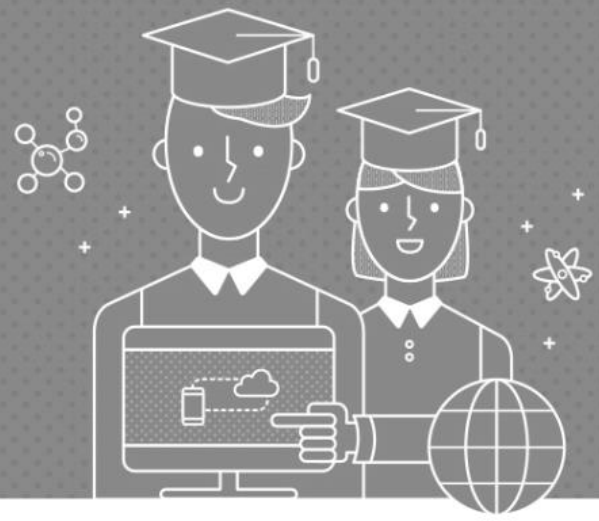
백트래킹



상태공간트리를 구축하여 문제를 해결

```
bool bactrack(선택 집합, 선택한 수, 모든 선택수)
{
     if (선택한 수 == 모든 선택수) // 더 이상 탐색할 노드가 없다.
     {
          찾는 솔루션인지 체크;
          return 결과;
     }

    현재 선택한 상태집합에 포함되지 않는 후보 선택들(노드) 생성

    모든 후보 선택들에 대해
    {
        선택 집합에 하나의 후보선택을 추가
        선택한 수 = 선택한 수 + 1
        결과 = backtrack 호출(선택집합, 선택한 수, 모든 선택수)

        if (결과 == 성공)
             return 성공; //성공한 경우 상위로 전달
     }
     return 실패;
}
```

상태공간트리를 구축하여 문제를 해결

- {1,2,3} 의 powerset을 구하는 백트래킹 알고리즘

```
backtrack(a[ ], k, input)
        c[MAXCANDIDATES]
        ncands

        IF k == input :  process_solution(a[ ], k)
        ELSE
                k++
                make_candidates(a[ ], k, input, c[ ], ncands)
                FOR i in 0 → ncands - 1
                        a[k] ← c[i]
                        backtrack(a, k, input)

main( )
        a[MAX]                  // powerset을 저장할 배열
        backtrack(a[ ], 0, 3) // 3개의 원소를 가지는 powerset
```

상태공간트리를 구축하여 문제를 해결

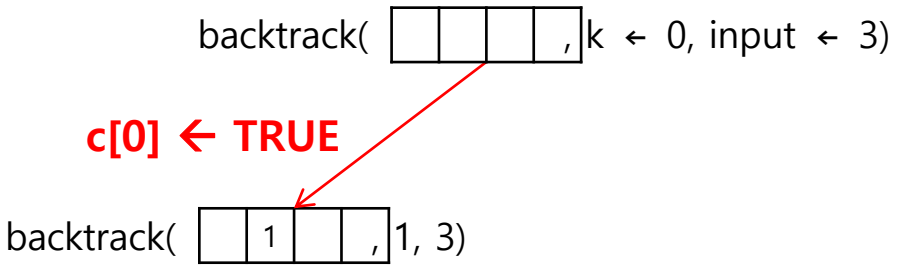
- powerset을 구하는 백트래킹 알고리즘(계속)

```
make_candidates(a[], k, n, c[], ncands)
        c[0] ← TRUE
        c[1] ← FALSE
        ncands ← 2

process_solution(a[], k)
        FOR  i in 1 → k
                IF a[i] == TRUE  :  print( i )
```

```
main( )
        a[MAX]
        backtrack(a[ ], 0, 3)
```

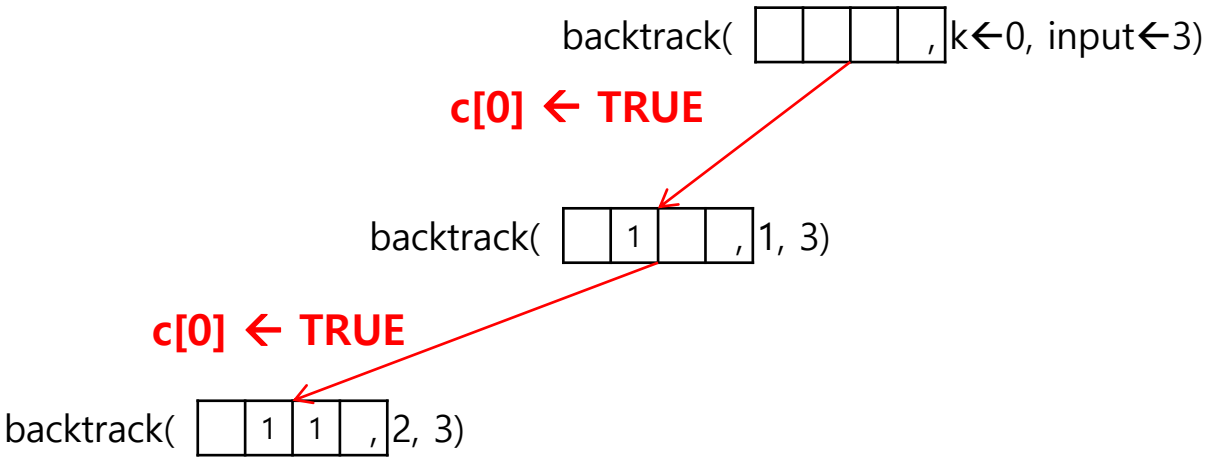
backtrack( , k ← 0, input ← 3)

```
backtrack(a[ ], k, input)   // a, 0, 3
        c[MAXCANDIDATES]
        ncands

        IF k == input :  process_solution(a[ ], k)
        ELSE
                k++
                make_candidates(a[ ], k, input, c[ ], ncands)
                FOR i in 0 → ncands - 1
                        a[k] ← c[i]
                        backtrack(a, k, input)
```




```
backtrack(a[ ], k, input)  // a, 0, 3
        c[MAXCANDIDATES]
        ncands

        IF k == input :  process_solution(a[ ], k)
        ELSE
                k++
                make_candidates(a[ ], k, input, c[ ], ncands)
                FOR i in 0 → ncands - 1
                        a[k] ← c[i]    // c[0] ← TRUE
                        backtrack(a, k, input)
```

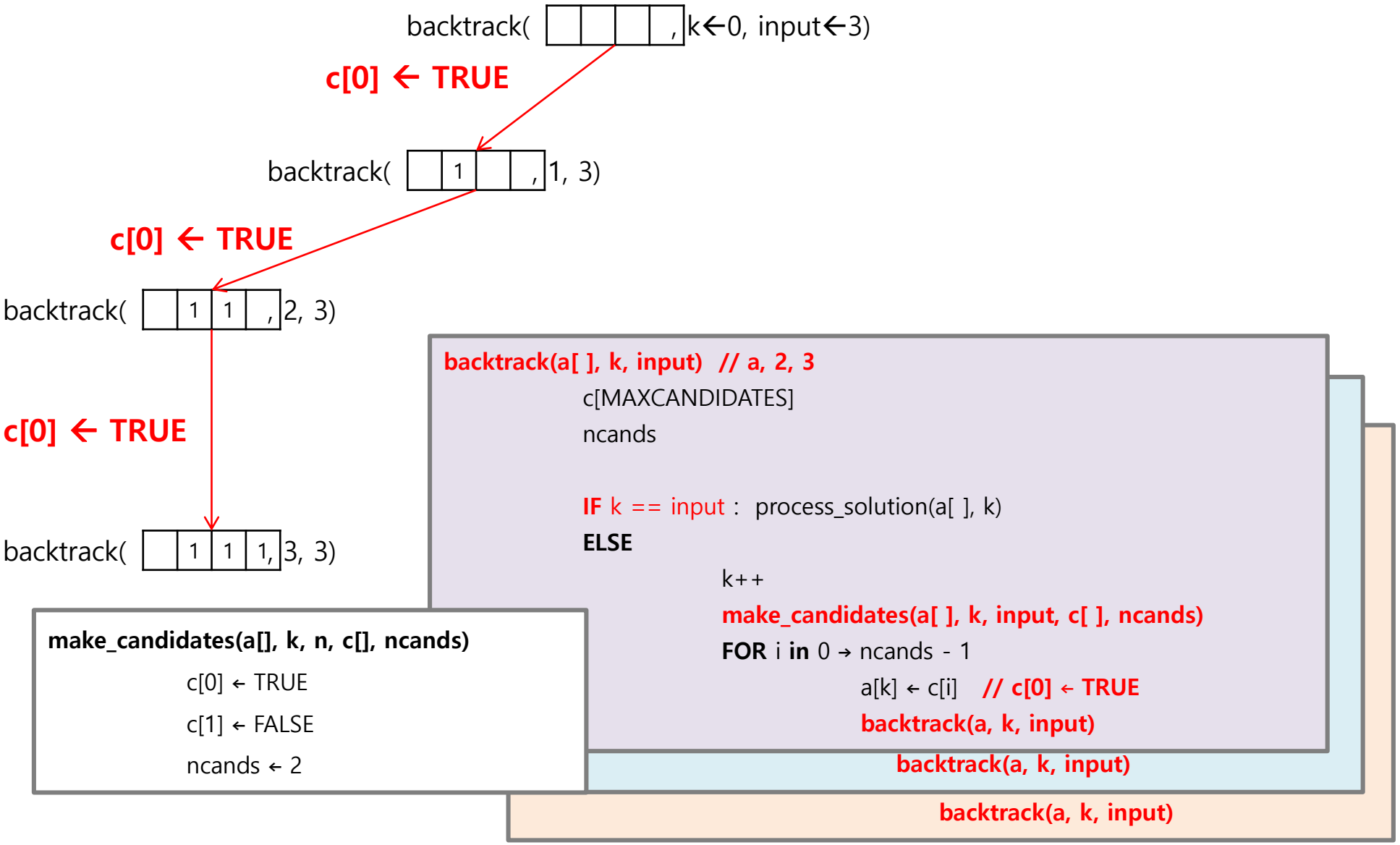
```
make_candidates(a[], k, n, c[], ncands)
        c[0] ← TRUE
        c[1] ← FALSE
        ncands ← 2
```

