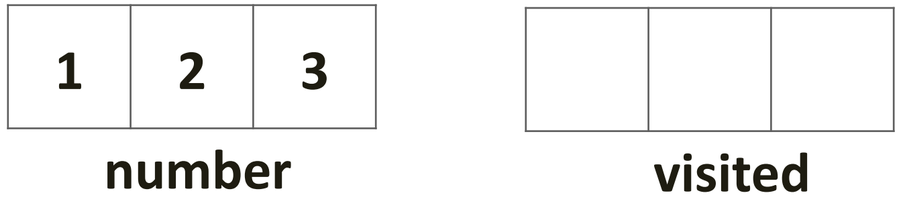
먼저, 깊이라는 개념부터 정립해보자.

백트래킹은 기본적으로 재귀 함수(스택)을 쓰기 때문에 깊이라는 개념이 존재한다.

각 깊이는 하나의 상태 공간을 가진다고 생각하면 된다.

백트래킹

현재 depth : 0

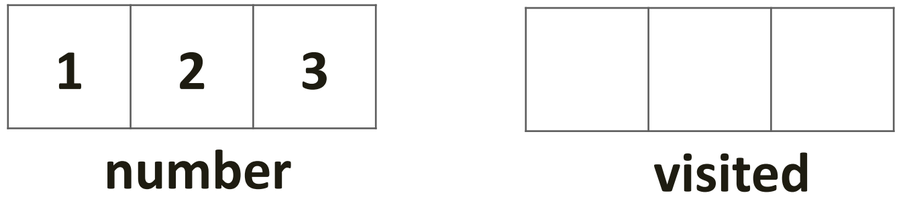


그 상태 공간안에서 우리는 아무 행위나 할 수 있으며, 그 상태 공간에서 벗어날 수도 있고(return) 혹은 다음 상태 공간으로 넘어갈 수도 있다.(현재 상태공간을 쌓아두고 다음 깊이로 가기)

백트래킹의 기본 전제 조건에서 미루어 볼 때, 각 깊이의 상태공간은 유일하다. 즉, 다른 깊이에서 현재 깊이를 침해하는 어떤 행위를 하지 않는 이상은 영향을 주지 못한다는 것이다.

백트래킹

현재 depth : 0

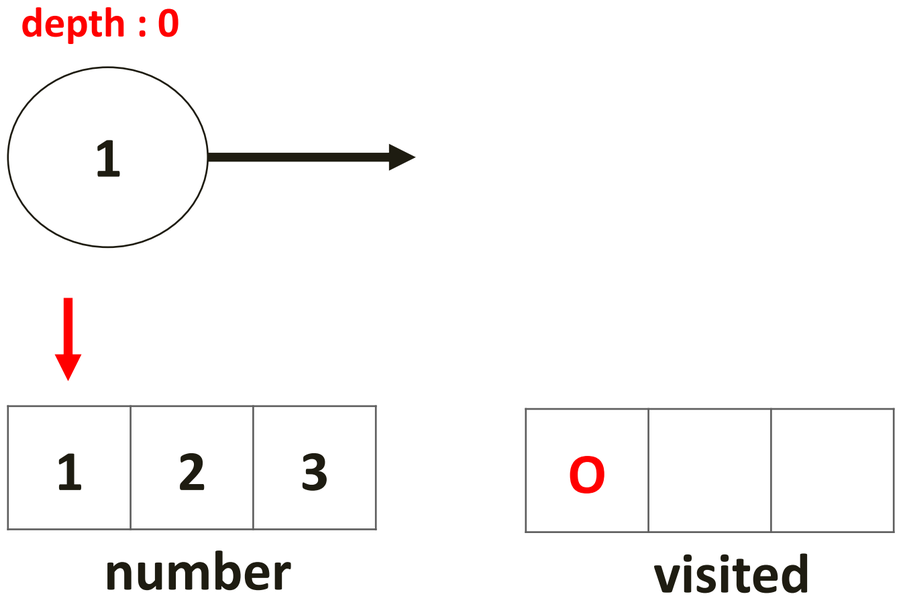


이러한 깊이는 우리가 고른 수의 개수로 환원 될 수 있다. (깊이 == 이미 고른 수의 개수)
이번 예제 에서는 3개의 수 중에서 2개를 고르는 경우를 살펴본다고 하였다.

결국 기저 조건은 깊이가 2일 때일 것이다. 기저 조건에 도달했으면 조건을 확인하고 적절한 행동을 취한 뒤에 return을 하면 된다. 우리가 할 수 있는 것은 각 깊이에서 수를 고르는 것이다.

백트래킹

현재 depth : 0



현재 깊이는 0이다.

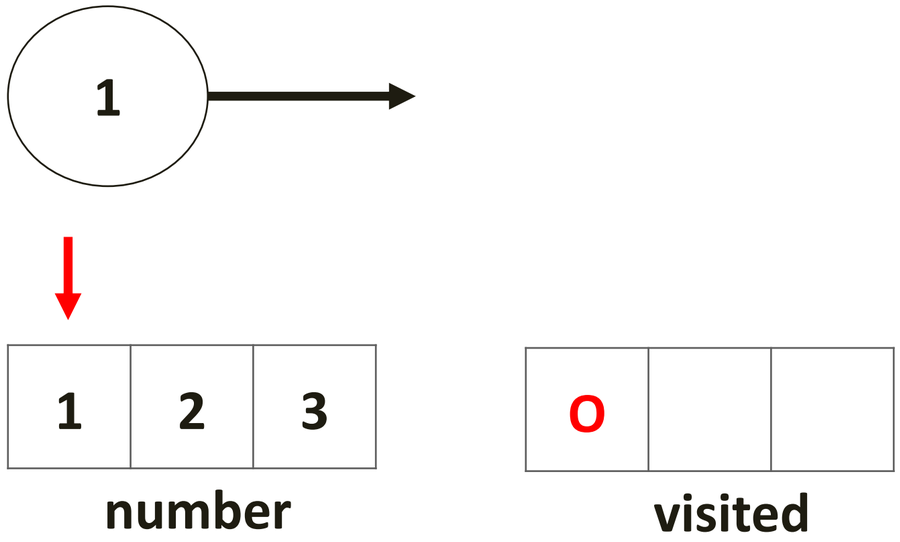
사전순으로 골라야 하니 1,2,3 순서대로 원소를 선택하겠다.

먼저 1이 방문 처리가 안되어 있다. 고로 1을 방문 처리하고 1을 고른 뒤에 다음 깊이로 넘어간다.

