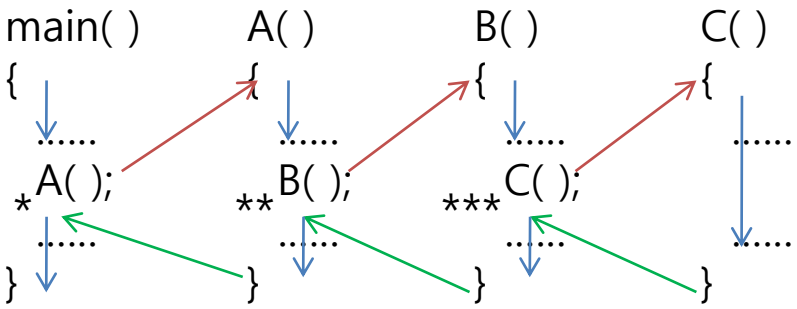
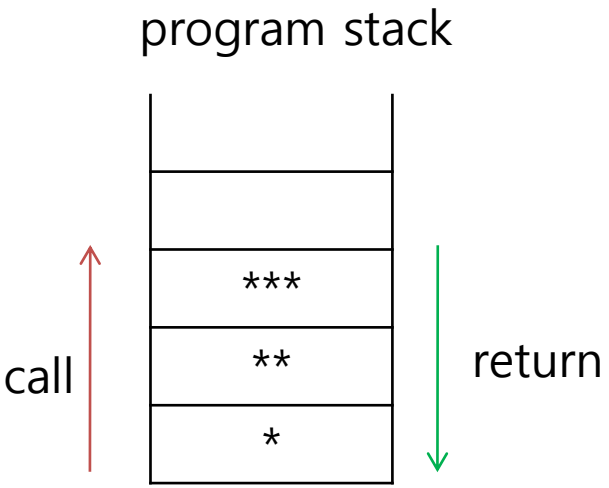
Merge Sort

Recursive Function

- 함수의 호출 (Call) – 복귀 (Return) 동작
  - Program Stack
    - 함수 호출 (call) 시 리턴 (return) 할 위치를 저장하기 위한 메모리
    - 후입선출 (LIFO) 동작





call

return

*, **, *** : 프로그램 주소

Recursive Function

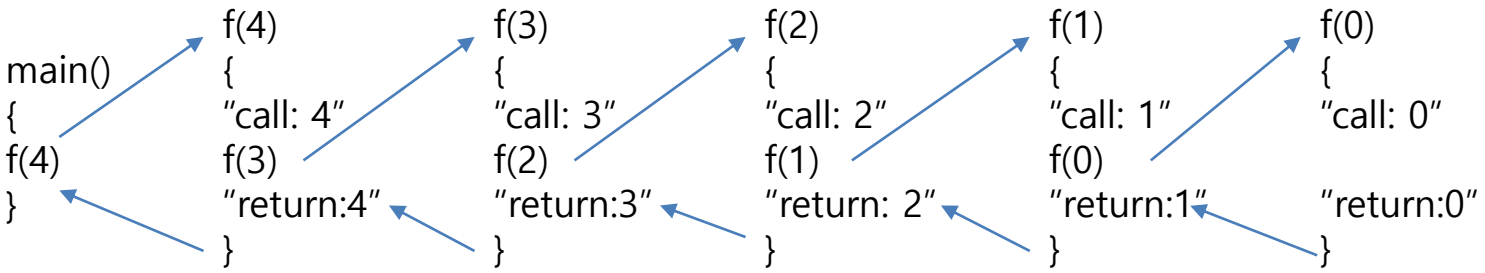
- Recursive function (재귀함수)
  - 자기 자신을 호출하는 함수
  - Program stack 에 의한 call – return
  - 복잡한 알고리즘을 쉽게 구현할 수 있다
  - 동작의 이해 및 디버깅이 어렵다

(Ex) 다음 프로그램의 출력은 ?

```
int main()
{
    f(4);
    return 0;
}
void f(int n)
{
    cout << "call: " << n << '₩n';
    if(n>0)
        f(n-1);
    cout << "return: " << n << '₩n';
}
```

Recursive Function

```
int main()
{
    f(4);
    return 0;
}
void f(int n)
{
    cout << "call: " << n << '₩n';
    if(n>0)
        f(n-1);
    cout << "return: " << n << '₩n';
}
```



