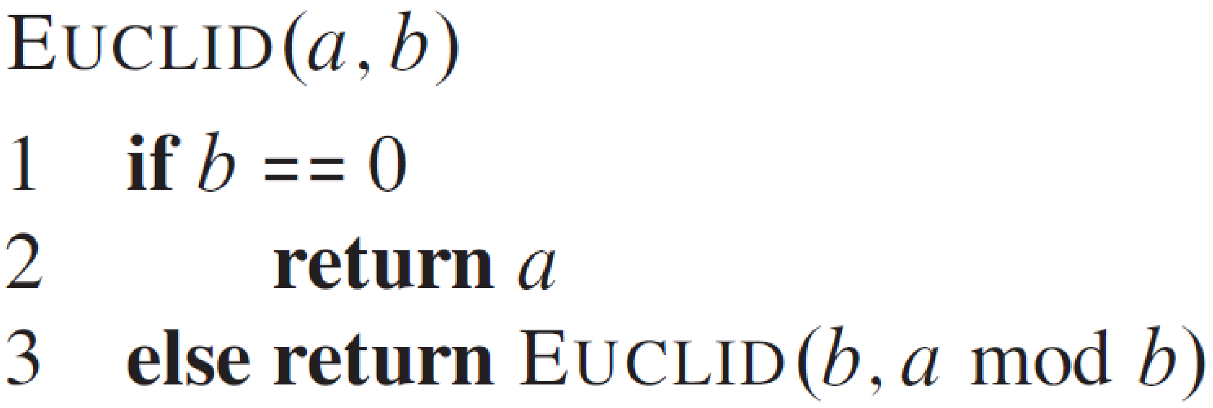
알고리즘 정리

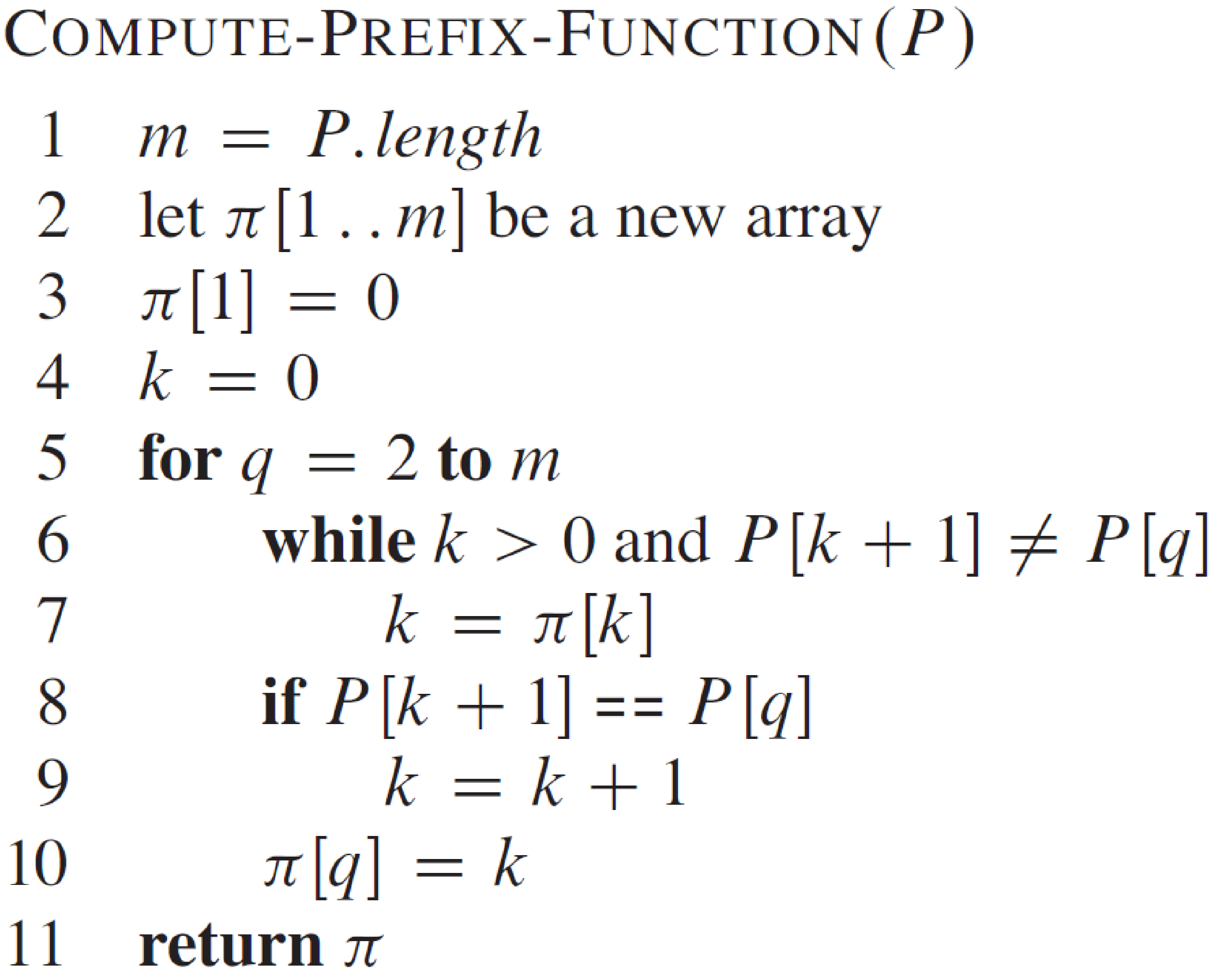
유클리디안 알고리즘

- $\gcd(a, b) = \gcd(b, a \bmod b)$



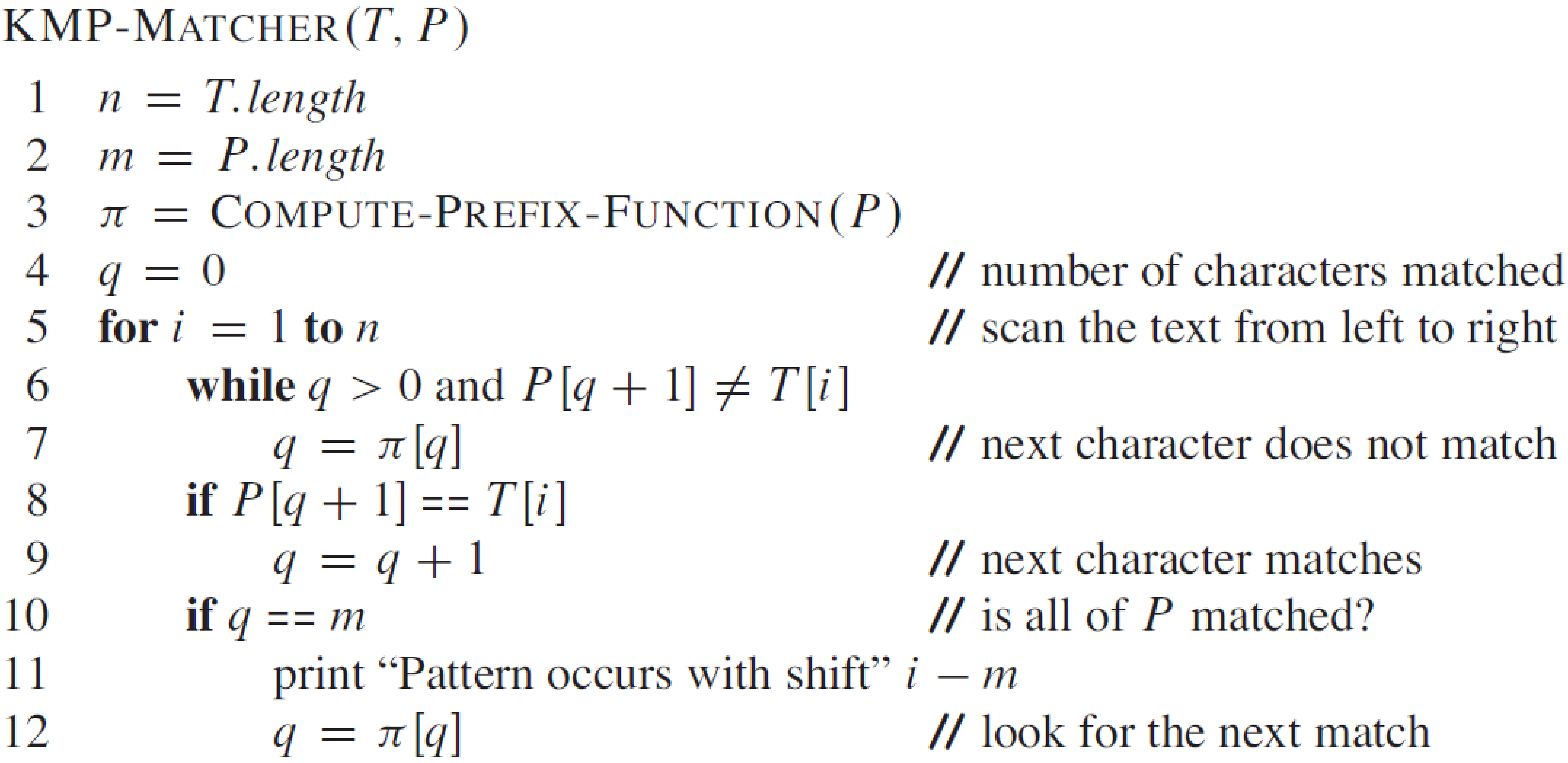
```
EUCLID($a, b$)
if $b == 0$
        return $a$
else return EUCLID($b, a \bmod b$)
```

KMP 알고리즘 (pi 배열 만들기)