백트래킹

현재 depth : 1

depth : 0



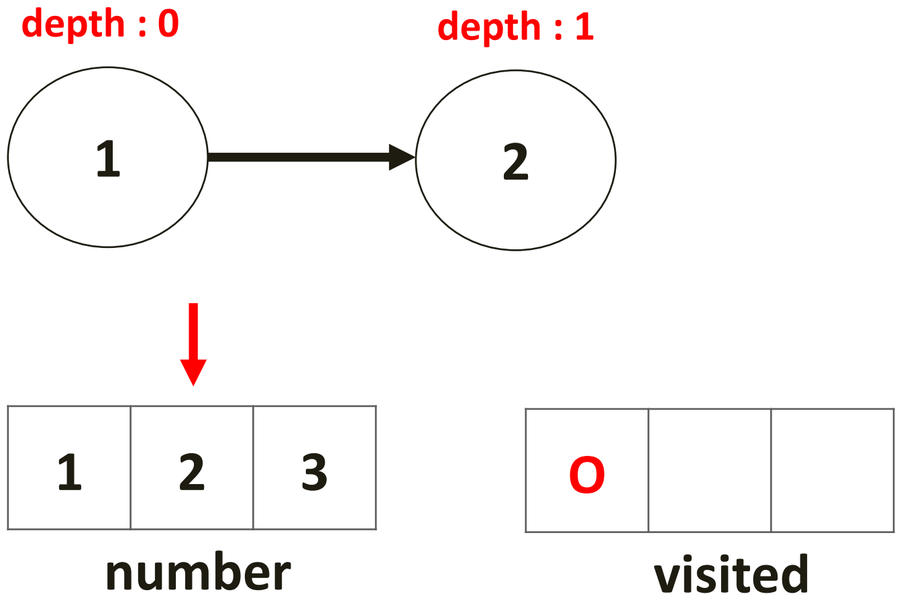
현재 깊이는 1이다. 즉, 1개의 숫자를 골랐다는 의미 이다.

사전순으로 골라야 하니 1,2,3 순서대로 원소를 선택하겠다.

1을 고르려고 하니 이미 골라져 있다.(방문처리 되어 있음) 고로 다른 수를 골라야 한다.

백트래킹

현재 depth : 1



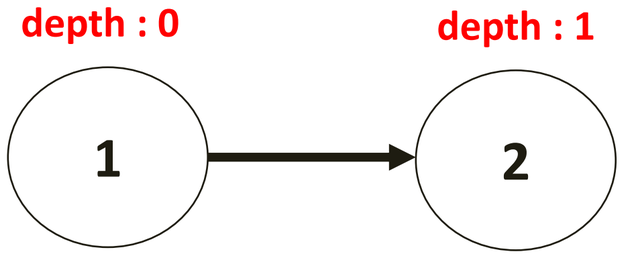
2는 방문처리가 안되어 있는 원소이다.

고로 2를 방문처리 하고 2를 고른 뒤에 다음 깊이로 넘어가자.

(depth는 재귀함수로, 수를 고르는 작업은 일반적인 반복문으로 구현되게 된다.)

백트래킹

현재 depth : 2



1	2	3

number

O	O	

visited

현재 깊이는 2이다. 우리의 목표는 3개의 수 중에서 중복 없이 2개를 고르는 것이였다.

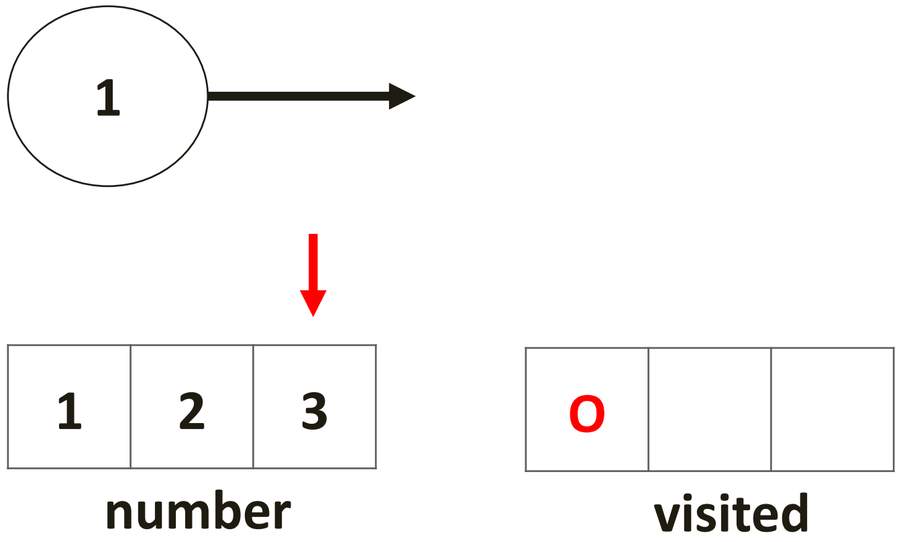
깊이가 2인 것은 원소를 2개 골랐다는 걸 의미한다. 고로 [1 2]를 출력한다.

다른 순열을 찾기 위해서 return을 한다.

백트래킹

현재 depth : 1

depth : 0



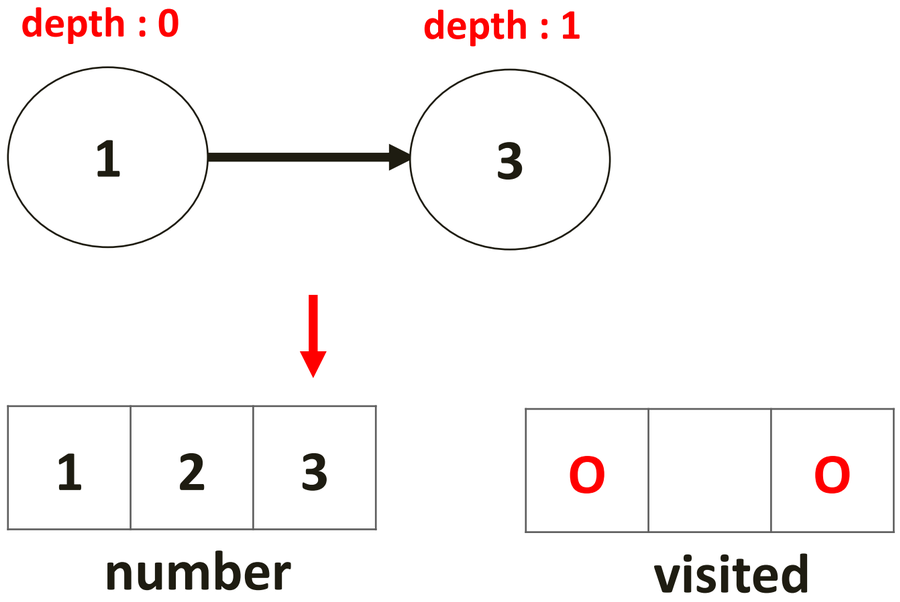
깊이 1로 돌아 왔다. 우리는 깊이 1이라는 상태 공간에서 2를 이미 사용한 상태이다.