Merge

- Problem

  두 개의 정렬된 숫자 열을 합쳐서 정렬

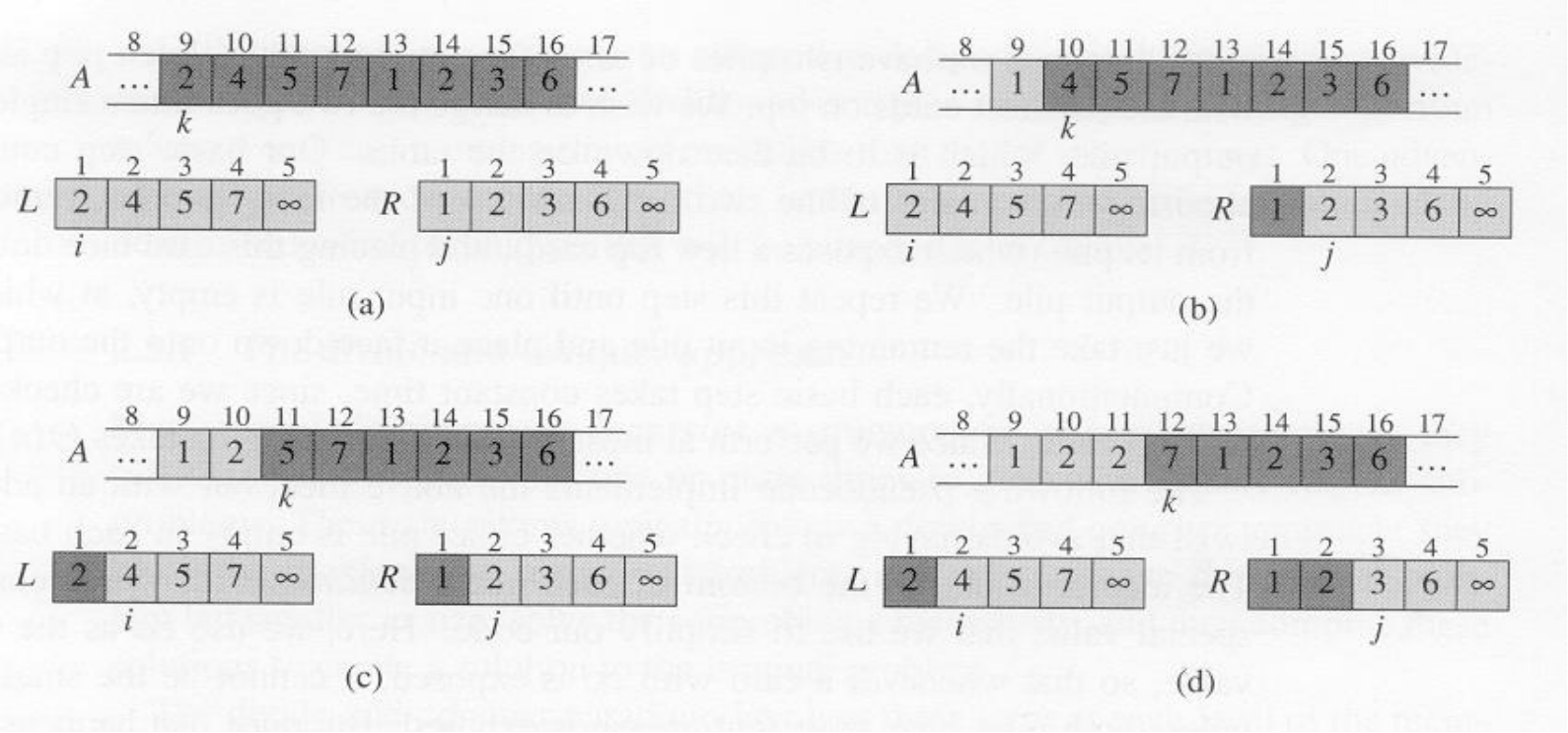
  (예) L = <2, 4 ,5, 7> R = <1, 2, 3, 6> => A = <1, 2, 2, 3, 4, 5, 6, 7>