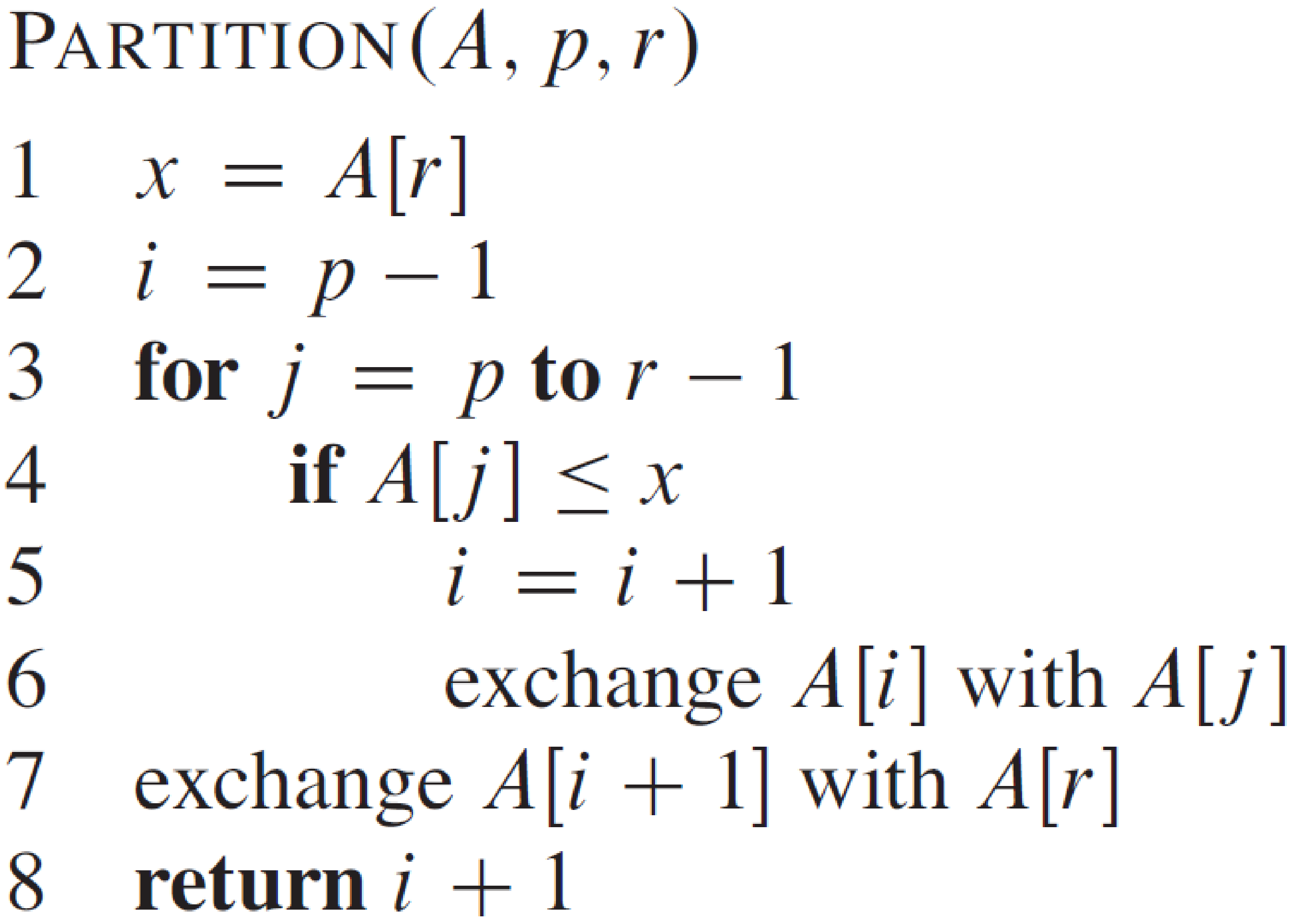
```
COMPUTE-PREFIX-FUNCTION(P)
m = P.length
let π[1..m] be a new array
π[1] = 0
k = 0
for q = 2 to m
    while k > 0 and P[k+1] ≠ P[q]
        k = π[k]
    if P[k+1] == P[q]
        k = k + 1
    π[q] = k
return π
```

KMP 알고리즘 (패턴 찾기)



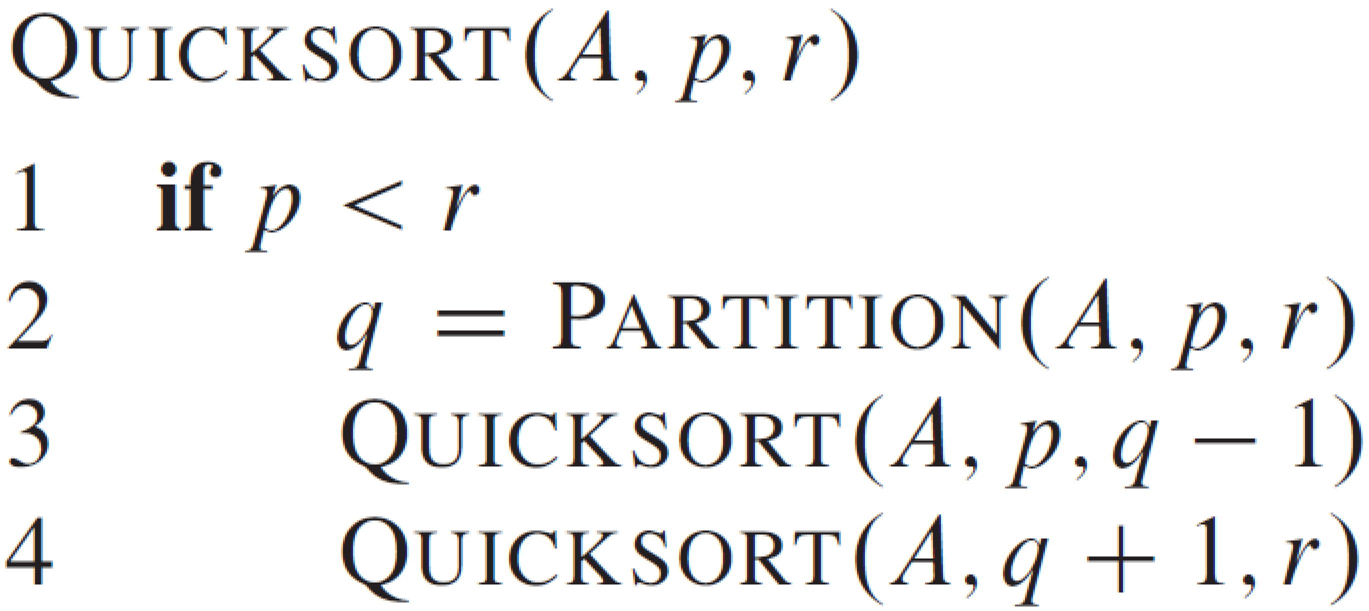
```
KMP-MATCHER(T, P)
  $n = T.length$
  $m = P.length$
  $\pi$ = COMPUTE-PREFIX-FUNCTION(P)
  $q = 0$                                    // number of characters matched
  for $i = 1$ to $n$                         // scan the text from left to right
      while $q > 0$ and $P[q+1] \neq T[i]$
          $q = \pi[q]$                       // next character does not match
      if $P[q+1] == T[i]$
          $q = q + 1$                        // next character matches
      if $q == m$                            // is all of P matched?
          print "Pattern occurs with shift" $i - m$
          $q = \pi[q]$                       // look for the next match
```

퀵 정렬 (파티션)



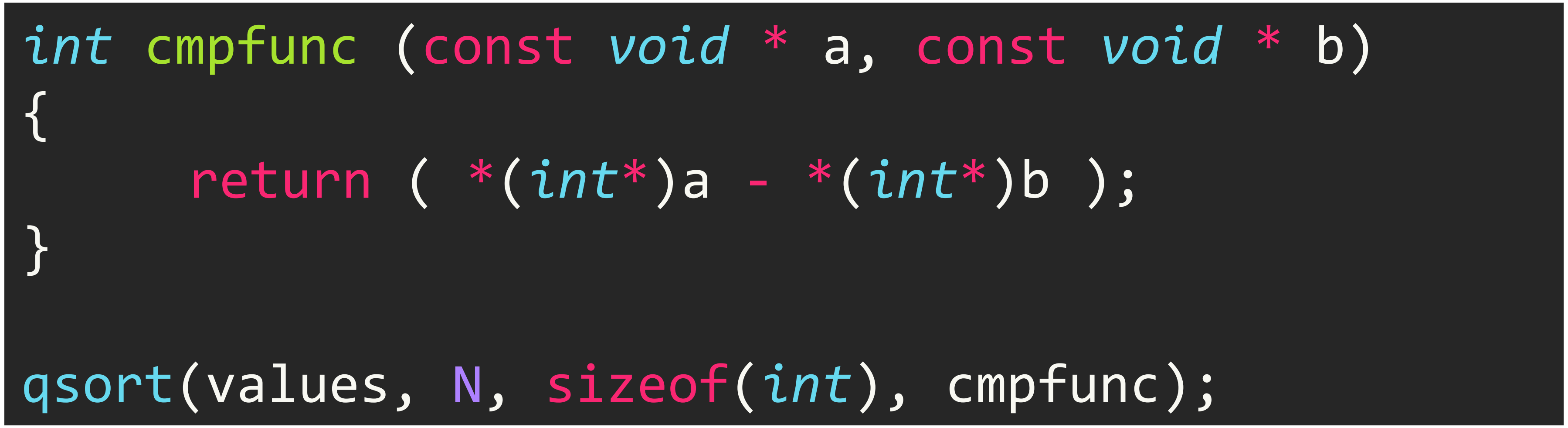
```
PARTITION($A, p, r$)
$x = A[r]$
$i = p - 1$
for $j = p$ to $r - 1$
        if $A[j] \leq x$
            $i = i + 1$
            exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$
```

퀵 정렬



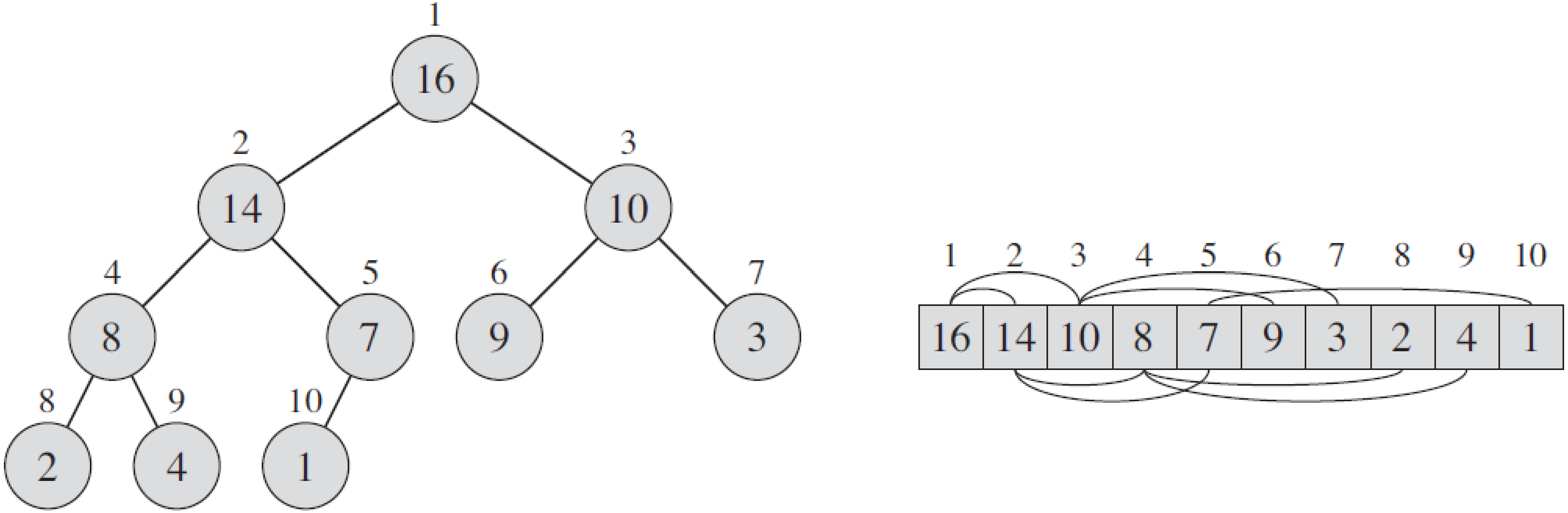
```
QUICKSORT($A, p, r$)
if $p < r$
    $q = $ PARTITION($A, p, r$)
    QUICKSORT($A, p, q - 1$)
    QUICKSORT($A, q + 1, r$)
```