다른 조합을 찾기 위해 2가 아닌 다른 숫자도 집어 넣어 봐야 하므로 2의 방문처리를 해제 한다.

수를 고르는 작업은 일반적인 반복문으로 구현되기 때문에 자동으로 2 다음인 3을 가리킨다.

백트래킹

현재 depth : 1



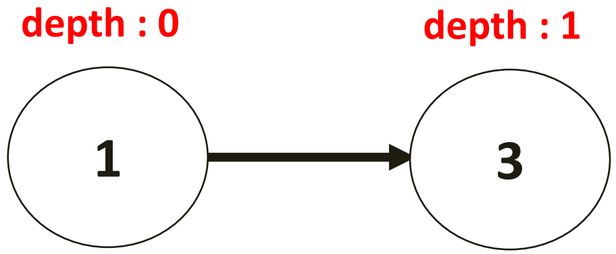
현재 반복문이 가리키는 3은 방문처리가 안되어 있는 원소이다.

고로 3를 방문처리 하고 3를 고른 뒤에 다음 깊이로 넘어가자.

(재귀에 쌓이는 상태 공간 정보는 재귀에 들어갈 당시의 모든 정보를 그대로 기억하고 있다.)

백트래킹

현재 depth : 2



1	2	3

number

O		O

visited

현재 깊이는 2이다. 우리의 목표는 3개의 수 중에서 중복 없이 2개를 고르는 것이였다.

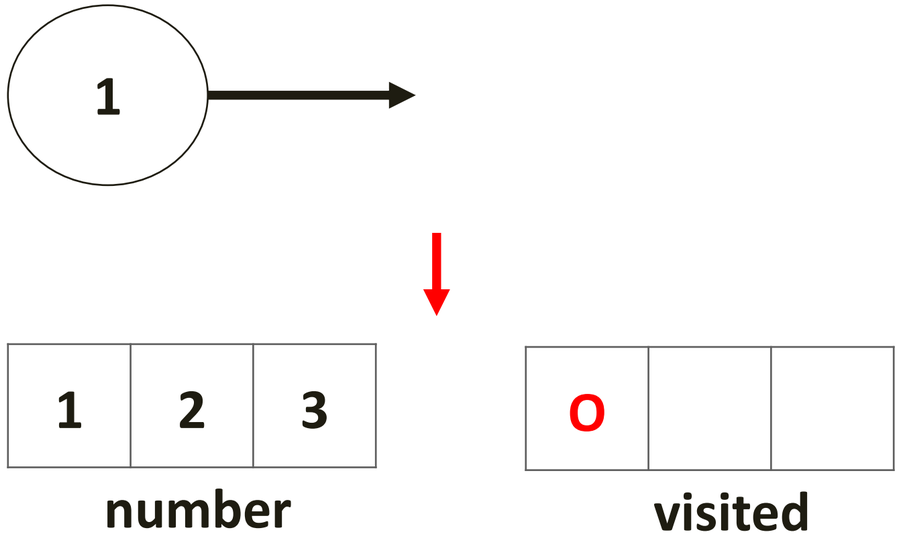
깊이가 2인 것은 원소를 2개 골랐다는 걸 의미한다. 고로 [1 3]를 출력한다.

다른 순열을 찾기 위해서 return을 한다.

백트래킹

현재 depth : 1

depth : 0



깊이 1로 돌아왔으니 사용한 값인 3의 방문 처리를 해제 해준다.

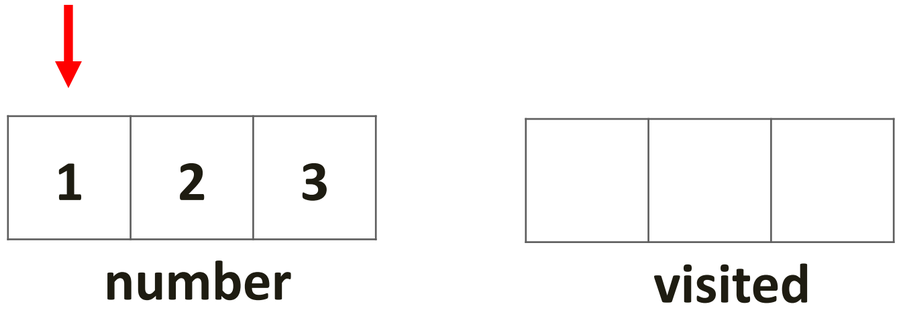
3 이후엔 원소가 없으므로 자동적으로 루프가 끝나게 된다.

이는 더 이상 고를 수 있는 수가 없다는 의미 이므로 이전 깊이로 return 한다.

백트래킹

현재 depth : 0

depth : 0



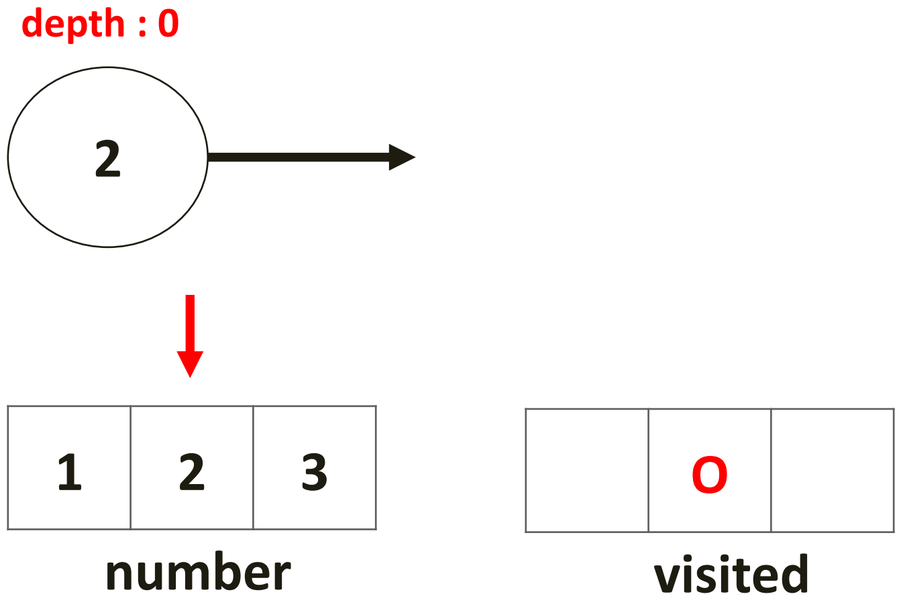
깊이 0으로 돌아왔다. 고로 사용한 값인 1의 방문 처리를 해제 해준다.

1을 고른 후에 반복문의 포인터는 그대로 유지 되어있다. (재귀의 상태공간 유지에 의해)

고로 반복문의 실행을 그대로 이어가자.