- Idea

  L 과 R 의 맨 왼쪽 값 비교, 작은 값을 뽑아서 A 로 이동

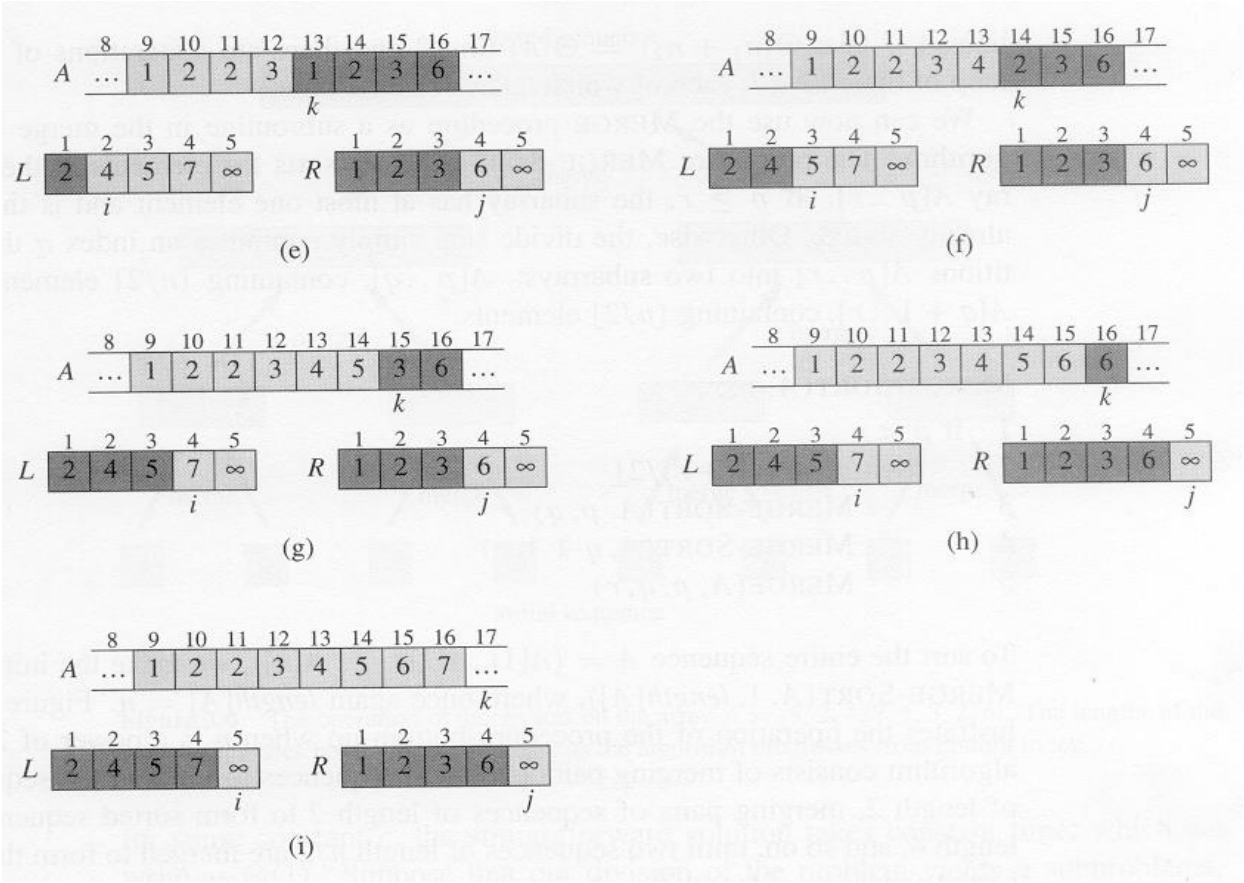
Merge

- Operation (1)



Merge

- Operation (2)



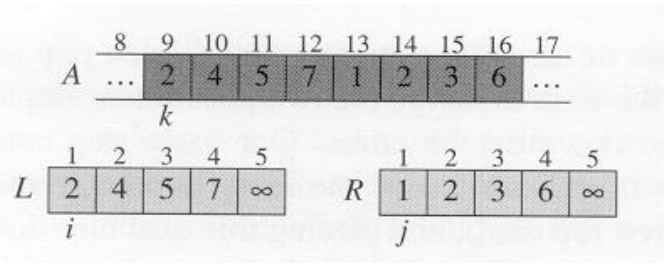
sentinel card: 각 배열 의 끝을 표시하기 위하여
매우 큰 숫자 (∞) 삽입

Merge

- Algorithm

  MERGE(A, p, q, r)
  - A: 입력 배열
  - p,q,r: 배열 주소 (p ≤ q < r)
    - A[p .. q], A[q+1 .. r] 은 각각 오름차순으로 정렬되어 있음.