퀵 정렬 (stdlib.h)



```
int cmpfunc (const void * a, const void * b)
{
    return ( *(int*)a - *(int*)b );
}

qsort(values, N, sizeof(int), cmpfunc);
```

힙

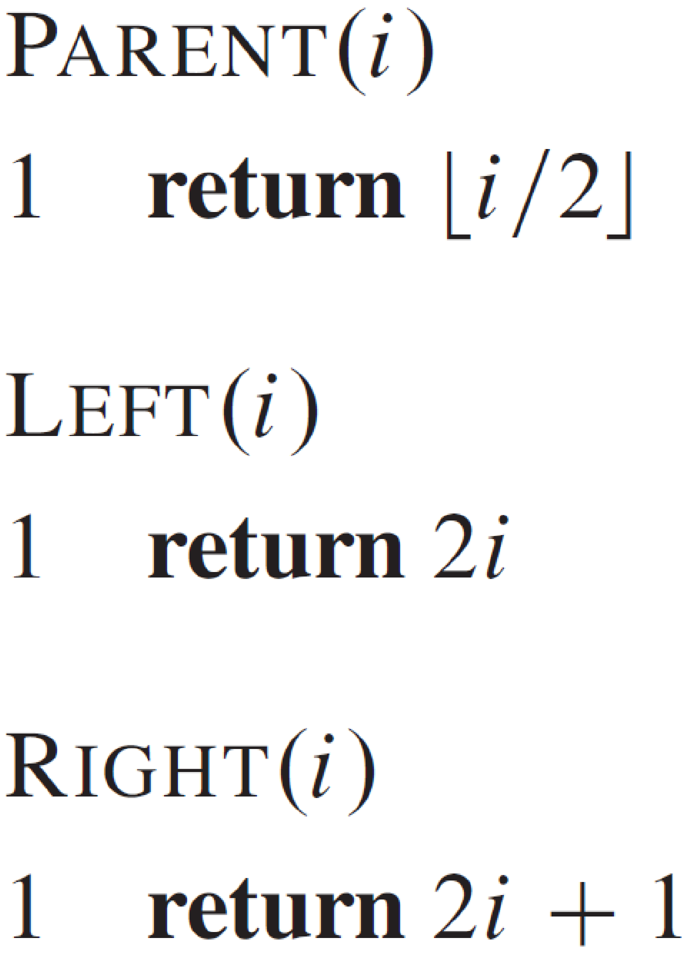


(a)

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

(b)

힙

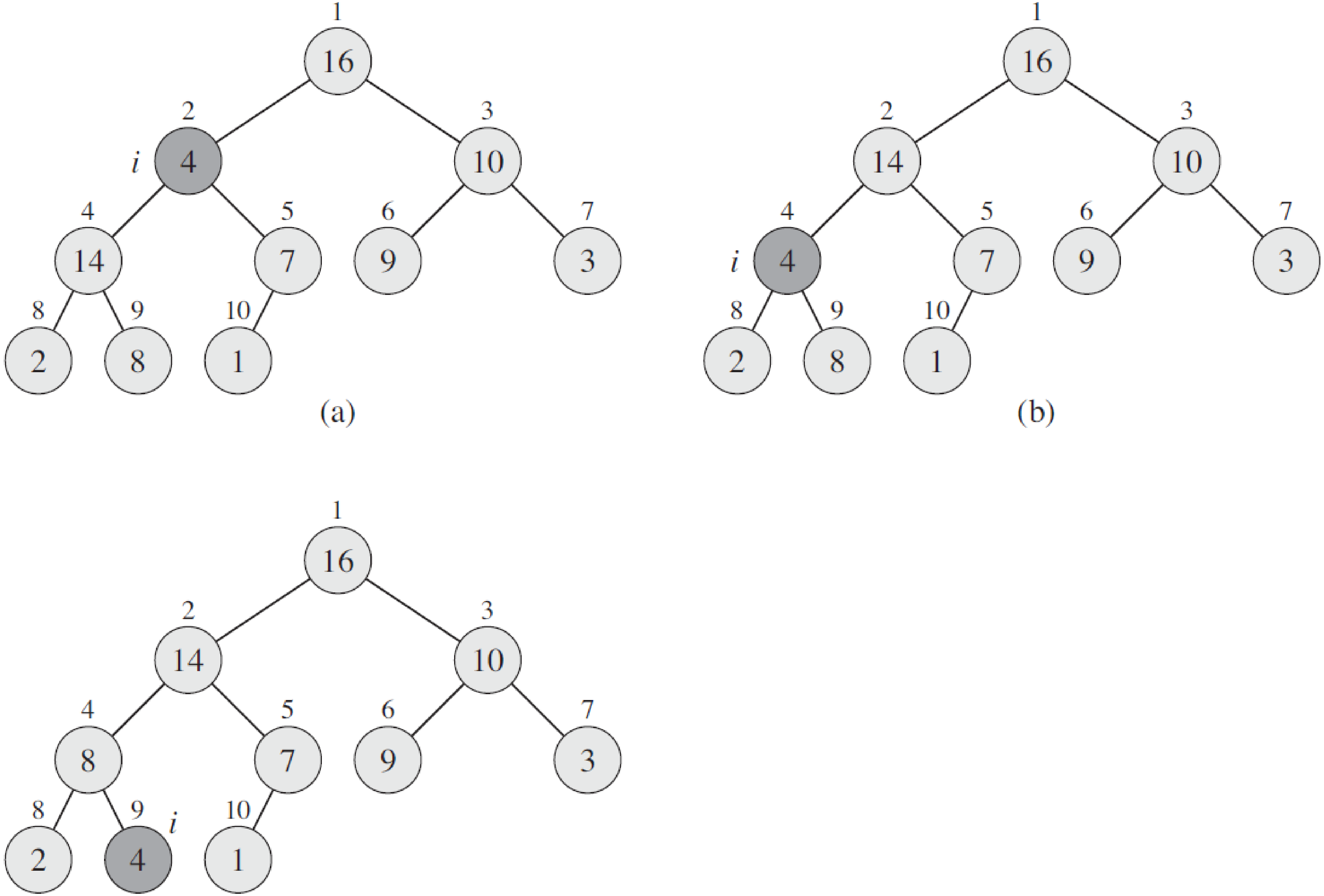


```
PARENT($i$)
    return $\lfloor i/2 \rfloor$

LEFT($i$)
    return $2i$

RIGHT($i$)
    return $2i + 1$
```

힙

```
MAX-HEAPIFY($A, i$)
$l = $ LEFT($i$)
$r = $ RIGHT($i$)
if $l \leq A.heap\text{-}size$ and $A[l] > A[i]$
    $largest = l$
else $largest = i$
if $r \leq A.heap\text{-}size$ and $A[r] > A[largest]$
    $largest = r$
if $largest \neq i$
    exchange $A[i]$ with $A[largest]$
    MAX-HEAPIFY($A, largest$)
```

힙

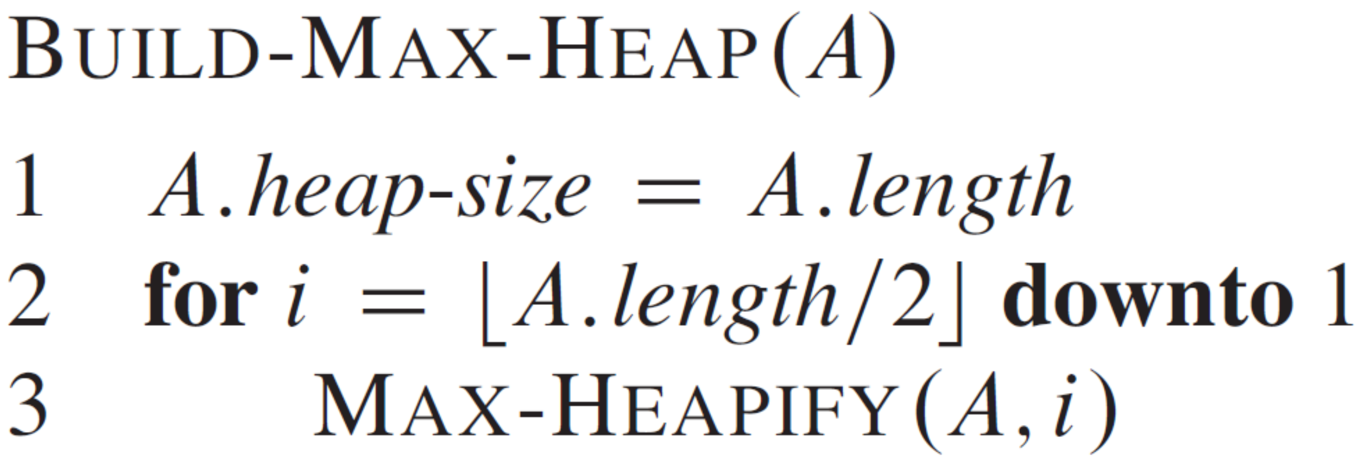

1
16
2
i
4
3
10
4
14
5
7
6
9
7
3
8
2
9
8
10
1
1
16
2
14
3
10
4
i
4
5
7
6
9
7
3
8
2
9
8
10
1
1
16
2
14
3
10
4
8
5
7
6
9
7
3
8
2
9
i
4
10
1

(a)

(b)