백트래킹

현재 depth : 0



현재 반복문은 2를 가리킨다. 2는 방문처리가 되어 있지 않다.

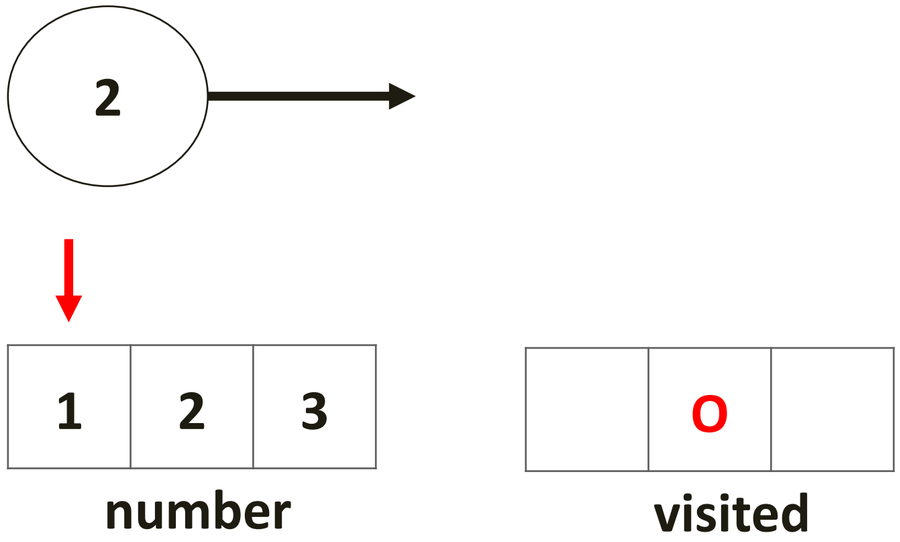
고로 2를 방문처리를 한다.

이후, 다음 깊이로 넘어간다.

백트래킹

현재 depth : 1

depth : 0



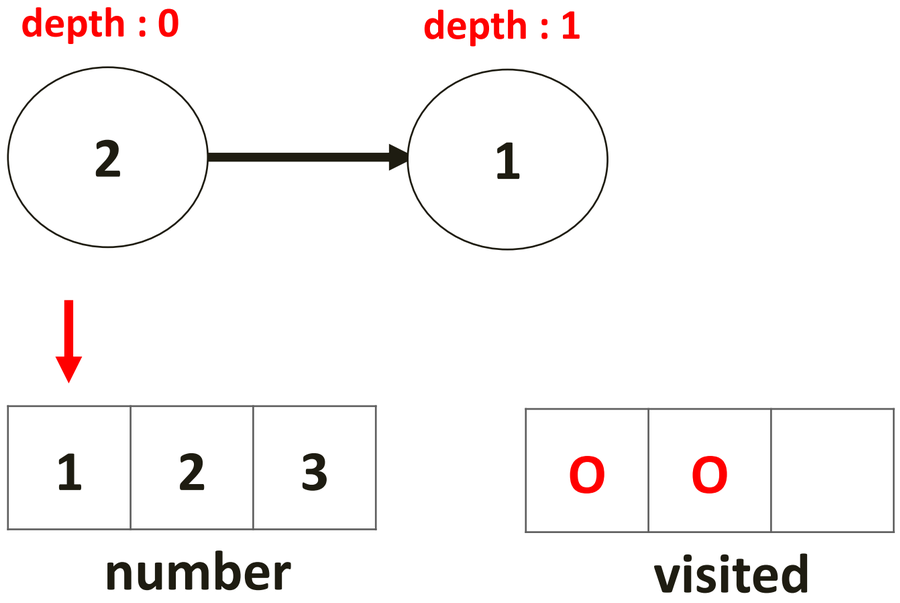
현재 깊이는 1이다. 새로운 깊이로 넘어왔기 때문에 상태 공간은 새로 정의 된다.

고로 반복문의 포인터도 1부터 다시 시작 되게 된다.

반복문의 실행 흐름을 따라가보자.

백트래킹

현재 depth : 1



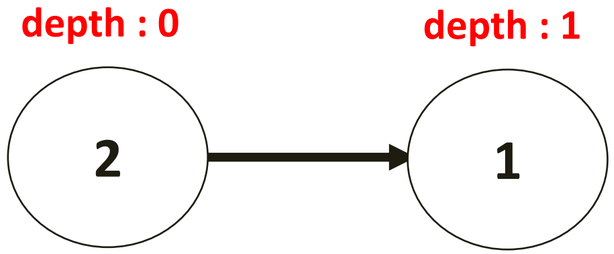
현재 반복문은 1을 가리킨다. 1은 방문처리가 되어 있지 않다.

고로 1을 방문처리를 한다.

이후, 다음 깊이로 넘어간다.

백트래킹

현재 depth : 2



1	2	3

number

O	O	

visited

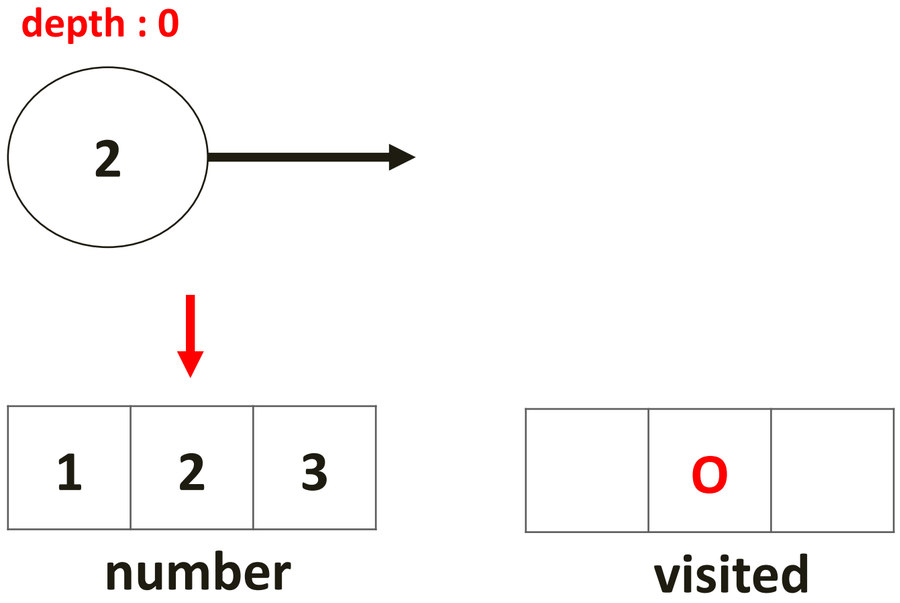
현재 깊이는 2이다. 우리의 목표는 3개의 수 중에서 중복 없이 2개를 고르는 것이였다.