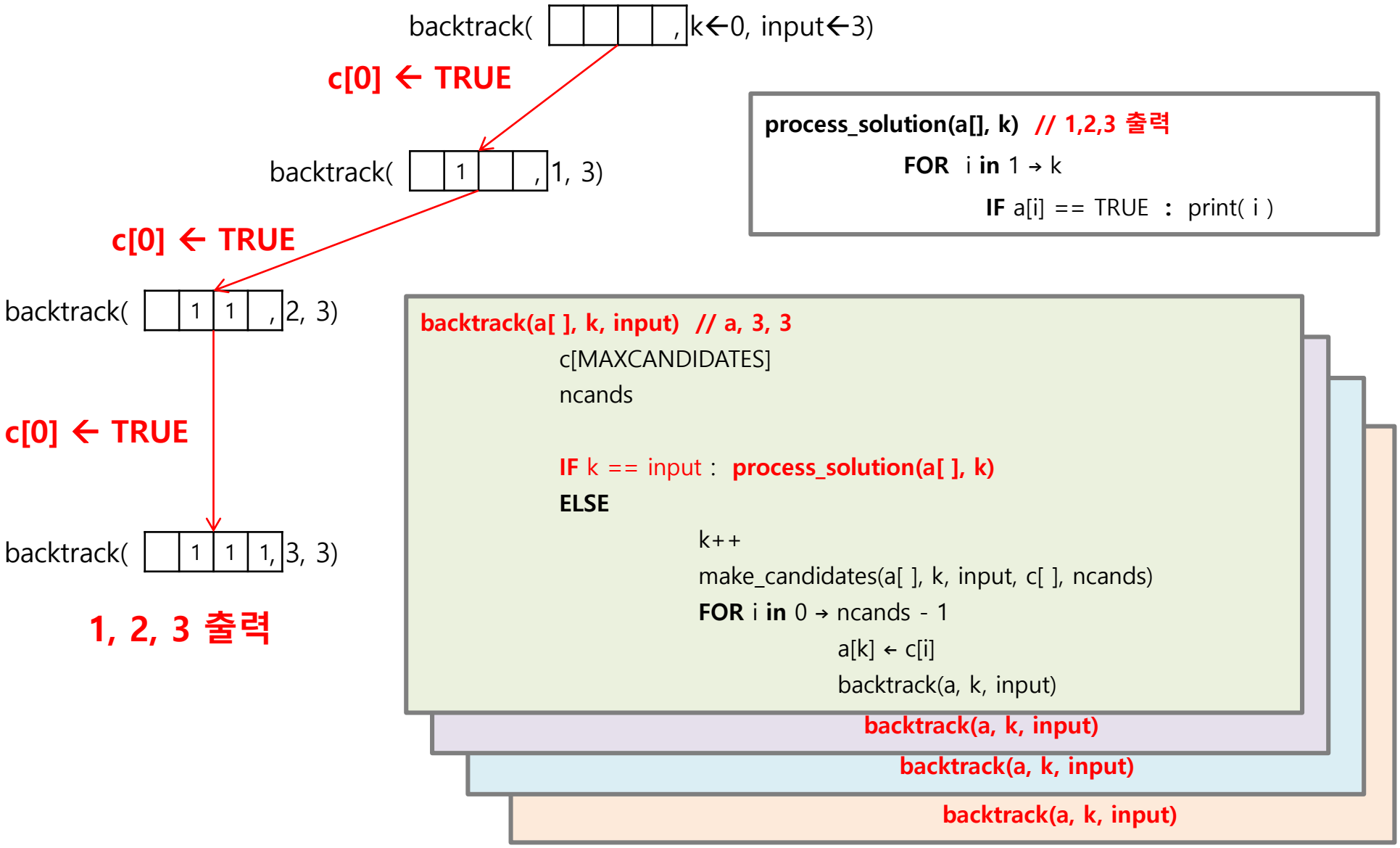


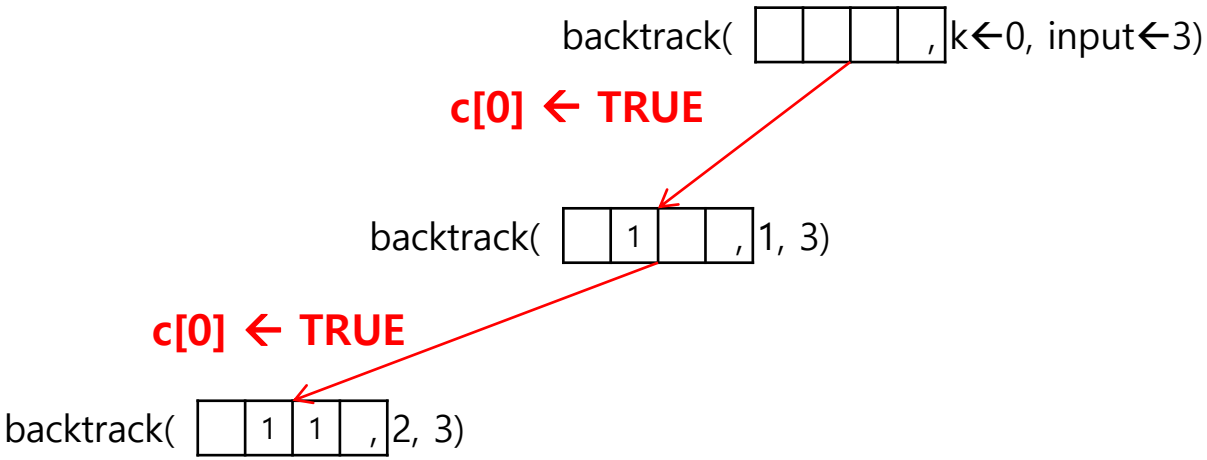

```
backtrack(a[ ], k, input)  // a, 1, 3
        c[MAXCANDIDATES]
        ncands

        IF k == input :  process_solution(a[ ], k)
        ELSE
                k++
                make_candidates(a[ ], k, input, c[ ], ncands)
                FOR i in 0 → ncands - 1
                        a[k] ← c[i]    // c[0] ← TRUE
                        backtrack(a, k, input)
```

```
make_candidates(a[], k, n, c[], ncands)
        c[0] ← TRUE
        c[1] ← FALSE
        ncands ← 2
```

```
backtrack(a, k, input)
```


backtrack(, , , , k←0, input←3)
c[0] ← TRUE
backtrack(1 , , 1, 3)
c[0] ← TRUE
backtrack(1 1 , 2, 3)
c[0] ← TRUE
backtrack(1 1 1, 3, 3)
backtrack(a[ ], k, input) // a, 2, 3
c[MAXCANDIDATES]
ncands
IF k == input : process_solution(a[ ], k)
ELSE
k++
make_candidates(a[ ], k, input, c[ ], ncands)
FOR i in 0 → ncands - 1
a[k] ← c[i] // c[0] ← TRUE
backtrack(a, k, input)
backtrack(a, k, input)
backtrack(a, k, input)
make_candidates(a[], k, n, c[], ncands)
c[0] ← TRUE
c[1] ← FALSE
ncands ← 2