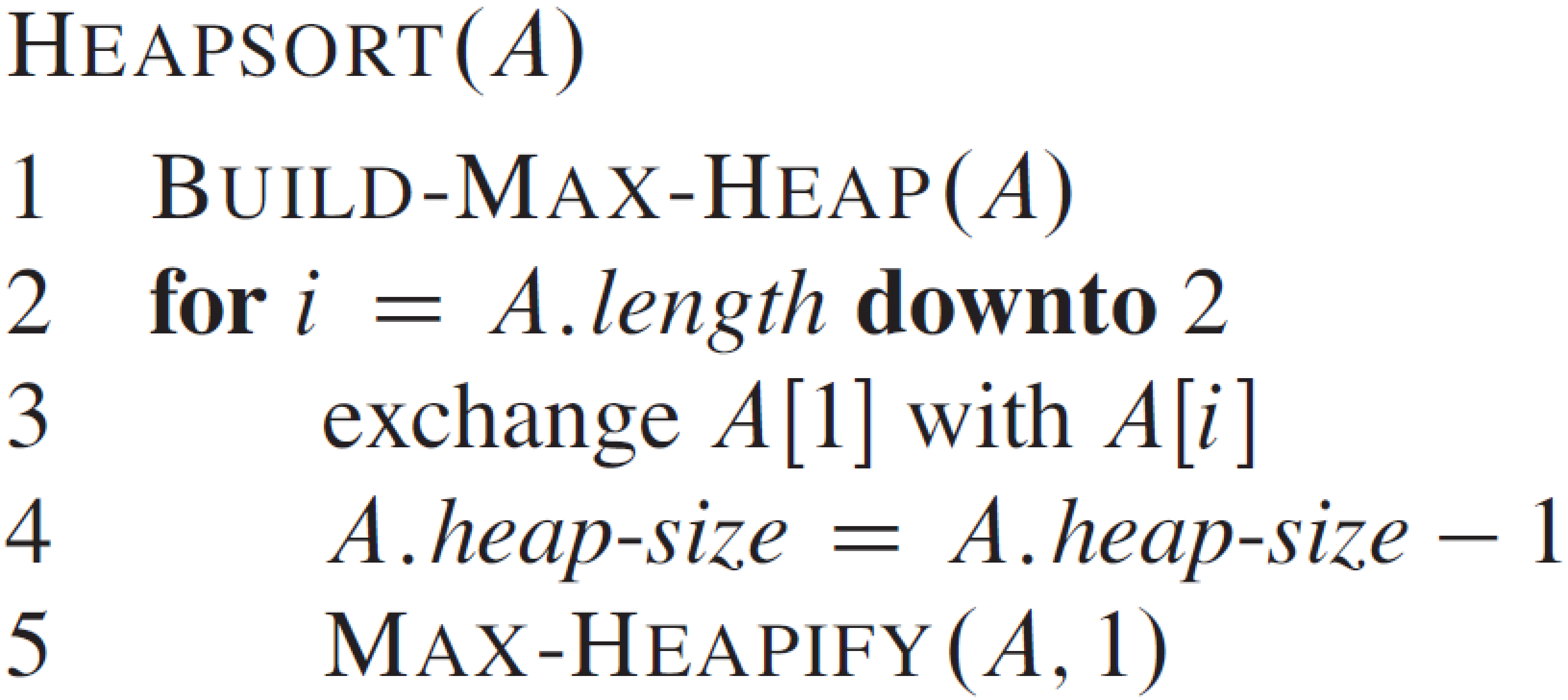
(c)

힙



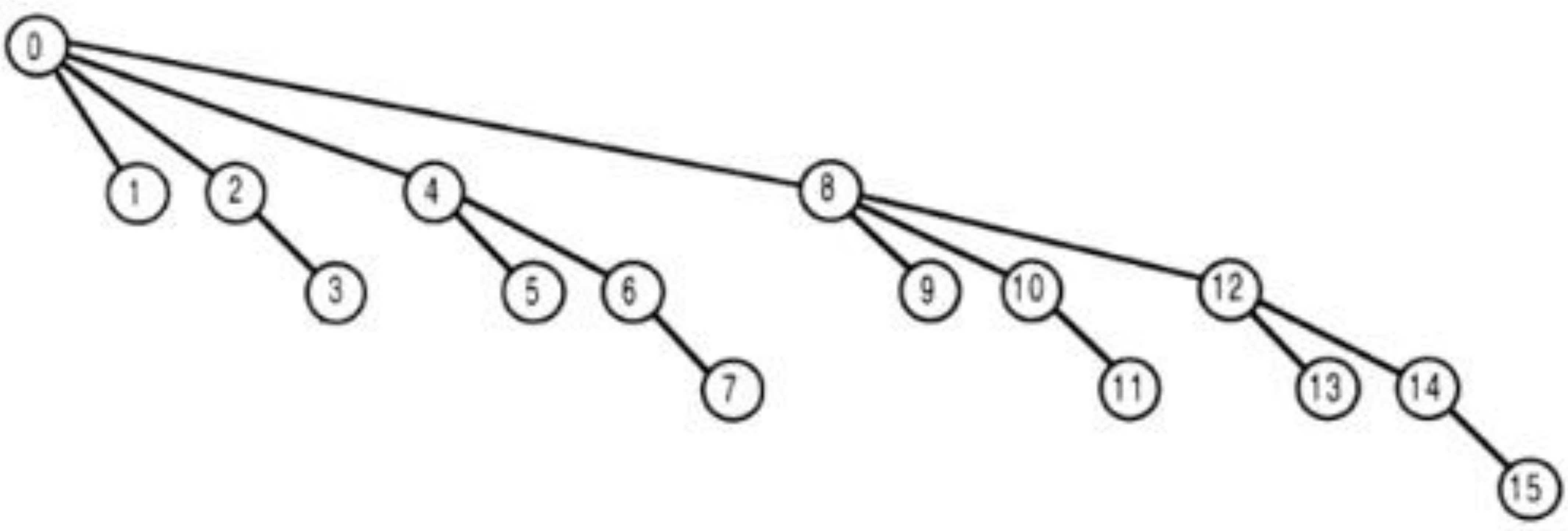
```
BUILD-MAX-HEAP(A)
$A.heap\text{-}size = A.length$
for $i = \lfloor A.length/2 \rfloor$ downto 1
        MAX-HEAPIFY(A, i)
```

힙 정렬



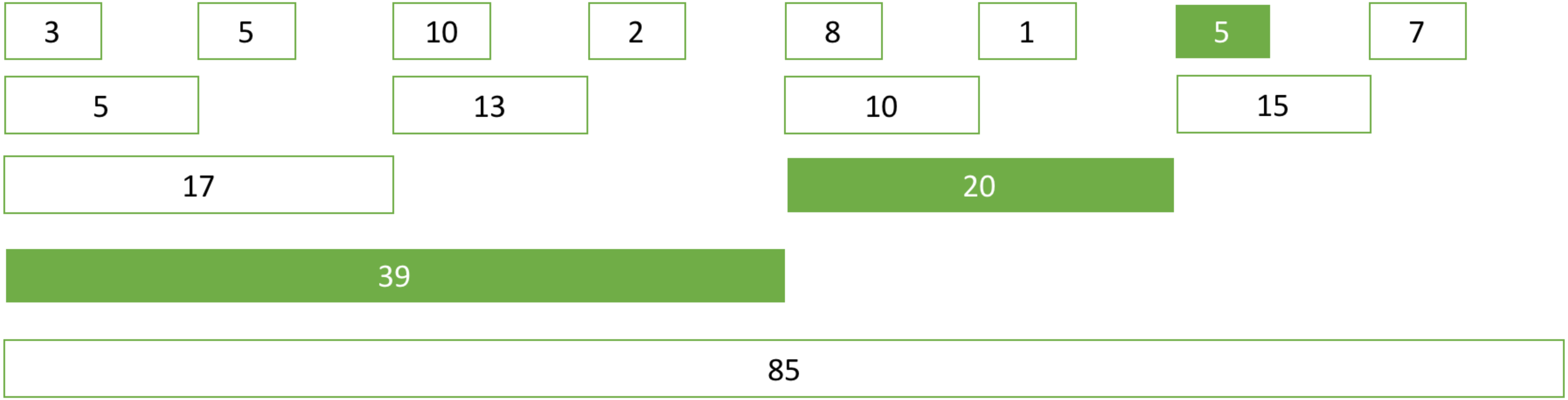
```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for $i = A.length$ downto 2
    exchange $A[1]$ with $A[i]$
    $A.heap\text{-}size = A.heap\text{-}size - 1$
    MAX-HEAPIFY(A, 1)
```

Fenwick tree (Binary Indexed Tree)

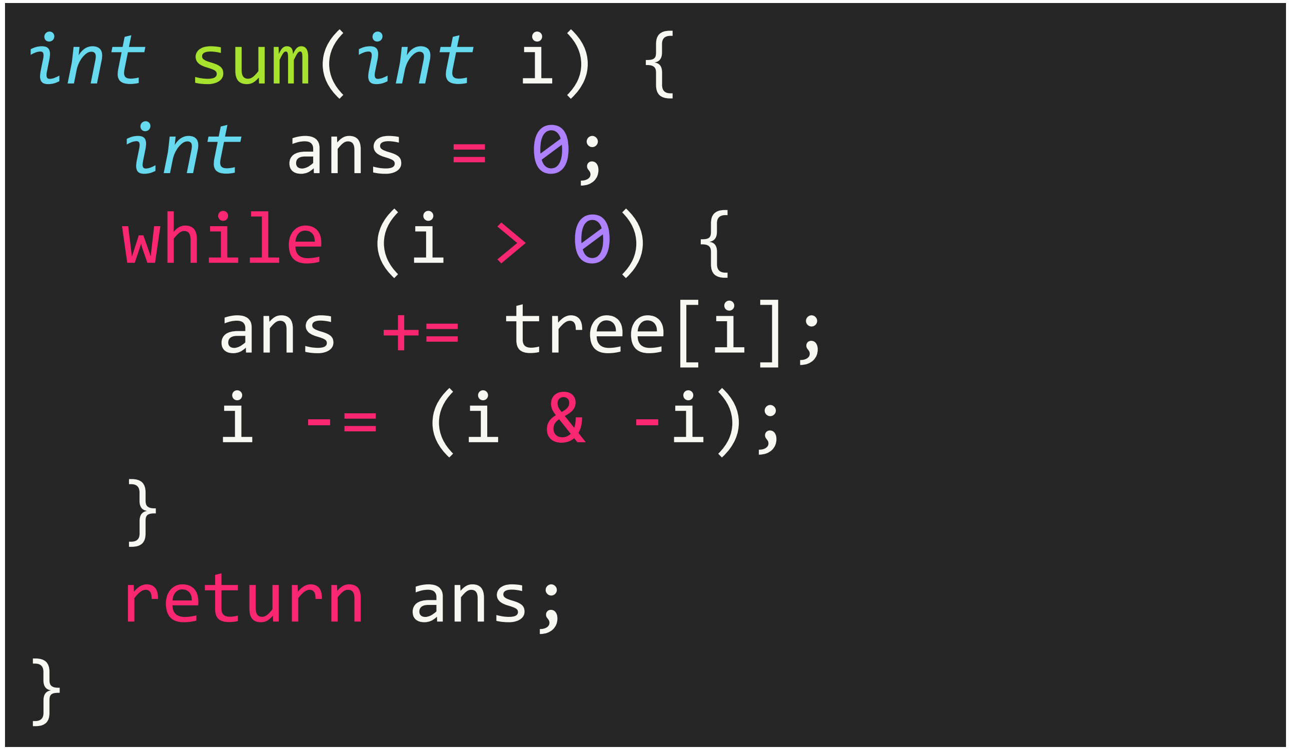


Fenwick tree (합 구하기)