깊이가 2인 것은 원소를 2개 골랐다는 걸 의미한다. 고로 [2 1]를 출력한다.

다른 순열을 찾기 위해서 return을 한다.

백트래킹

현재 depth : 1



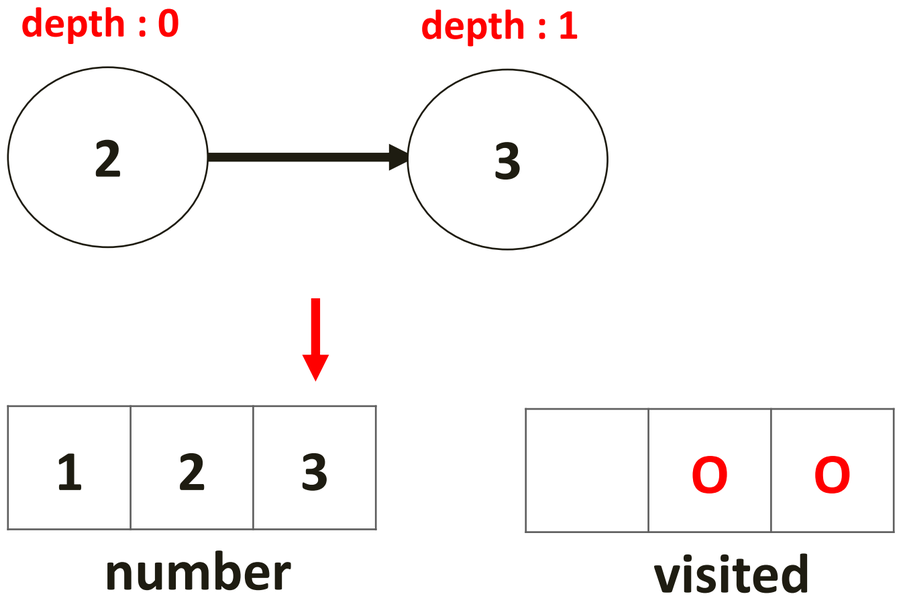
깊이 1로 다시 복귀 하였다. 이미 사용한 값인 1을 방문 처리 해제 한다.

반복문에 의해 포인터가 한 칸 늘려진다.

이때, 반복문이 가리키는 원소는 2이다. 하지만 이미 방문 처리 상태이므로 무시한다.

백트래킹

현재 depth : 1



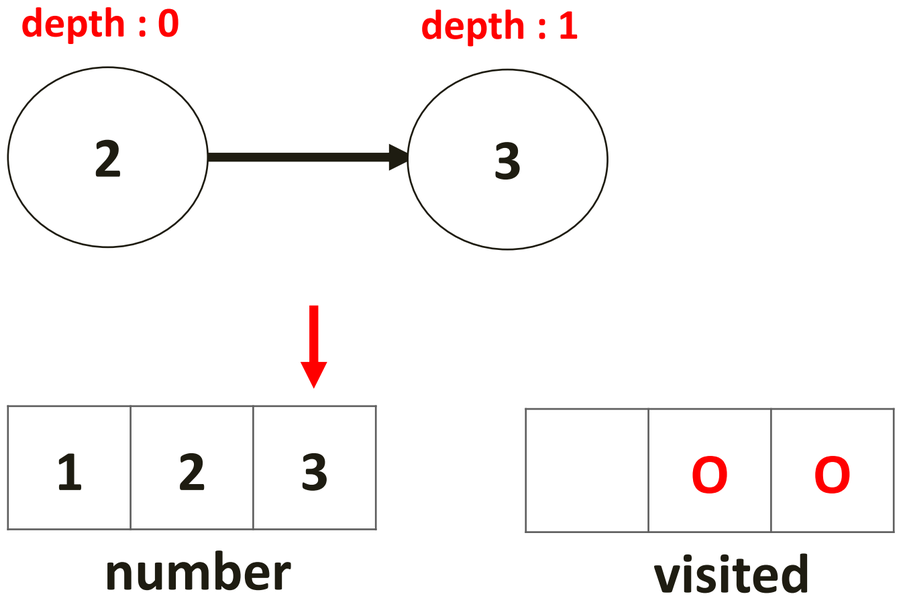
현재 반복문은 3을 가리킨다. 3은 방문 처리가 되어 있지 않다.

고로 3을 방문 처리한다.

이후, 다음 깊이로 넘어간다.

백트래킹

현재 depth : 2



현재 깊이는 2이다. 우리의 목표는 3개의 수 중에서 중복 없이 2개를 고르는 것이였다.

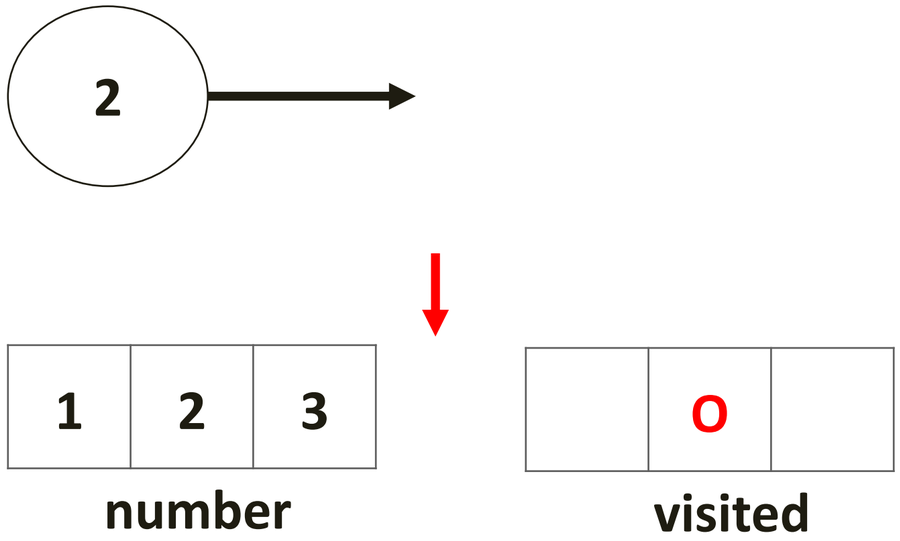
깊이가 2인 것은 원소를 2개 골랐다는 걸 의미한다. 고로 [2 3]를 출력한다.

다른 순열을 찾기 위해서 return을 한다.