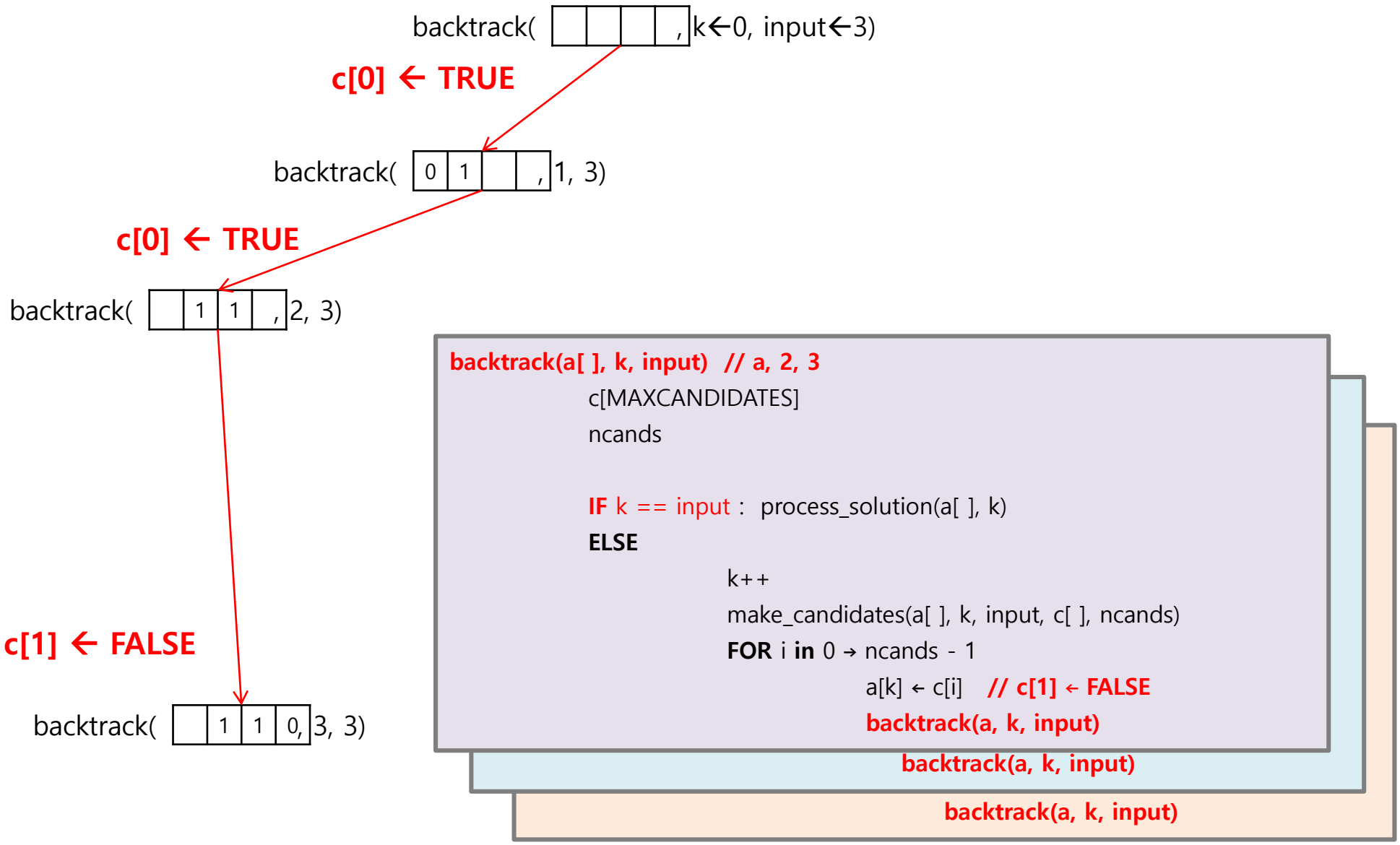

backtrack(, , , , k←0, input←3)
c[0] ← TRUE
backtrack(1 , , 1, 3)
c[0] ← TRUE
backtrack(1 1 , 2, 3)
c[0] ← TRUE
backtrack(1 1 1, 3, 3)
1, 2, 3 출력
process_solution(a[], k) // 1,2,3 출력
FOR i in 1 → k
IF a[i] == TRUE : print(i)
backtrack(a[ ], k, input) // a, 3, 3
c[MAXCANDIDATES]
ncands
IF k == input : process_solution(a[ ], k)
ELSE
k++
make_candidates(a[ ], k, input, c[ ], ncands)
FOR i in 0 → ncands - 1
a[k] ← c[i]
backtrack(a, k, input)
backtrack(a, k, input)
backtrack(a, k, input)
backtrack(a, k, input)

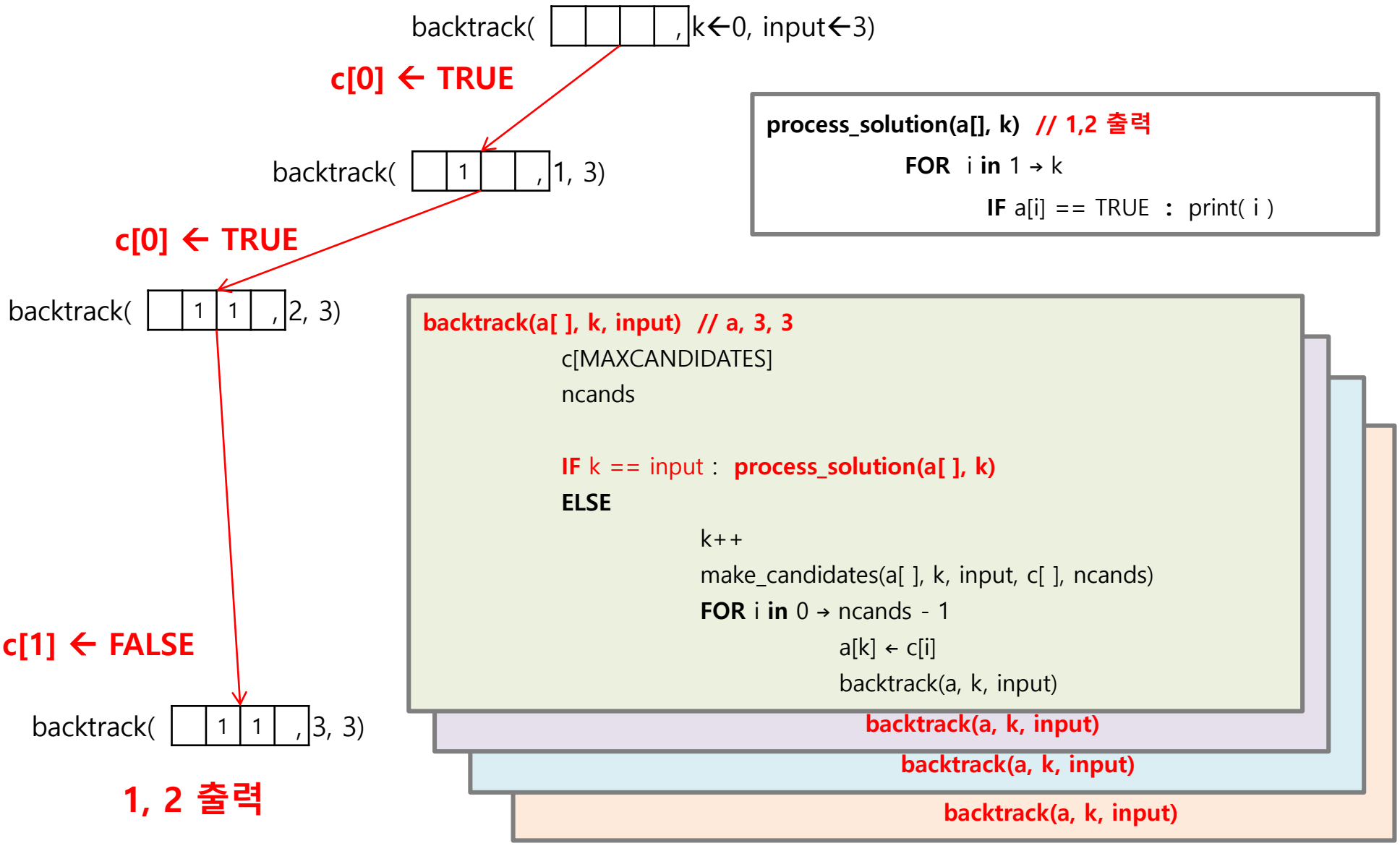


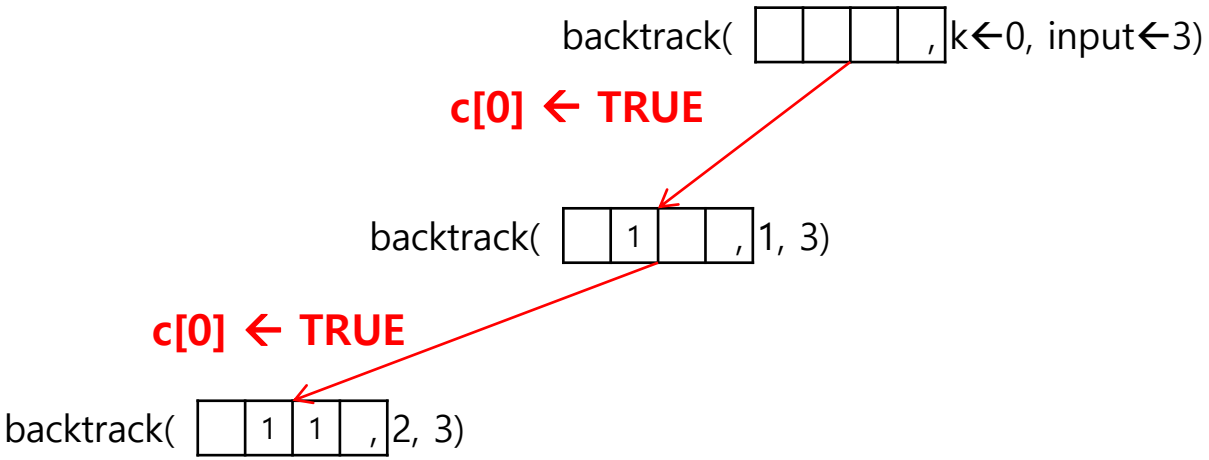

```
backtrack(a[ ], k, input)  // a, 2, 3
        c[MAXCANDIDATES]
        ncands

        IF k == input :  process_solution(a[ ], k)
        ELSE
                k++
                make_candidates(a[ ], k, input, c[ ], ncands)
                FOR i in 0 → ncands - 1
                        a[k] ← c[i]    // c[1] ← FALSE
                        backtrack(a, k, input)
```