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]	A[12]	A[13]	A[14]	A[15]	A[16]
3	2	5	7	10	3	2	7	8	2	1	9	5	10	7	4

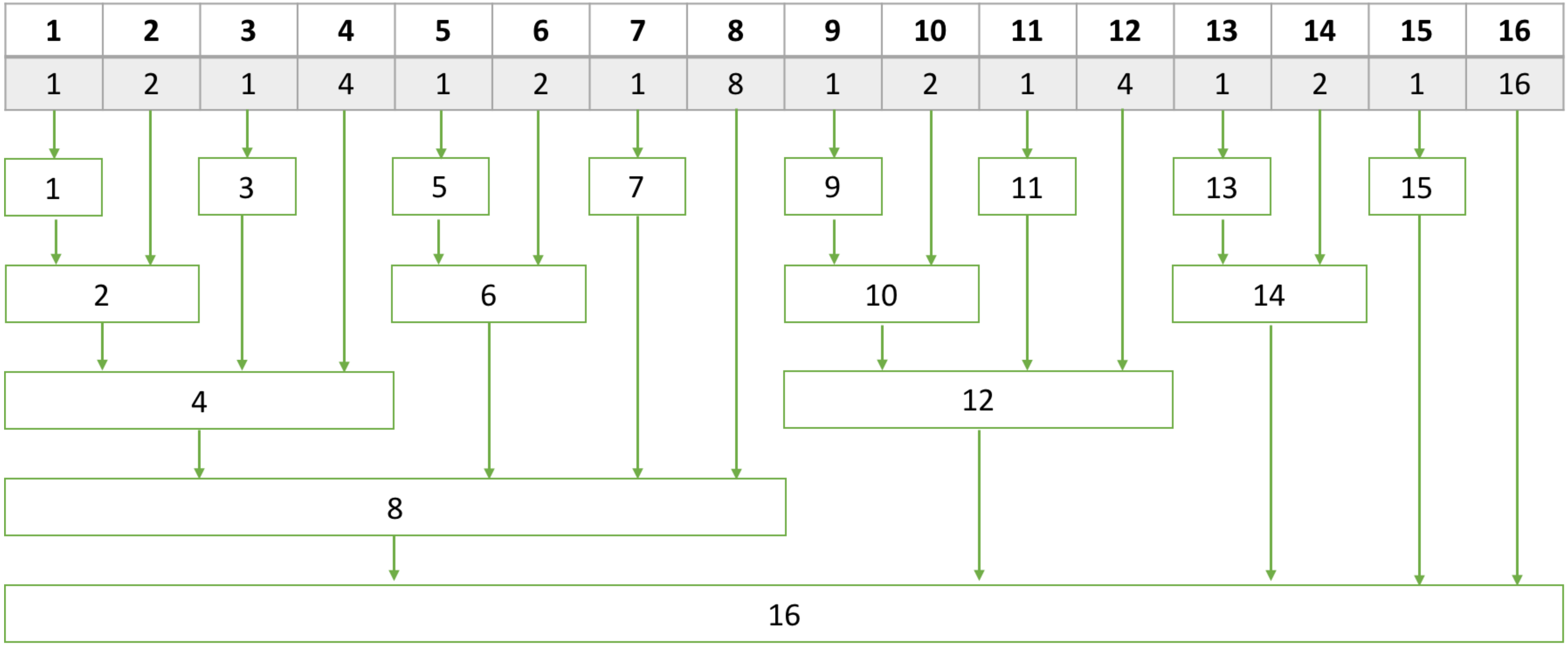


Fenwick tree (합 구하기)



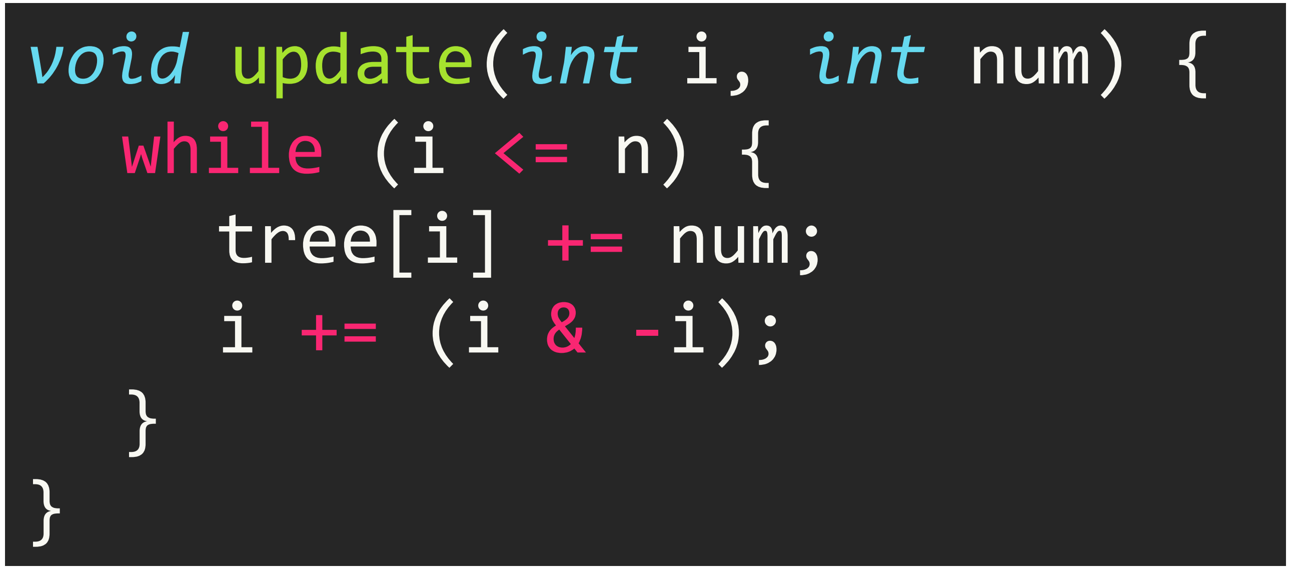
```
int sum(int i) {
   int ans = 0;
   while (i > 0) {
      ans += tree[i];
      i -= (i & -i);
   }
   return ans;
}
```

Fenwick tree (배열 업데이트)



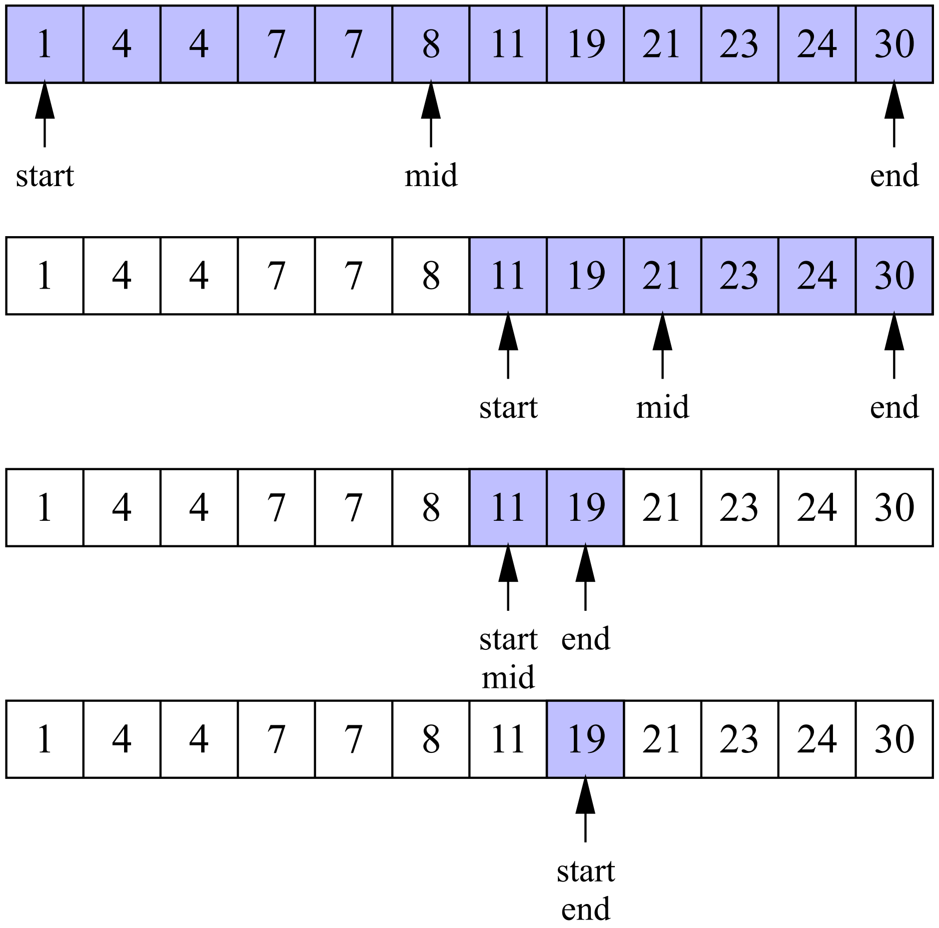
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	2	1	4	1	2	1	8	1	2	1	4	1	2	1	16

Fenwick tree (배열 업데이트)



```
void update(int i, int num) {
   while (i <= n) {
      tree[i] += num;
      i += (i & -i);
   }
}
```