백트래킹

현재 depth : 1

depth : 0



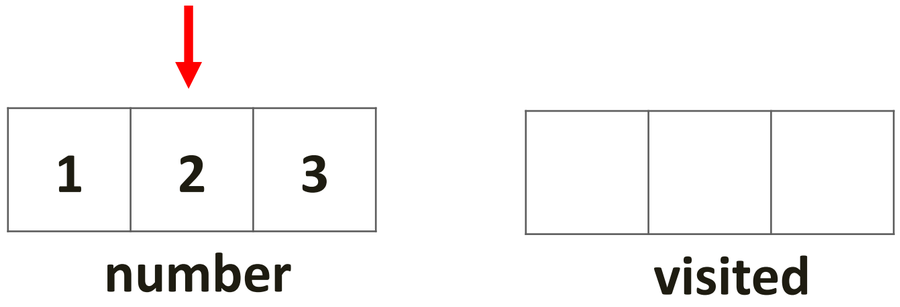
깊이 1로 복귀 하였기 때문에 이미 사용한 원소인 3의 방문 처리를 해제한다.

3 이후엔 원소가 없으므로 자동적으로 루프가 끝나게 된다.

이는 더 이상 고를 수 있는 수가 없다는 의미 이므로 이전 깊이로 return 한다.

백트래킹

현재 depth : 0



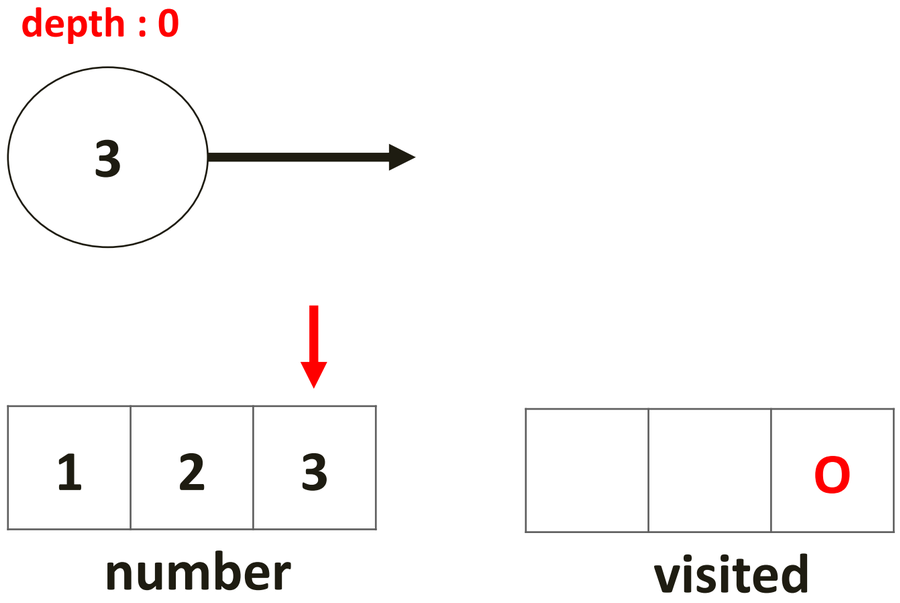
깊이 0으로 돌아왔다. 고로 사용한 값인 2의 방문 처리를 해제 해준다.

2을 고른 후에 반복문의 포인터는 그대로 유지 되어있다. (재귀의 상태공간 유지에 의해)

고로 반복문의 실행을 그대로 이어가자.

백트래킹

현재 depth : 0



현재 반복문은 3를 가리킨다. 3는 방문처리가 되어 있지 않다.

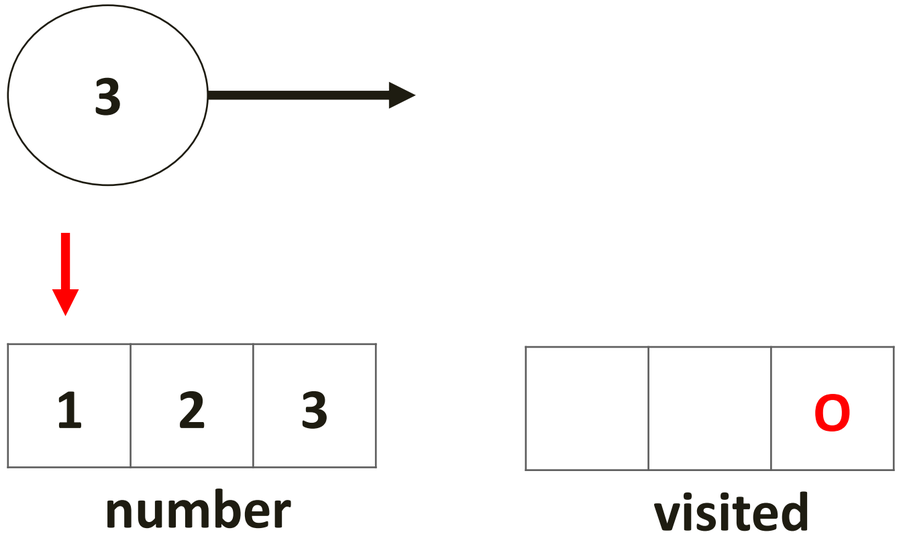
고로 3를 방문처리를 한다.

이후, 다음 깊이로 넘어간다.

백트래킹

현재 depth : 1

depth : 0



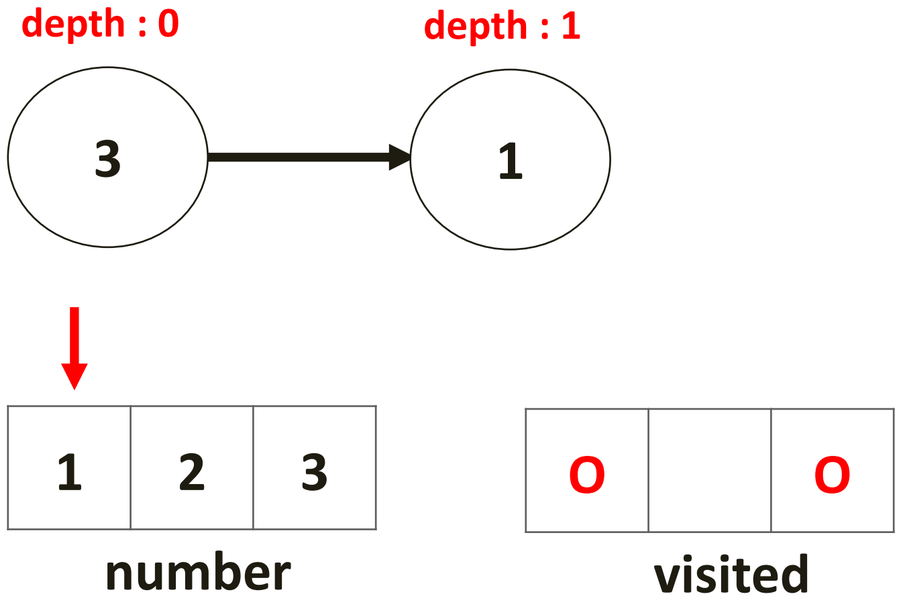
현재 깊이는 1이다. 새로운 깊이로 넘어왔기 때문에 상태 공간은 새로 정의 된다.