backtrack(a, k, input)

backtrack(a, k, input)


backtrack(, , , , k←0, input←3)
c[0] ← TRUE
backtrack(0 1 , , 1, 3)
c[0] ← TRUE
backtrack(1 1 , , 2, 3)
c[1] ← FALSE
backtrack(1 1 0, 3, 3)
backtrack(a[ ], k, input) // a, 2, 3
c[MAXCANDIDATES]
ncands
IF k == input : process_solution(a[ ], k)
ELSE
k++
make_candidates(a[ ], k, input, c[ ], ncands)
FOR i in 0 → ncands - 1
a[k] ← c[i] // c[1] ← FALSE
backtrack(a, k, input)
backtrack(a, k, input)
backtrack(a, k, input)

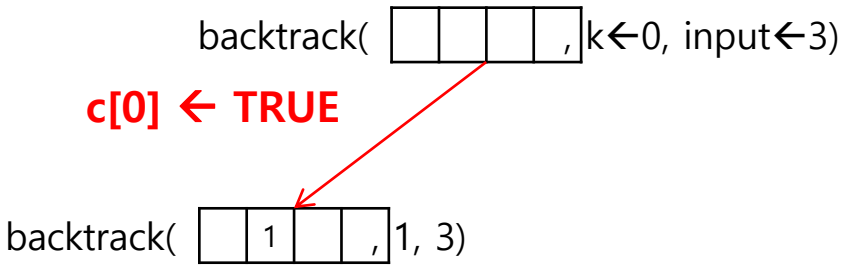

backtrack(, , , , k←0, input←3)
c[0] ← TRUE
backtrack(1 , , 1, 3)
c[0] ← TRUE
backtrack(1 1 , 2, 3)
c[1] ← FALSE
backtrack(1 1 , 3, 3)
1, 2 출력
process_solution(a[], k) // 1,2 출력
FOR i in 1 → k
IF a[i] == TRUE : print(i)
backtrack(a[ ], k, input) // a, 3, 3
c[MAXCANDIDATES]
ncands
IF k == input : process_solution(a[ ], k)
ELSE
k++
make_candidates(a[ ], k, input, c[ ], ncands)
FOR i in 0 → ncands - 1
a[k] ← c[i]
backtrack(a, k, input)
backtrack(a, k, input)
backtrack(a, k, input)
backtrack(a, k, input)




```
backtrack(a[ ], k, input)  // a, 2, 3
        c[MAXCANDIDATES]
        ncands

        IF k == input :  process_solution(a[ ], k)
        ELSE
                k++
                make_candidates(a[ ], k, input, c[ ], ncands)
                FOR i in 0 → ncands – 1 // 2개 모두 처리
                        a[k] ← c[i]
                        backtrack(a, k, input)
```

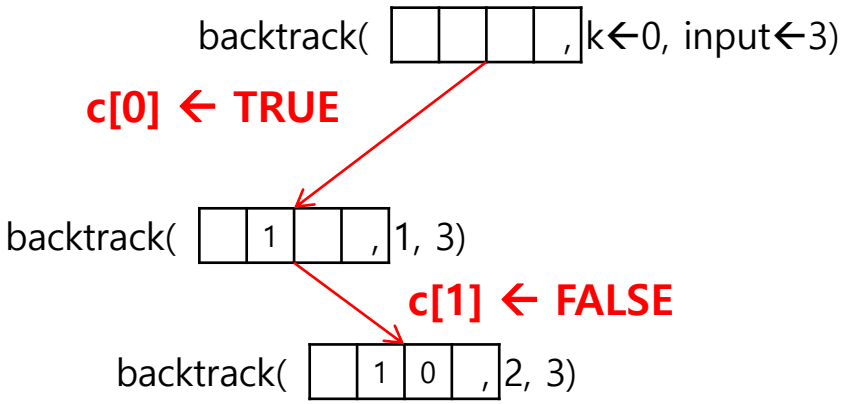
backtrack(a, k, input)

backtrack(a, k, input)



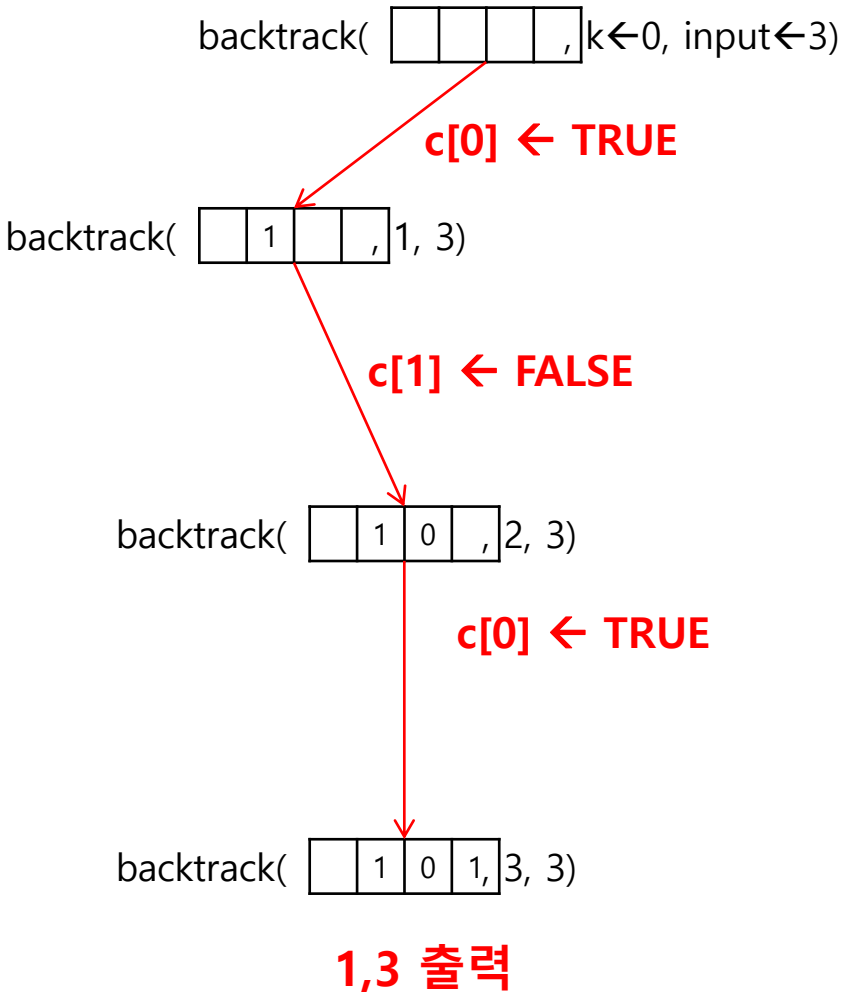

```
backtrack(a[ ], k, input)  // a, 1, 3
        c[MAXCANDIDATES]
        ncands

        IF k == input :  process_solution(a[ ], k)
        ELSE
                k++
                make_candidates(a[ ], k, input, c[ ], ncands)
                FOR i in 0 → ncands - 1
                        a[k] ← c[i]    // c[1] ← FALSE
                        backtrack(a, k, input)
```