Binary search



1	4	4	7	7	8	11	19	21	23	24	30

start mid end

1	4	4	7	7	8	11	19	21	23	24	30

start mid end

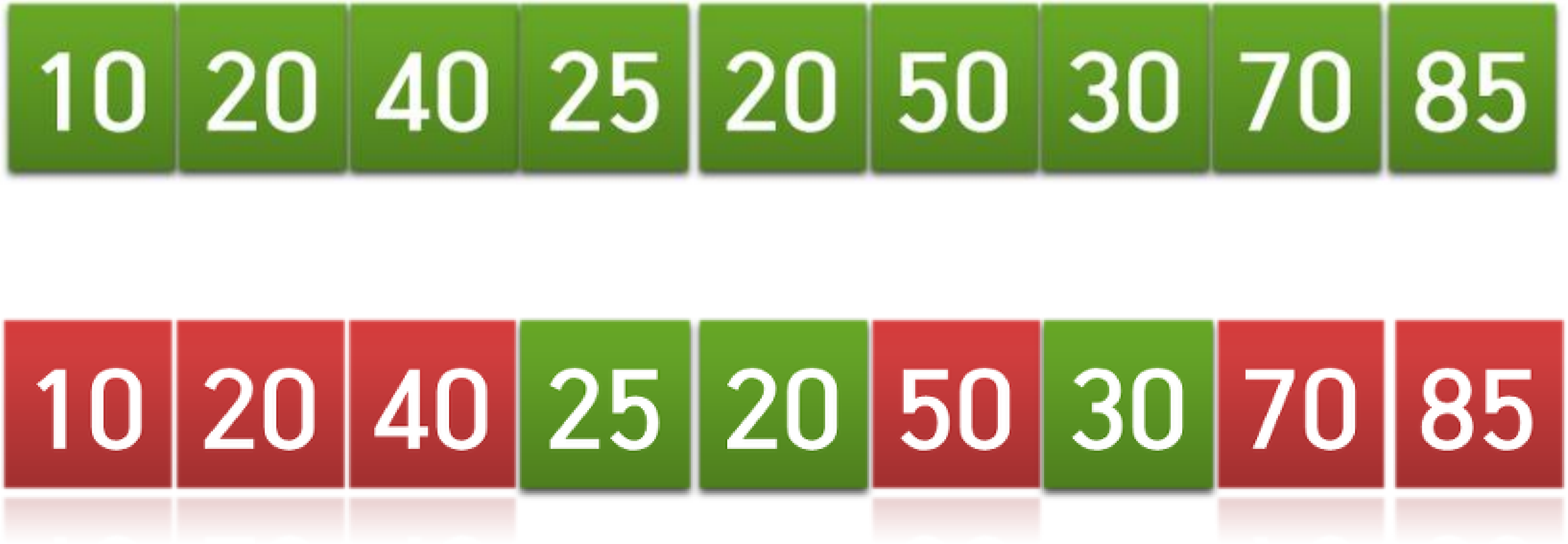
1	4	4	7	7	8	11	19	21	23	24	30

start
mid
end

1	4	4	7	7	8	11	19	21	23	24	30

start
end

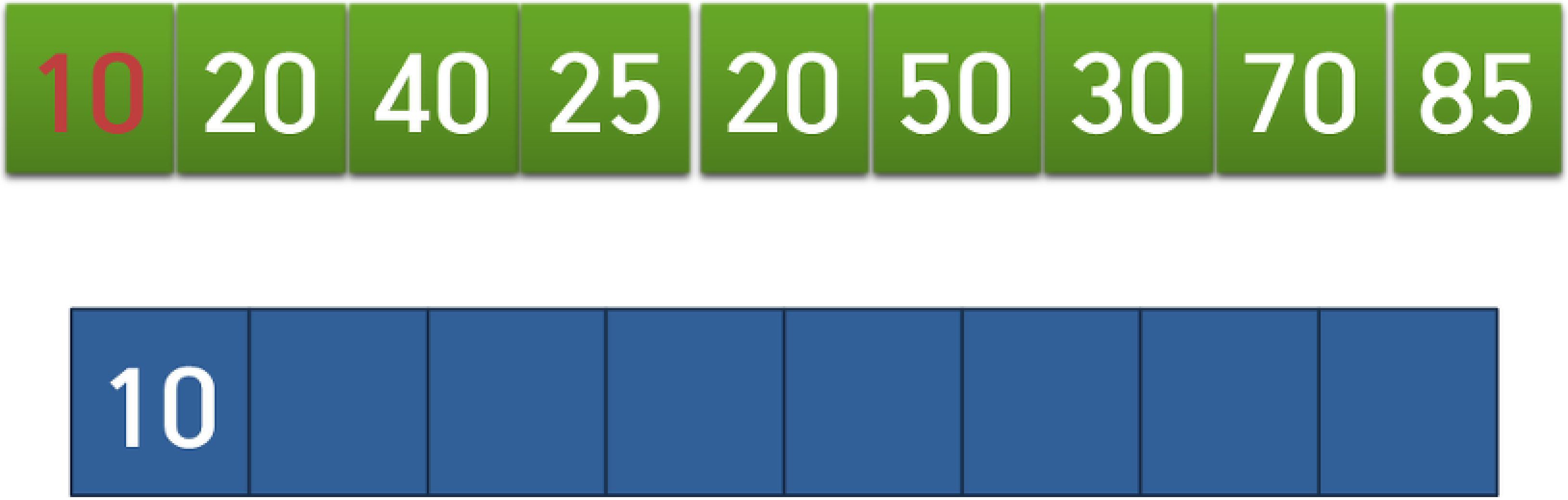
Longest Increasing Subsequence



10 20 40 25 20 50 30 70 85

10 20 40 25 20 50 30 70 85

LIS



10	20	40	25	20	50	30	70	85

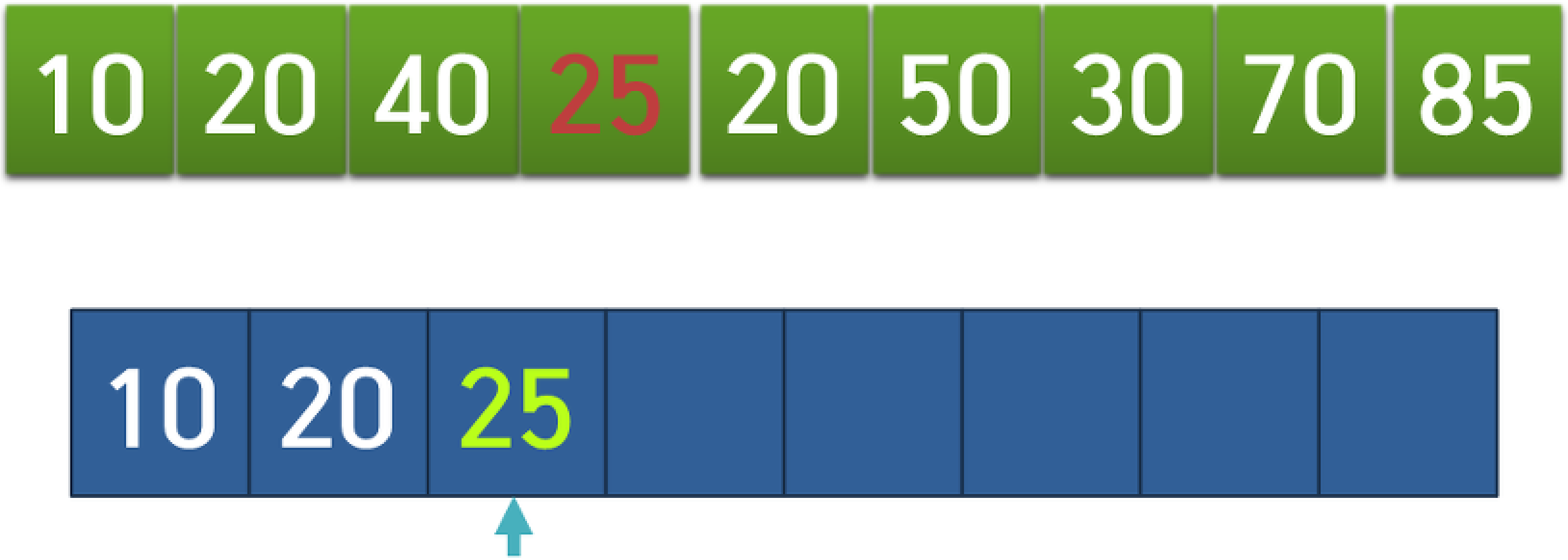
10							

LIS

10	20	40	25	20	50	30	70	85

10	20	40					

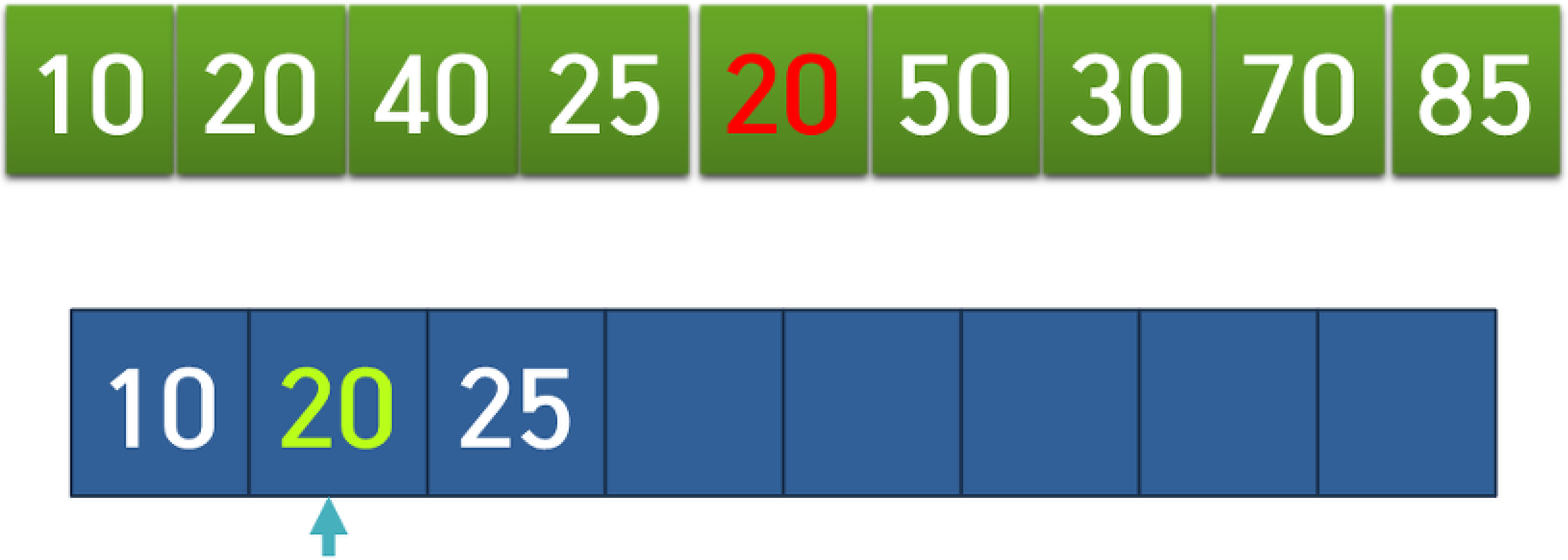