고로 반복문의 포인터도 1부터 다시 시작 되게 된다.

반복문의 실행 흐름을 따라가보자.

백트래킹

현재 depth : 1



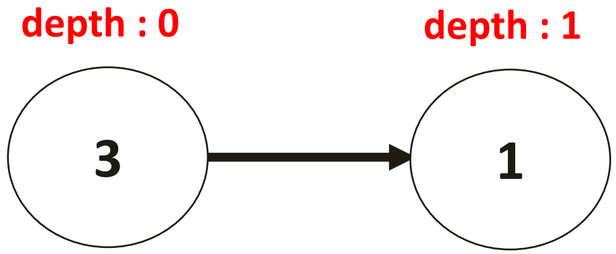
현재 반복문은 1을 가리킨다. 1은 방문처리가 되어 있지 않다.

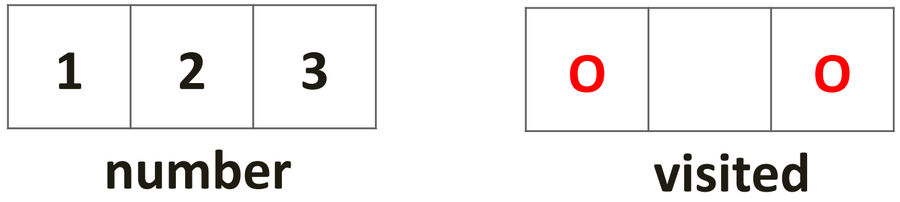
고로 1을 방문처리를 한다.

이후, 다음 깊이로 넘어간다.

백트래킹

현재 depth : 2





현재 깊이는 2이다. 우리의 목표는 3개의 수 중에서 중복 없이 2개를 고르는 것이였다.

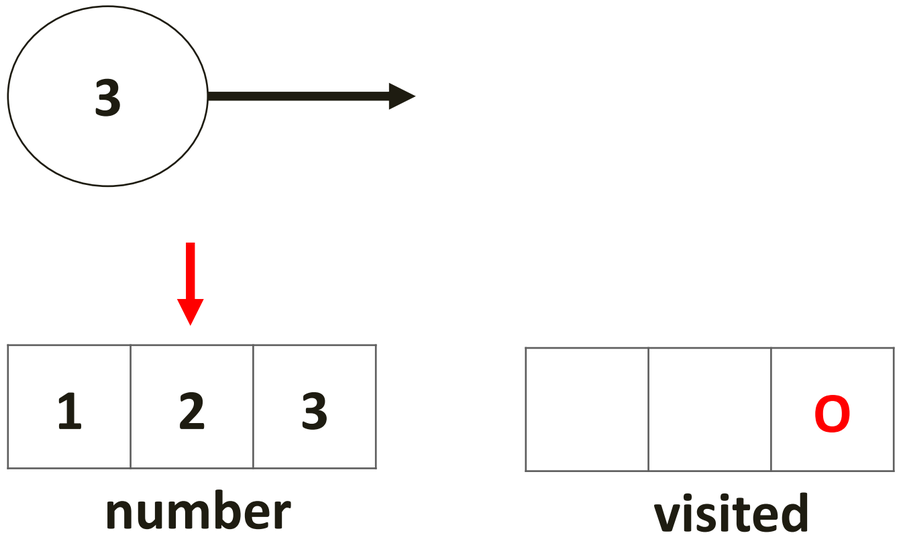
깊이가 2인 것은 원소를 2개 골랐다는 걸 의미한다. 고로 [3 1]를 출력한다.

다른 순열을 찾기 위해서 return을 한다.

백트래킹

현재 depth : 1

depth : 0



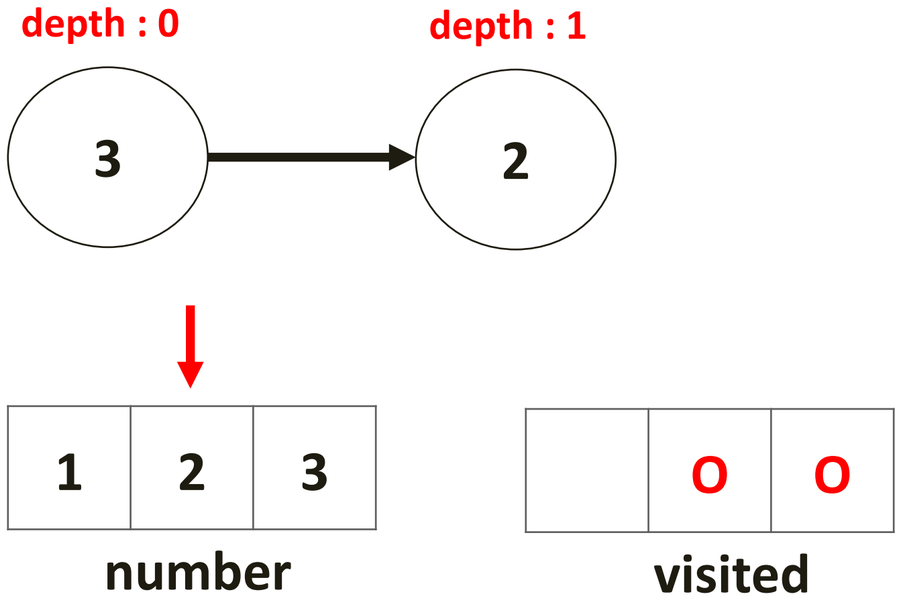
깊이 1로 다시 복귀 하였다. 이미 사용한 값인 1을 방문 처리 해제 한다.

반복문에 의해 포인터가 한 칸 늘려진다.

반복문의 흐름에 따라 원소가 가능한지 확인해보자.