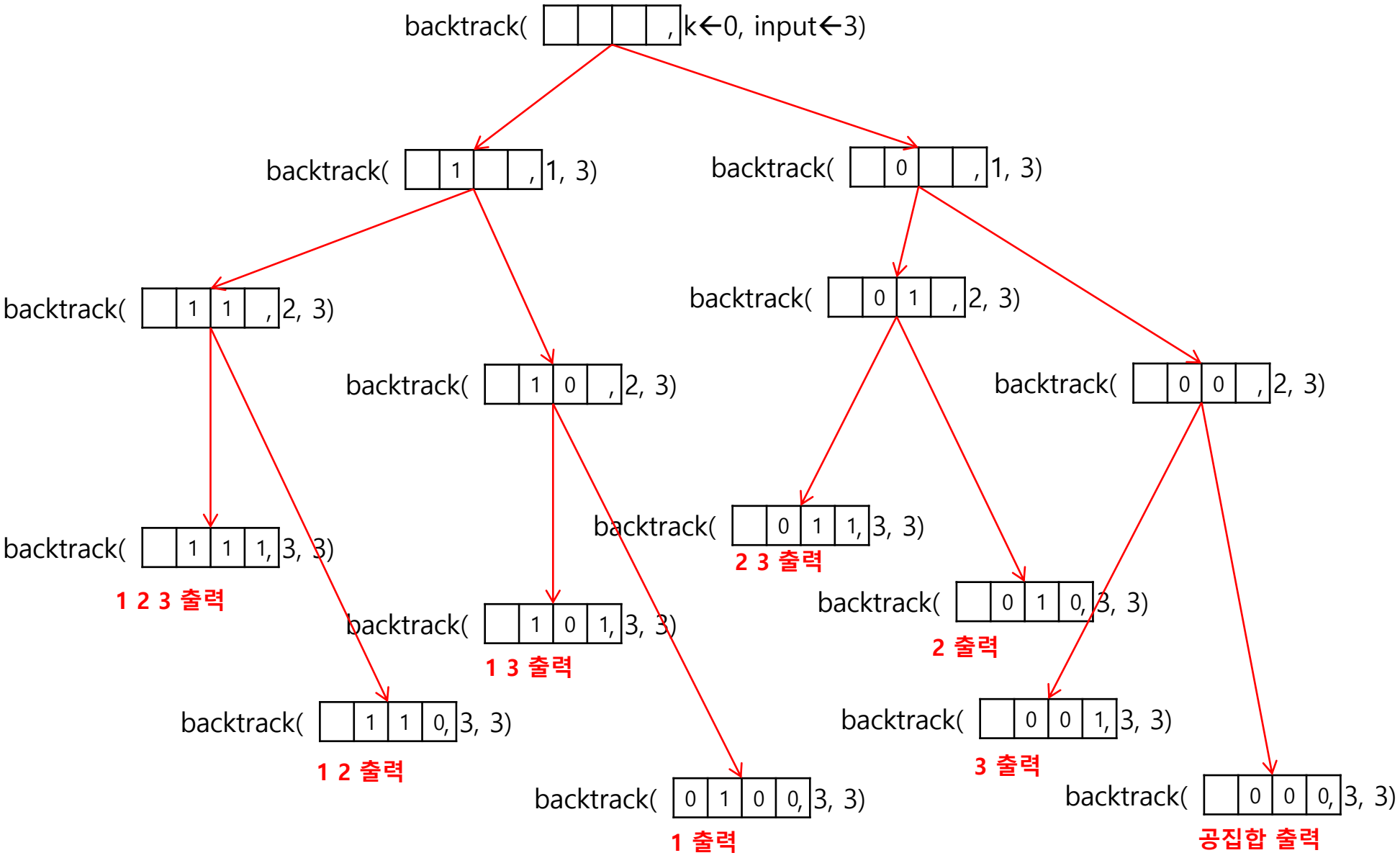
backtrack(a, k, input)




```
backtrack(a[ ], k, input)  // a, 1, 3
        c[MAXCANDIDATES]
        ncands

        IF k == input :  process_solution(a[ ], k)
        ELSE
                k++
                make_candidates(a[ ], k, input, c[ ], ncands)
                FOR i in 0 → ncands - 1
                        a[k] ← c[i]    // c[1] ← FALSE
                        backtrack(a, k, input)
```

backtrack(a, k, input)


backtrack(, , , , k←0, input←3)
c[0] ← TRUE
backtrack(1 , , 1, 3)
c[1] ← FALSE
backtrack(1 0 , 2, 3)
c[0] ← TRUE
backtrack(1 0 1, 3, 3)
1,3 출력


backtrack(, , , , k←0, input←3)
backtrack(, 1, , , 1, 3)
backtrack(, 0, , , 1, 3)
backtrack(, 1, 1, , 2, 3)
backtrack(, 1, 0, , 2, 3)
backtrack(, 0, 1, , 2, 3)
backtrack(, 0, 0, , 2, 3)
backtrack(, 1, 1, 1, 3, 3)
1 2 3 출력
backtrack(, 1, 0, 1, 3, 3)
1 3 출력
backtrack(, 1, 1, 0, 3, 3)
1 2 출력
backtrack(, 0, 1, 1, 3, 3)
2 3 출력
backtrack(, 0, 1, 0, 3, 3)
2 출력
backtrack(, 0, 0, 1, 3, 3)
3 출력
backtrack(, 0, 1, 0, 0, 3, 3)
1 출력
backtrack(, 0, 0, 0, 3, 3)
공집합 출력

상태공간트리를 구축하여 문제를 해결

백트래킹을 이용하여 순열 구하기

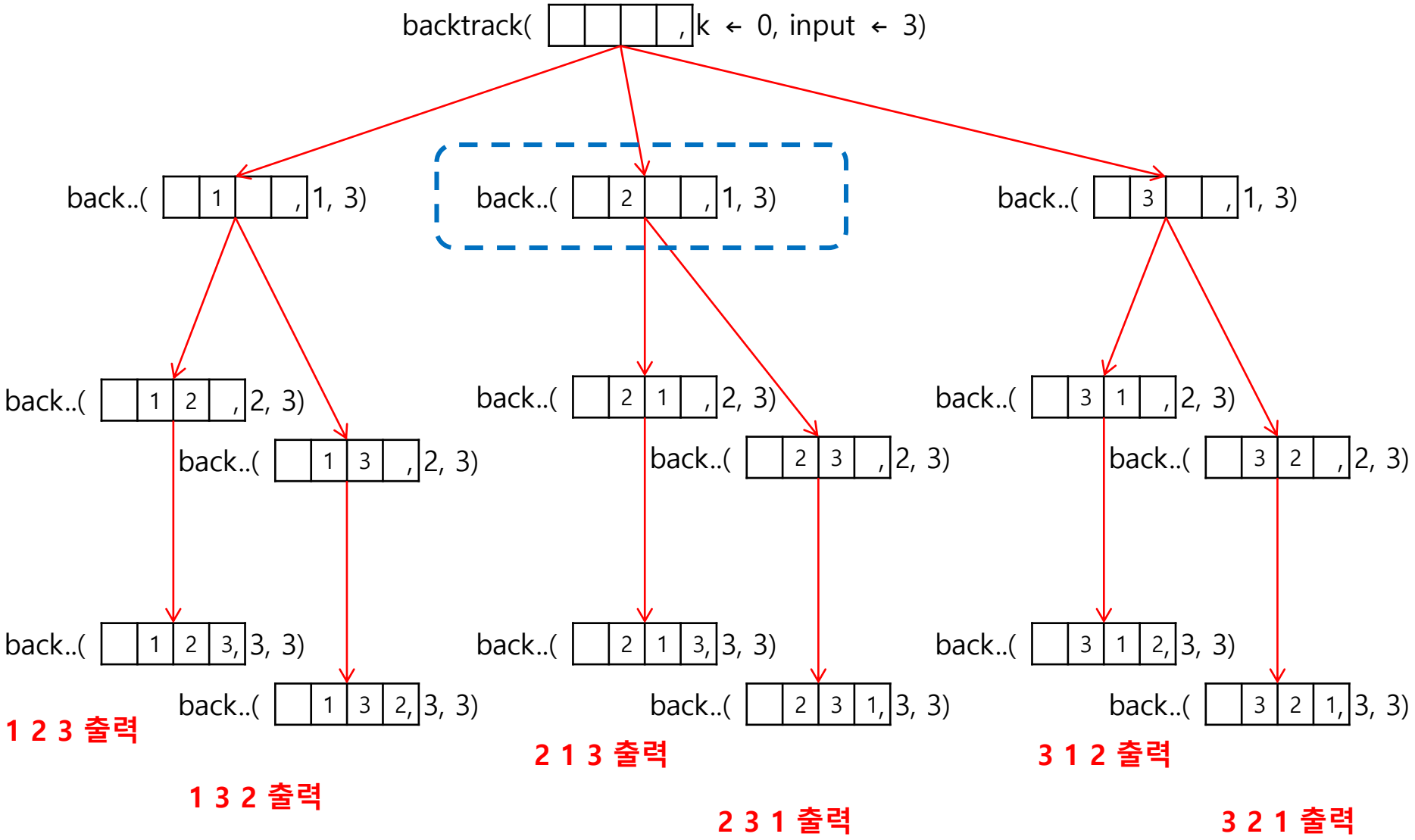
- 접근 방법은 앞의 부분집합 구하는 방법과 유사하다.

```
backtrack(a[ ], k, input)
          c[MAXCANDIDATES]
          ncands

          IF k == input :  process_solution(a[ ], k)
          ELSE
                    k++
                    make_candidates(a[ ], k, input, c[ ], ncands)
                    FOR i in 0 → ncands - 1
                              a[k] ← c[i]
                              backtrack(a, k, input)

main( )
          a[MAX]                          // 순열을 저장할 배열
          backtrack(a[ ], 0, 3)           // 3개의 원소를 가지는 순열
```