LIS



10	20	40	25	20	50	30	70	85

10	20	25					

LIS



10	20	40	25	20	50	30	70	85

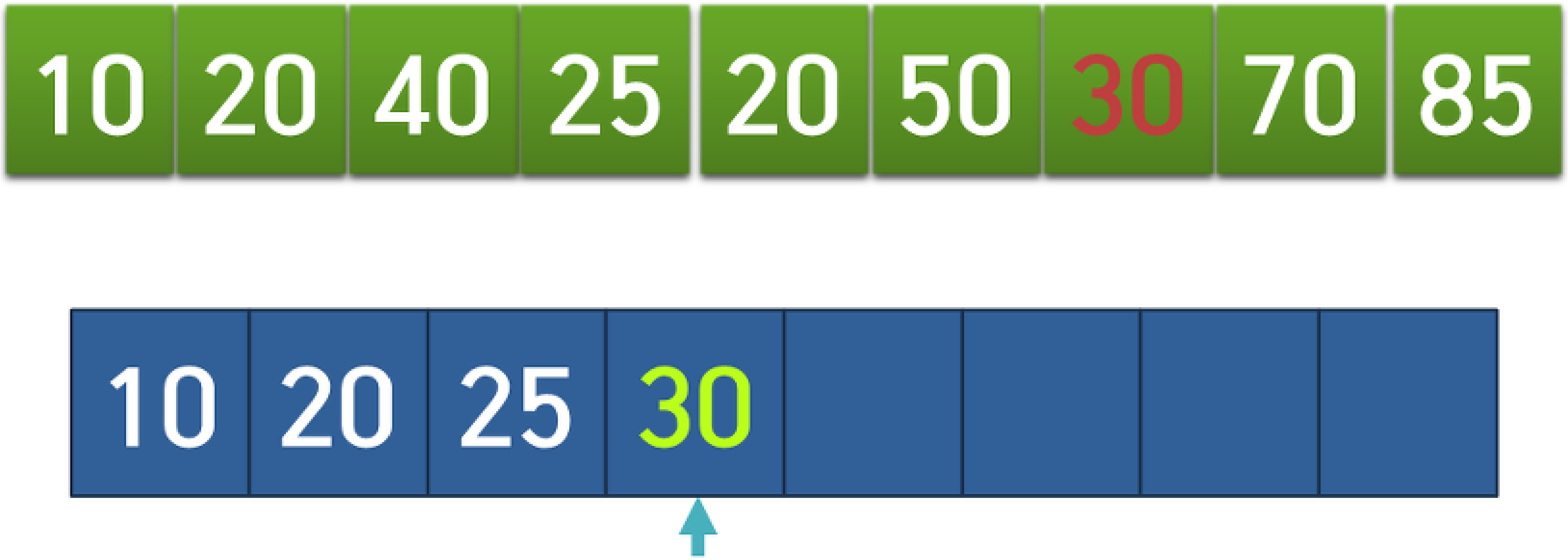
10	20	25					

LIS

10	20	40	25	20	50	30	70	85

10	20	25	50				

LIS



10	20	40	25	20	50	30	70	85

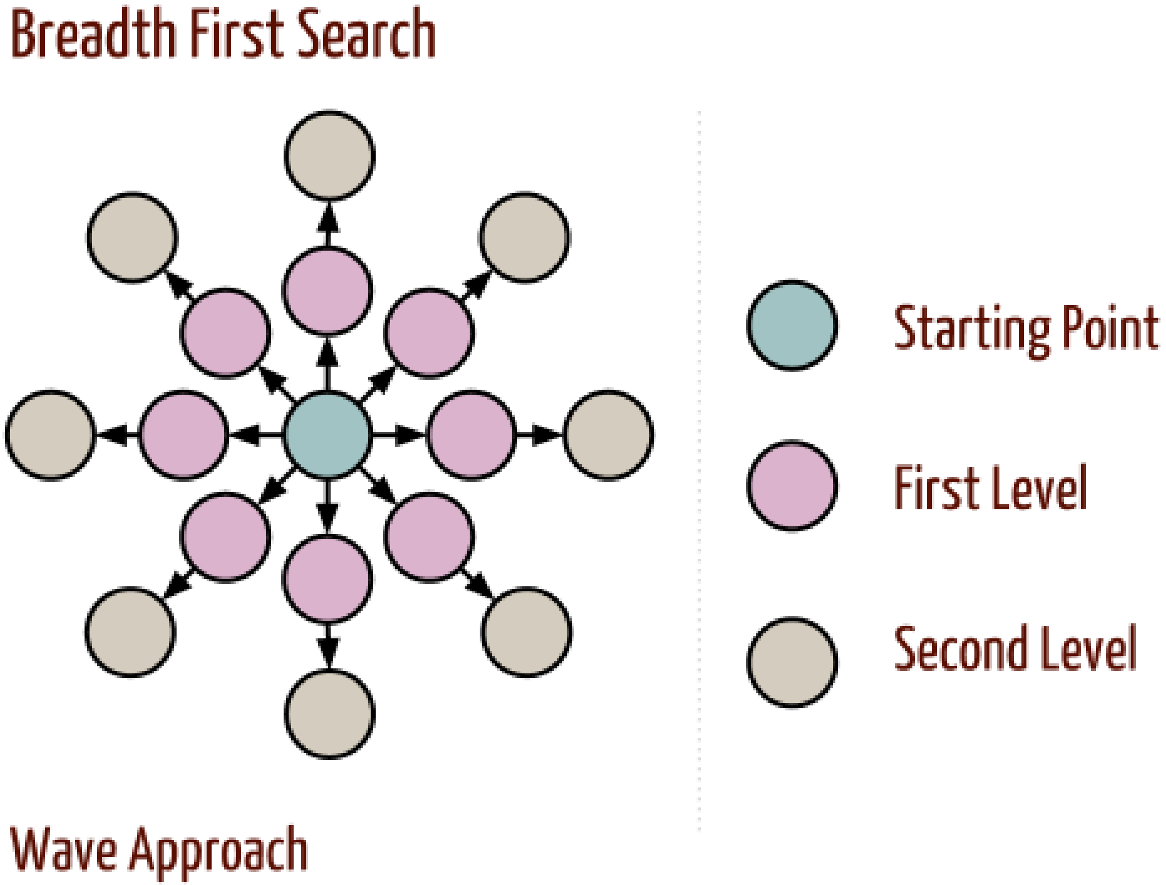
10	20	25	30				

LIS

10	20	40	25	20	50	30	70	85

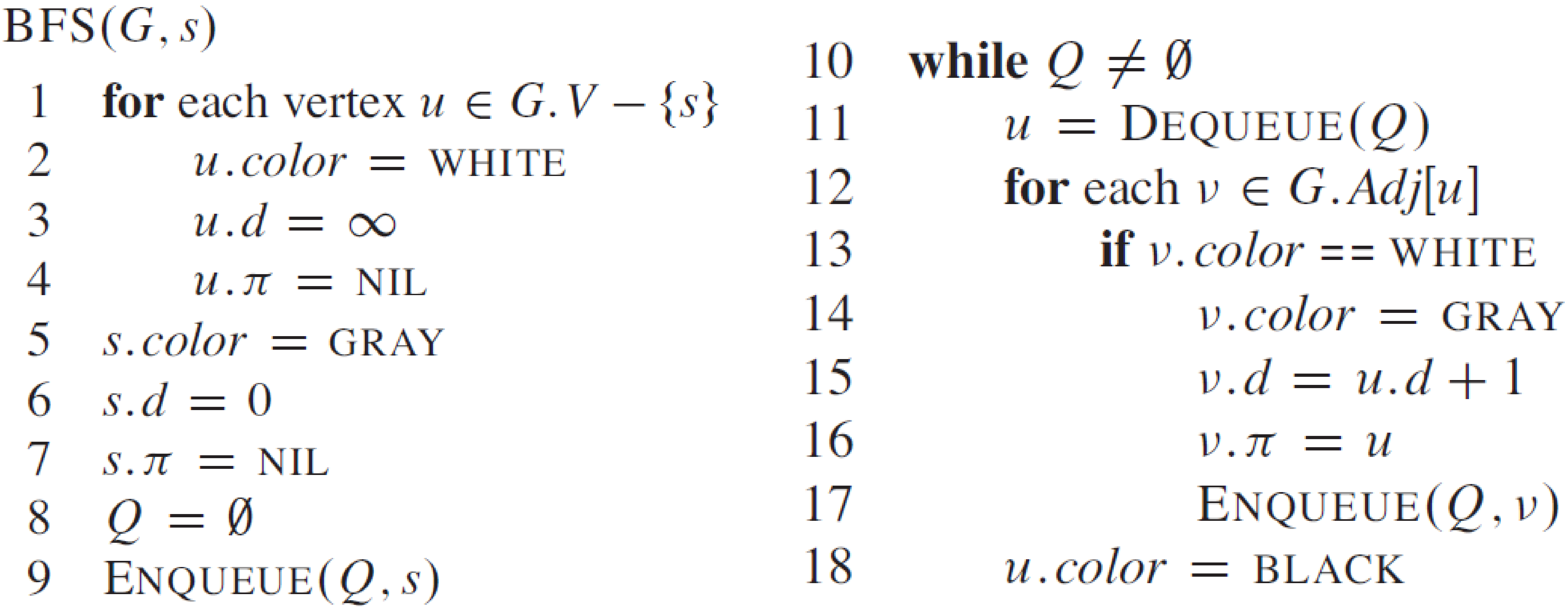
10	20	25	30	70	85		

Breadth First Search



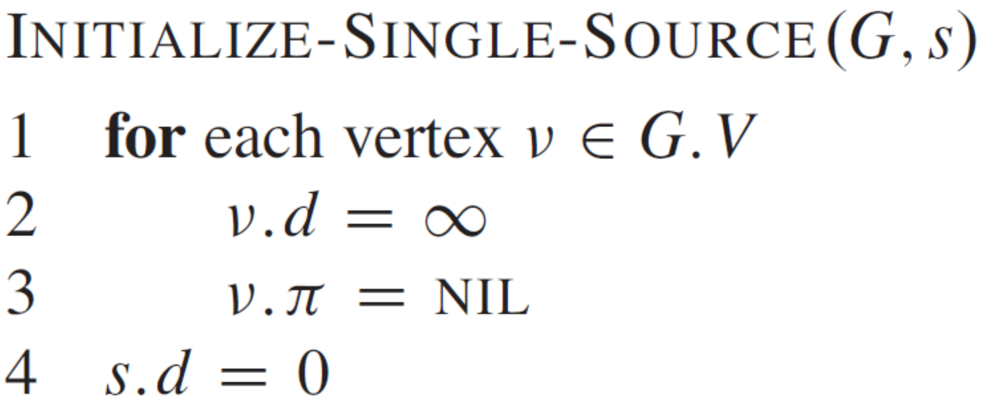
BFS

visited 배열을 color로 표현함 (white = 탐색하지 않음, gray = 연결된 vertex를 탐색해야 함, black= 탐색 완료)