백트래킹

현재 depth : 1



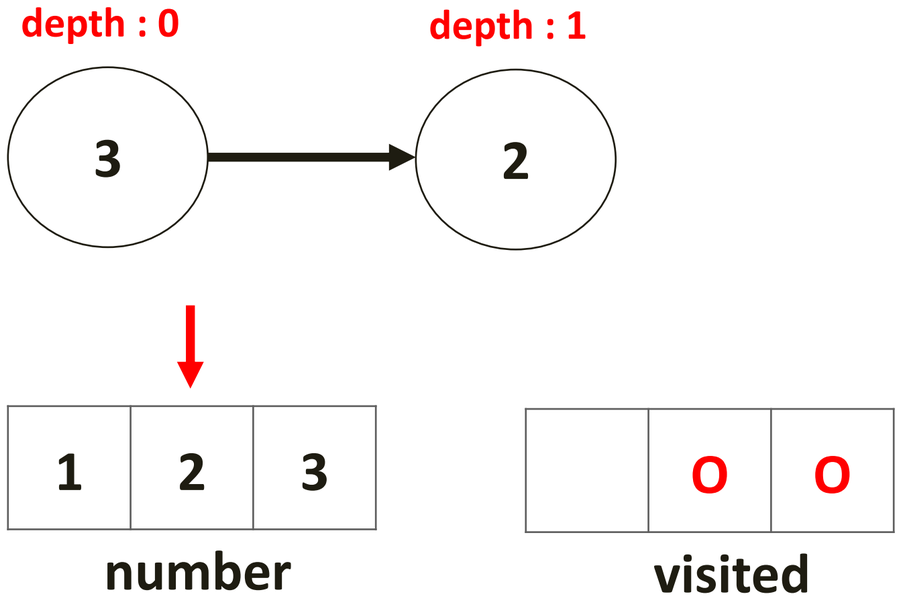
현재 반복문은 2을 가리킨다. 2는 방문처리가 되어 있지 않다.

고로 2를 방문처리를 한다.

이후, 다음 깊이로 넘어간다.

백트래킹

현재 depth : 2



현재 깊이는 2이다. 우리의 목표는 3개의 수 중에서 중복 없이 2개를 고르는 것이였다.

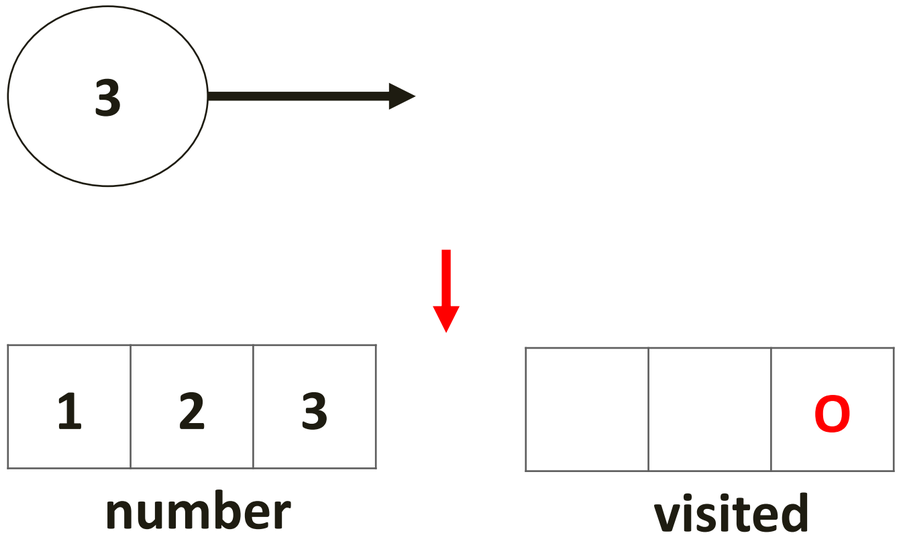
깊이가 2인 것은 원소를 2개 골랐다는 걸 의미한다. 고로 [3 2]를 출력한다.

다른 순열을 찾기 위해서 return을 한다.

백트래킹

현재 depth : 1

depth : 0



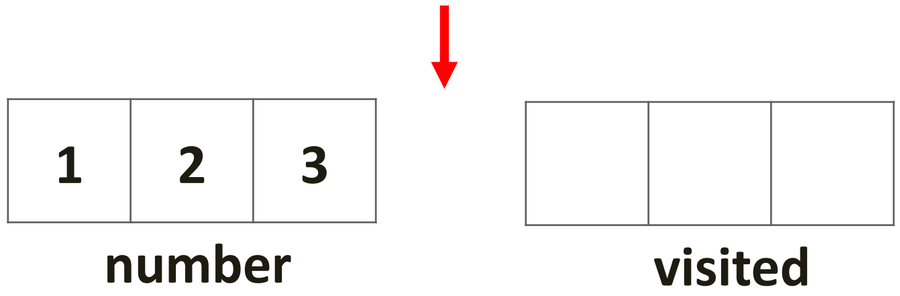
깊이 1로 복귀 하였기 때문에 이미 사용한 원소인 2의 방문 처리를 해제한다.

2 이후에 있는 원소는 3이지만, 이미 방문 처리가 되었으므로 무시한다. 이후, 반복문이 끝난다.

이는 더 이상 고를 수 있는 수가 없다는 의미 이므로 이전 깊이로 return 한다.

백트래킹

현재 depth : 0



깊이 0으로 복귀 하였기 때문에 이미 사용한 원소인 3의 방문 처리를 해제한다.

반복문이 끝났다. 이는 더 이상 고를 수 있는 수가 없다는 의미 이므로 이전 깊이로 return 한다.

지금까지 총 6개의 순열을 만드는 방법을 보여줬다.

백트래킹

일반적으로 백트래킹은 전수 조사이기 때문에 큰 입력에 대해서는 적합하지 않은 알고리즘이다.
그러나 입력 범위가 비교적 작고(보통 n≤15), 문제가 전수 조사를 요구한다면 충분히 시도해볼 만한 방법이다.
대부분의 백트래킹 문제는 어떤 요소를 선택할지 말지를 결정하는 형태로 이루어져 있다.

하지만 무작정 전수조사를 하면 시간 복잡도가 너무 커지는 경향이 있는데, 이를 방지 하기 위해서 가지치기(Pruning) 라는 기법을 사용한다.

아쉽게도, 가지치기에는 특정한 공식이나 방법이 존재하지 않는다.
아래와 같이 조건을 따져 프로그래머가 직접 최적화할 수 있는 방법이 존재한다.

(1) 조기 종료 – 탐색 도중 정답이 될 가능성이 아예 없는 경우 그 선택지를 즉시 중단하는 방식.
(부분 수열의 합이 특정 값을 초과하는 경우 더 이상 탐색할 필요가 없다)

(2) 해시나 비트마스크, 혹은 배열로 방문 체크 활용 – 중복된 탐색을 방지하기 위해 제일 많이 쓰이는 방법.
(같은 원소를 여러 번 선택하는 경우를 피할 때 매우 유용)

(3) 최적 조건을 활용한 탐색 축소 – 문제 조건에 따라 탐색 범위를 줄이는 방법.
(N-Queen 문제에서 같은 열이나 대각선에 위치하는 경우 탐색을 더 이상 진행하지 않는 방법)

(4) 특정 문제에서 불가능한 경우를 제거 – 특정 상태에서 답이 될 수 없는 경우를 미리 제거.
(최소 비용 경로를 찾는 문제에서 현재 비용이 이미 최솟값 보다 크면 진행할 필요가 없기 때문)