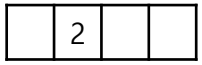
상태공간트리를 구축하여 문제를 해결

- 백트래킹을 이용하여 순열 구하기(계속)

```
make_candidates(a[], k, n, c[], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[i] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[i] == FALSE
                        c[ncands] ← i
                        ncands++


process_solution(a[], k)

        FOR  i in 1 → k  :  print( a[i] )
```

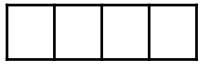

backtrack(, k ← 0, input ← 3)
back..(1 , 1, 3)
back..(2 , 1, 3)
back..(3 , 1, 3)
back..(1 2 , 2, 3)
back..(1 3 , 2, 3)
back..(2 1 , 2, 3)
back..(2 3 , 2, 3)
back..(3 1 , 2, 3)
back..(3 2 , 2, 3)
back..(1 2 3, 3, 3)
back..(1 3 2, 3, 3)
back..(2 1 3, 3, 3)
back..(2 3 1, 3, 3)
back..(3 1 2, 3, 3)
back..(3 2 1, 3, 3)
1 2 3 출력
1 3 2 출력
2 1 3 출력
2 3 1 출력
3 1 2 출력
3 2 1 출력

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] 

k ← 1

n ← 3

c[ ] 

ncands

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE


        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

in_perm[ ] | 0 | 0 | 0 | 0 |

k ← 1

n ← 3

c[ ] | | | | |

ncands

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

n ← 3

c[ ] | | | | |

ncands

in_perm[ ] | 0 | 0 | 1 | 0 |

i ← 1

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

in_perm[ ] | 0 | 0 | 1 | 0 |

k ← 1

i ← 1

n ← 3

c[ ] | | | | |

ncands ← 0

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

n ← 3

c[ ] | | | | |

ncands ← 0

in_perm[ ] | 0 | 0 | 1 | 0 |

i ← 1

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

n ← 3

c[ ] | | | | |

ncands ← 0

in_perm[ ] | 0 | 0 | 1 | 0 |

i ← 1

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

n ← 3

c[ ] | 1 | | | |

ncands ← 1

in_perm[ ] | 0 | 0 | 1 | 0 |

i ← 1

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

n ← 3

c[ ] | 1 | | | |

ncands ← 1

in_perm[ ] | 0 | 0 | 1 | 0 |

i ← 2

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

n ← 3

c[ ] | 1 | | | |

ncands ← 1

in_perm[ ] | 0 | 0 | 1 | 0 |

i ← 2

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

n ← 3

c[ ] | 1 | | | |

ncands ← 1

in_perm[ ] | 0 | 0 | 1 | 0 |

i ← 3

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

n ← 3

c[ ] | 1 | | | |

ncands ← 1

in_perm[ ] | 0 | 0 | 1 | 0 |

i ← 3

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

n ← 3

c[ ] | 1 | 3 | | |

ncands ← 2

in_perm[ ] | 0 | 0 | 1 | 0 |

i ← 3

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

n ← 3

c[ ] | 1 | 3 | | |

*ncandidates ← 2

in_perm[ ] | 0 | 0 | 1 | 0 |

i ← 4

```
make_candidates(a[ ], k, n, c[ ], ncands)
        in_perm[NMAX] ← FALSE

        FOR i in 1 → k - 1
                in_perm[ a[ i ] ] ← TRUE

        ncand ← 0
        FOR  i in 1 → n
                IF in_perm[ i ] == FALSE
                        c[ncands] ← i
                        ncands++
```

a[ ] | | 2 | | |

k ← 1

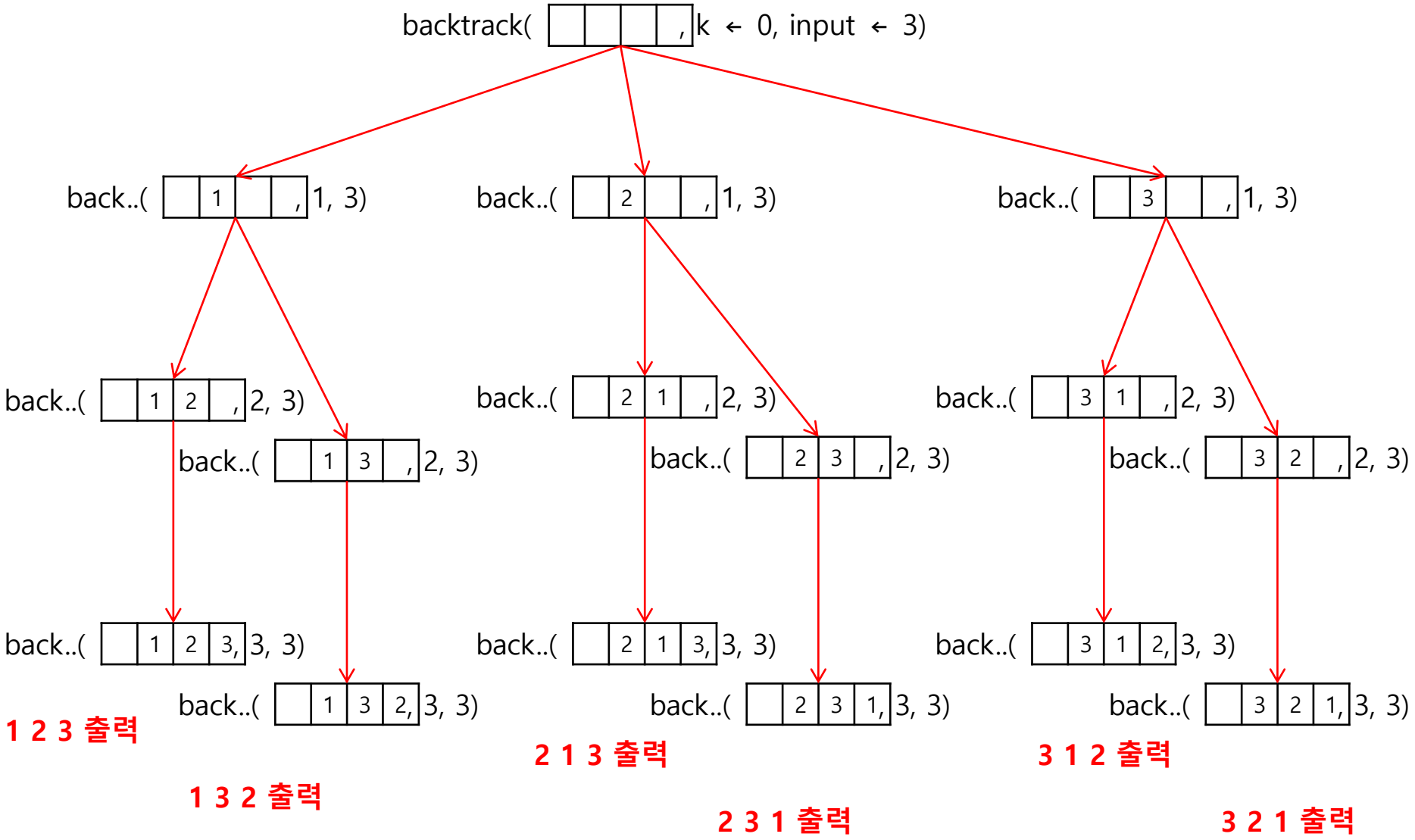
n ← 3

c[ ] | 1 | 3 | | |

ncands ← 2

in_perm[ ] | 0 | 0 | 1 | 0 |

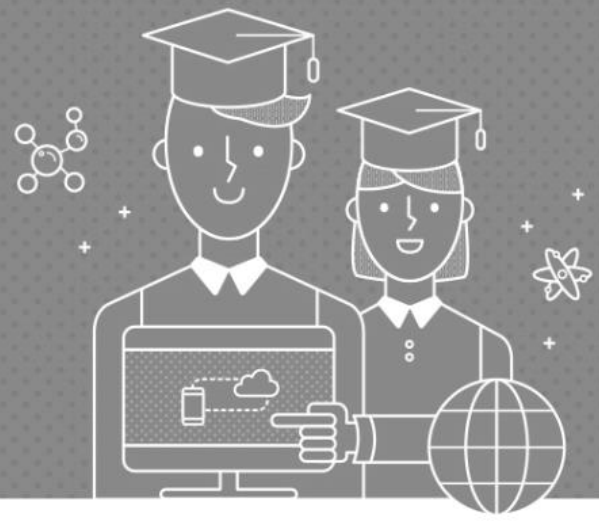
i ← 4


backtrack(, , , , k ← 0, input ← 3)
back..(1 , , 1, 3)
back..(2 , , 1, 3)
back..(3 , , 1, 3)
back..(1 2 , 2, 3)
back..(1 3 , 2, 3)
back..(2 1 , 2, 3)
back..(2 3 , 2, 3)
back..(3 1 , 2, 3)
back..(3 2 , 2, 3)
back..(1 2 3, 3, 3)
back..(1 3 2, 3, 3)
back..(2 1 3, 3, 3)
back..(2 3 1, 3, 3)
back..(3 1 2, 3, 3)
back..(3 2 1, 3, 3)
1 2 3 출력
1 3 2 출력
2 1 3 출력
2 3 1 출력
3 1 2 출력
3 2 1 출력

<연습문제2>

- {1,2,3,4,5,6,7,8,9,10}의 powerset 중 원소의 합이 10인 부분집합을 모두 출력하시오.