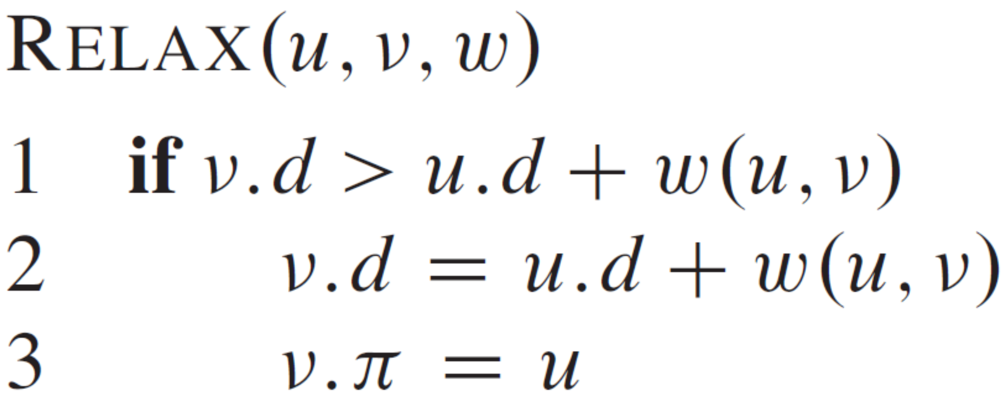


```
BFS($G, s$)
for each vertex $u \in G.V - \{s\}$
    $u.color = \text{WHITE}$
    $u.d = \infty$
    $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
ENQUEUE($Q, s$)
while $Q \neq \emptyset$
    $u = \text{DEQUEUE}(Q)$
    for each $v \in G.Adj[u]$
        if $v.color == \text{WHITE}$
            $v.color = \text{GRAY}$
            $v.d = u.d + 1$
            $v.\pi = u$
            ENQUEUE($Q, v$)
    $u.color = \text{BLACK}$
```

Dijkstra’s algorithm

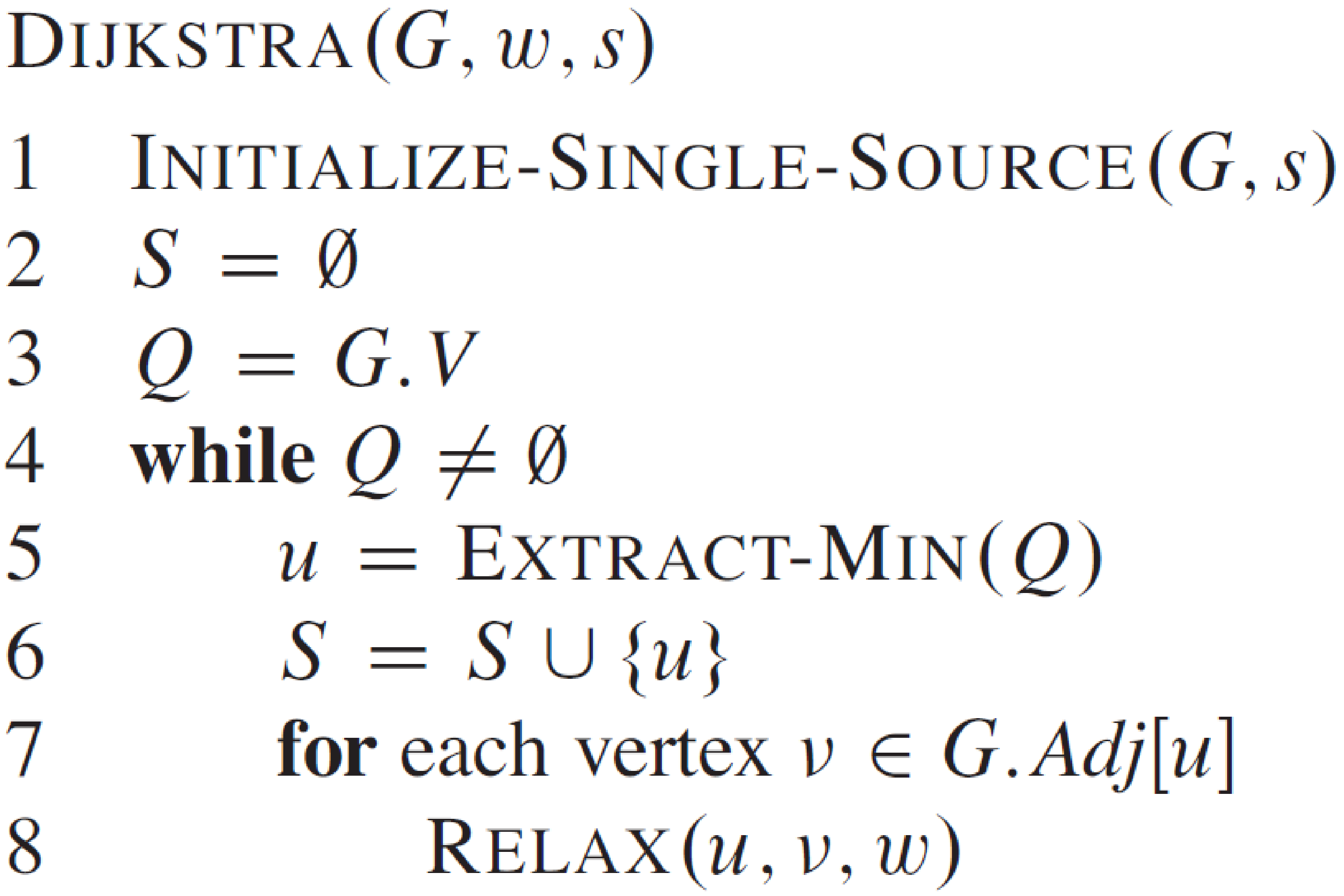


```
INITIALIZE-SINGLE-SOURCE$(G, s)$
for each vertex $v \in G.V$
    $v.d = \infty$
    $v.\pi = \text{NIL}$
$s.d = 0$
```



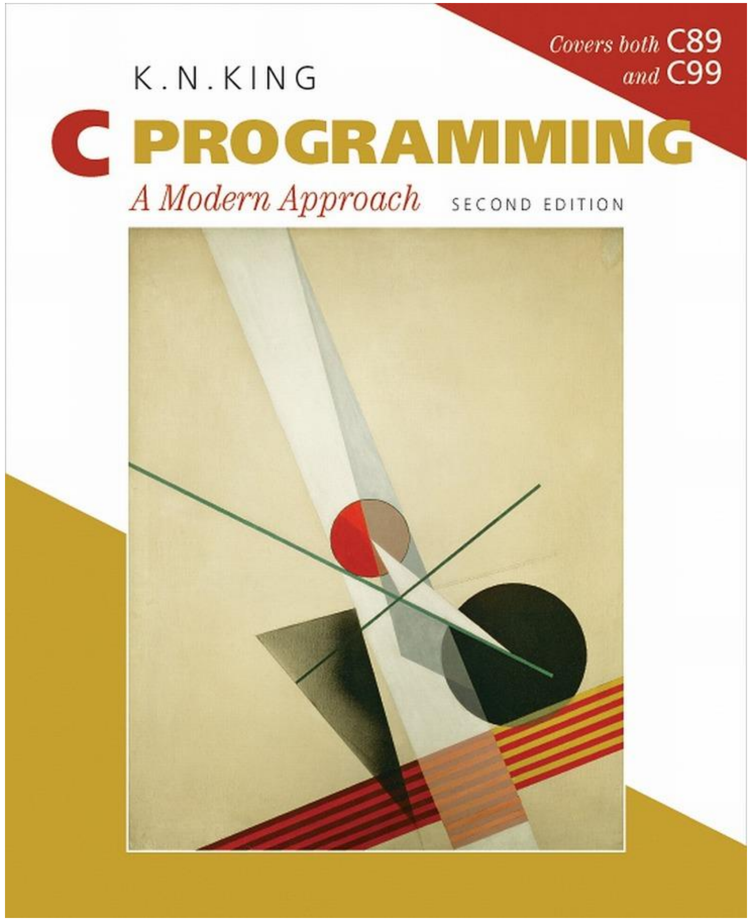
```
RELAX$(u, v, w)$
if $v.d > u.d + w(u, v)$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$
```

Dijkstra’s algorithm



```
DIJKSTRA$(G, w, s)$
INITIALIZE-SINGLE-SOURCE$(G, s)$
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
    $u =$ EXTRACT-MIN$(Q)$
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
        RELAX$(u, v, w)$
```

참고 서적

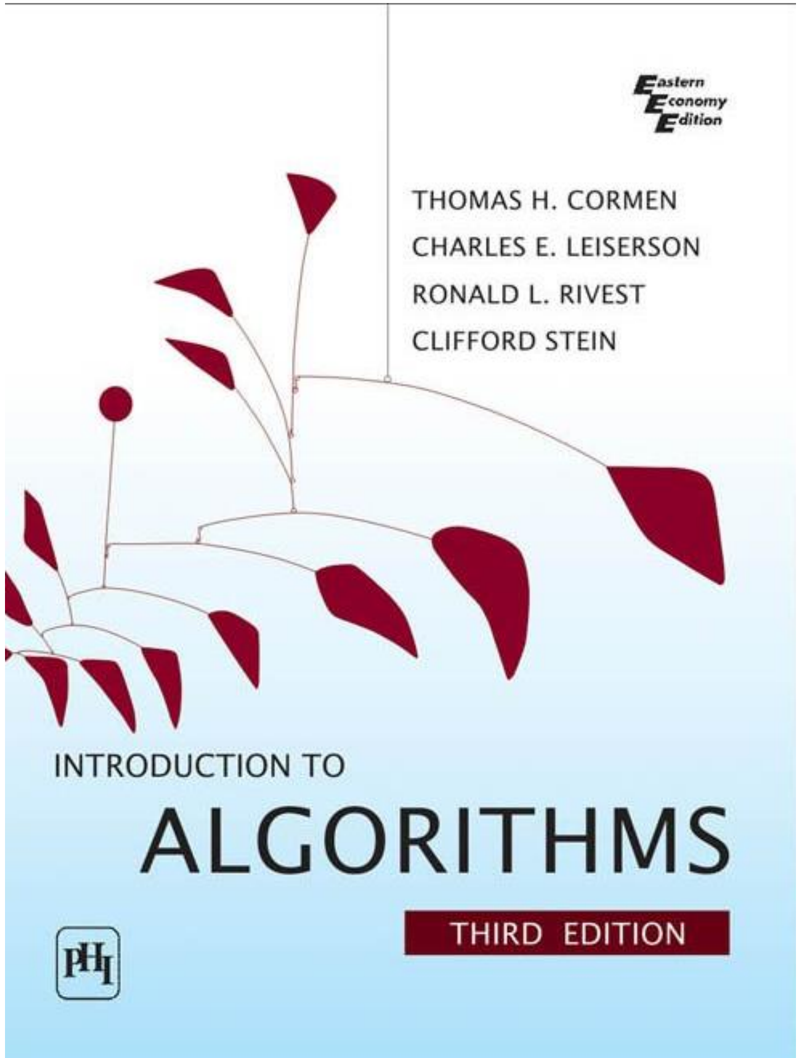


K. N. King

C Programming: A Modern Approach

2nd edition

참고 서적



Thomas H. Cormen
Introduction to Algorithms
3rd edition