Quick Sort



퀵 정렬

- 주어진 배열을 두 개로 분할하고, 각각을 정렬한다.
  - 병합 정렬과 동일?

- 다른 점 1 : 병합 정렬은 그냥 두 부분으로 나누는 반면에, 퀵 정렬은 분할할 때, 기준 아이템(pivot item) 중심으로, 이보다 작은 것은 왼편, 큰 것은 오른편에 위치시킨다.

- 다른 점 2 : 각 부분 정렬이 끝난 후, 병합정렬은 “병합”이란 후처리 작업이 필요하나, 퀵 정렬은 필요로 하지 않는다.

퀵 정렬

- 알고리즘

```
quickSort(A[], l, r)
        if l < r
            s ← partition(a, l, r)
            quickSort(A[], l, s - 1)
            quickSort(A[], s + 1, r)
```

퀵 정렬

Hoare-Partition 알고리즘

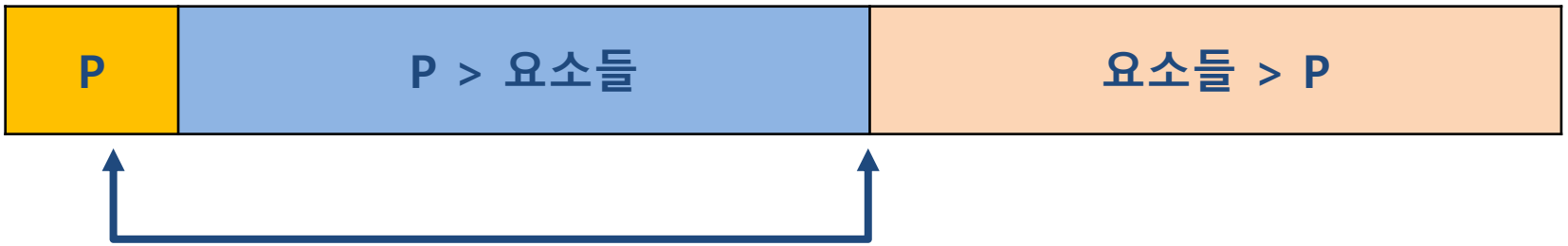
```
partition(A[], l, r)
        p ← A[l]                    // p: 피봇 값
        i ← l,  j ← r
        WHILE i ≤ j
            WHILE A[i] ≤ p : i++
            WHILE A[j] ≥ p : j--
            IF i < j : swap(A[i], A[j])

        swap(A[l], A[j])
        RETURN j
```

퀵 정렬

아이디어

- P(피봇)값들 보다 큰 값은 오른쪽, 작은 값들은 왼쪽 집합에 위치하도록 한다.

- 피봇을 두 집합의 가운데에 위치시킨다.



퀵 정렬

피봇 선택

- 왼쪽 끝/오른쪽 끝/임의의 세개 값 중에 중간 값

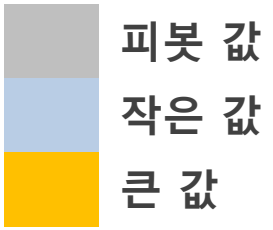
3	2	4	6	9	1	8	7	5

3	2	4	6	9	1	8	7	5

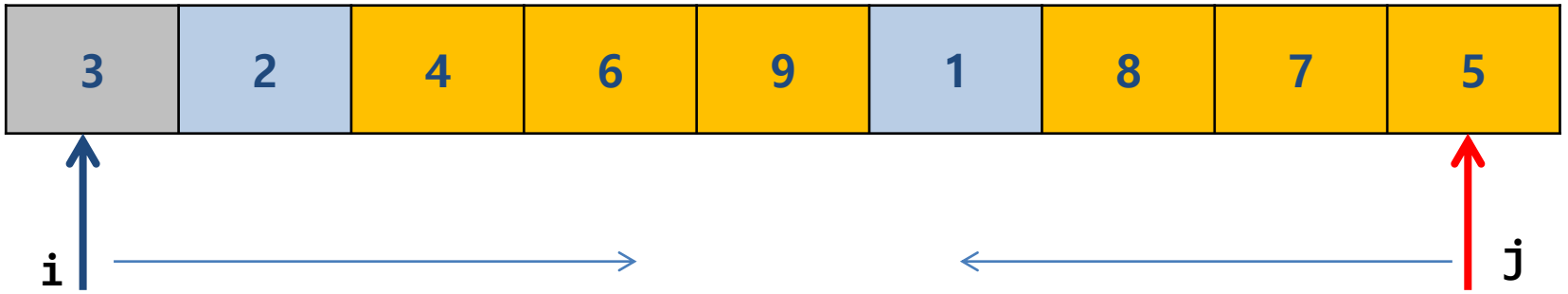
3	2	4	6	9	1	8	7	5

5	2	4	6	9	1	8	7	3

퀵 정렬

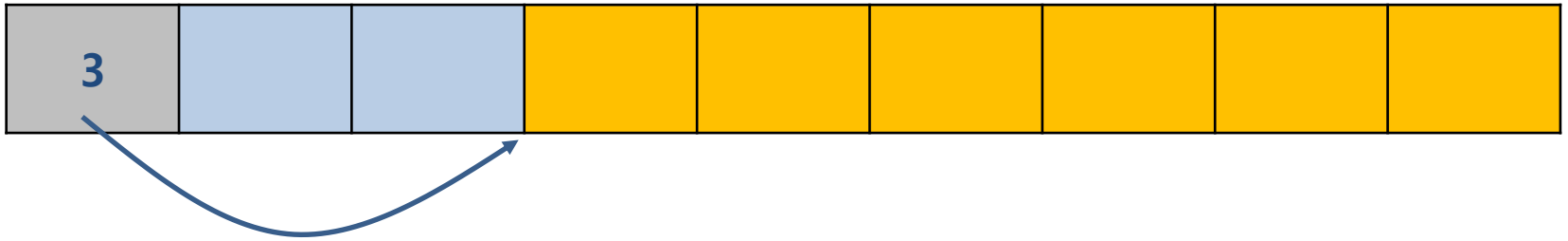


피봇

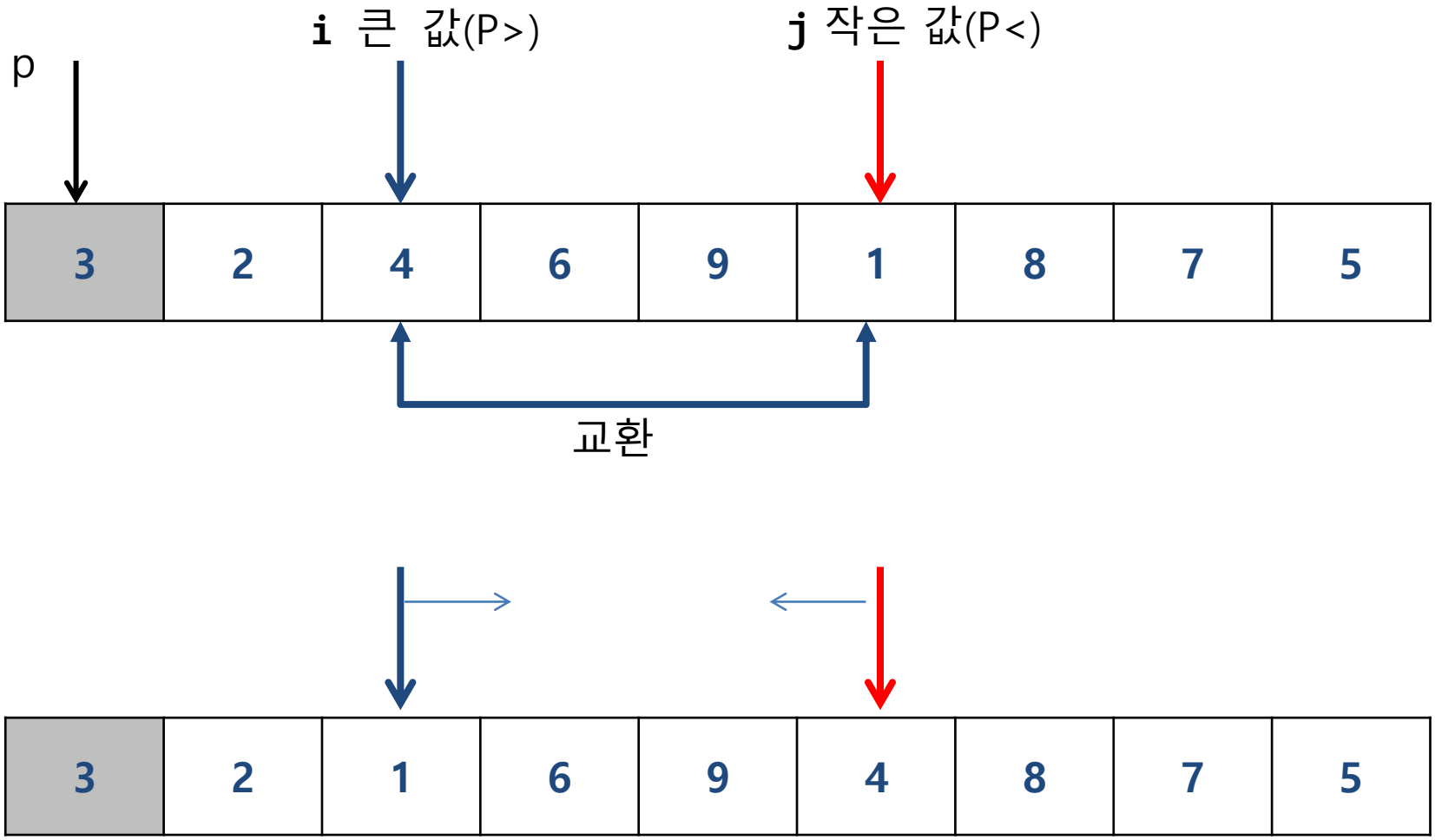


피봇 보다 큰 값 찾기

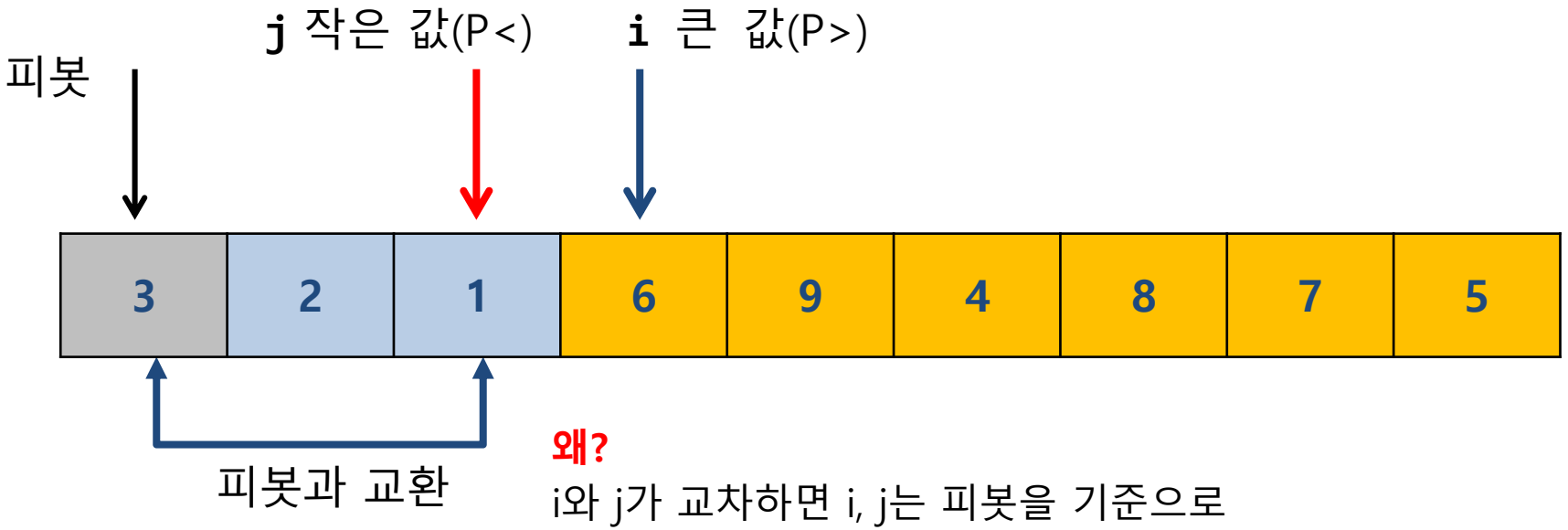
피봇 보다 작은 값 찾기



퀵 정렬



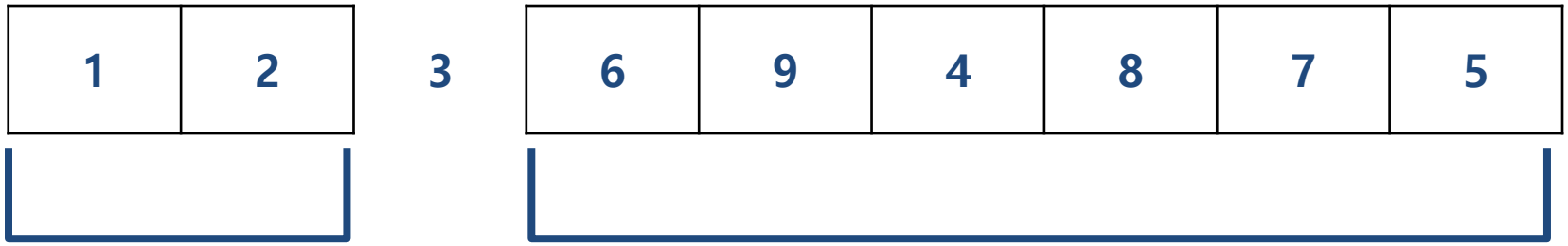
퀵 정렬

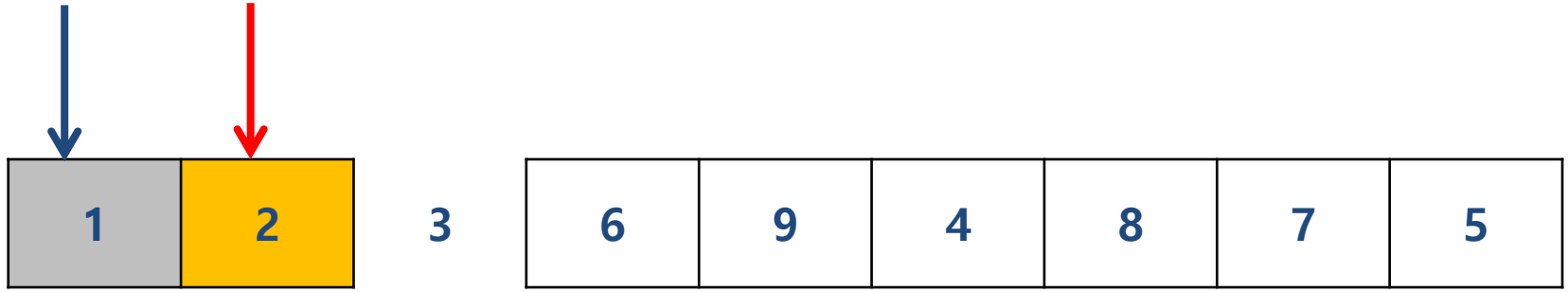


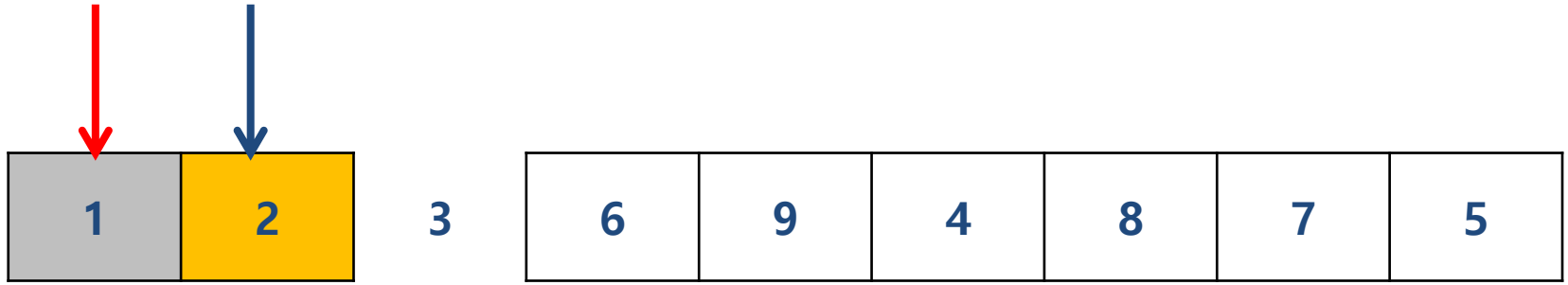
왜?

i와 j가 교차하면 i, j는 피봇을 기준으로 작은 값과 큰 값들의 경계에 위치한다.

퀵 정렬







퀵 정렬