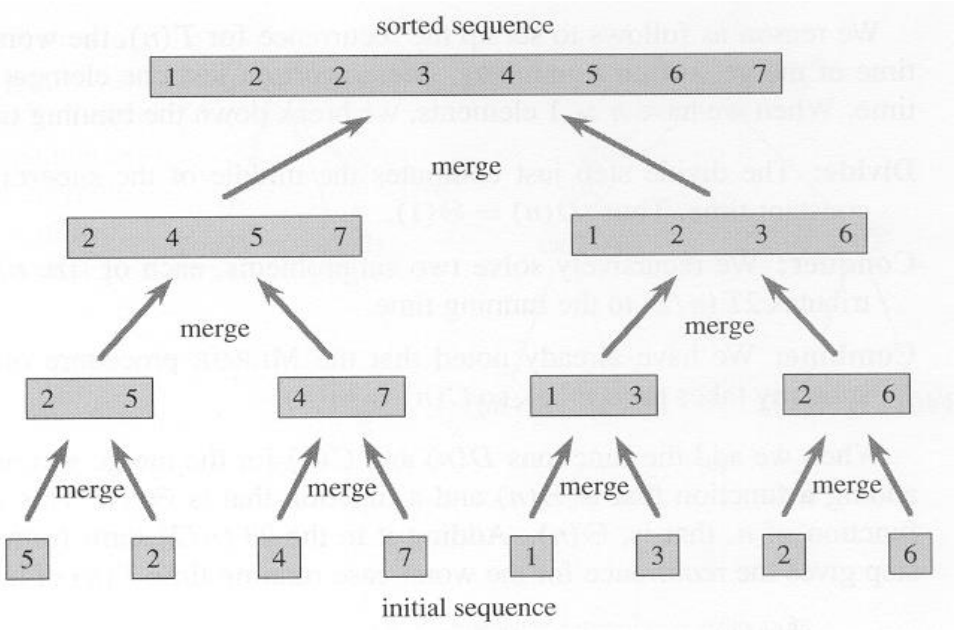
```
MERGE(A, p, q, r)
 1  n1 ← q − p + 1
 2  n2 ← r − q
 3  create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
 4  for i ← 1 to n1
 5        do L[i] ← A[p + i − 1]
 6  for j ← 1 to n2
 7        do R[j] ← A[q + j]
 8  L[n1 + 1] ← ∞
 9  R[n2 + 1] ← ∞
10  i ← 1
11  j ← 1
12  for k ← p to r
13        do if L[i] ≤ R[j]
14              then A[k] ← L[i]
15                   i ← i + 1
16              else A[k] ← R[j]
17                   j ← j + 1
```



running time : Θ(n) (n = r-p+1)

Merge Sort

- Problem
  - input: sequence of n numbers $<a_1, a_2, \ldots, a_n>$
  - output: $<a_1', a_2', \ldots, an'>$ such that $a_1' \le a_2' \le \ldots \le a_n'$
- Idea
  - Input sequence 를 n 개로 분할 : Divide
  - 분할된 sequence 에 대하여 Merge algorithm 반복수행 : Conquer



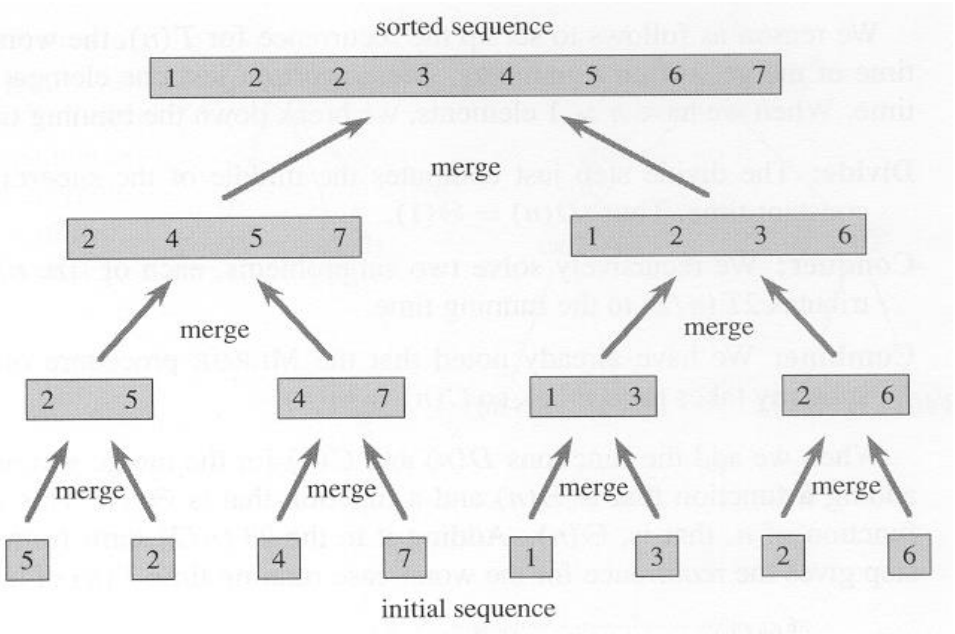

```
MERGE-SORT(A,1,8)
{
   MERGE(A,1,1,2)
   MERGE(A,3,3,4)
   MERGE(A,1,2,4)

   MERGE(A,5,5,6)
   MERGE(A,7,7,8)
   MERGE(A,5,6,8)

   MERGE(A,1,4,8)
}
```

Merge Sort

- Algorithm




```
MERGE-SORT(A, p, r)
1 if p < r
2    then q ← ⌊(p+r) / 2⌋
3         MERGE-SORT(A, p, q)
4         MERGE-SORT(A, q+1, r)
5         MERGE(A, p, q, r)
```

- p, r: 정렬하고자 하는 배열 A 의 처음 및 마지막 번지
- floor 함수 ⌊x⌋ : x 보다 같거나 작은 최대의 정수
  ceiling 함수 ⌈x⌉: x 보다 같거나 큰 최소의 정수

Merge Sort

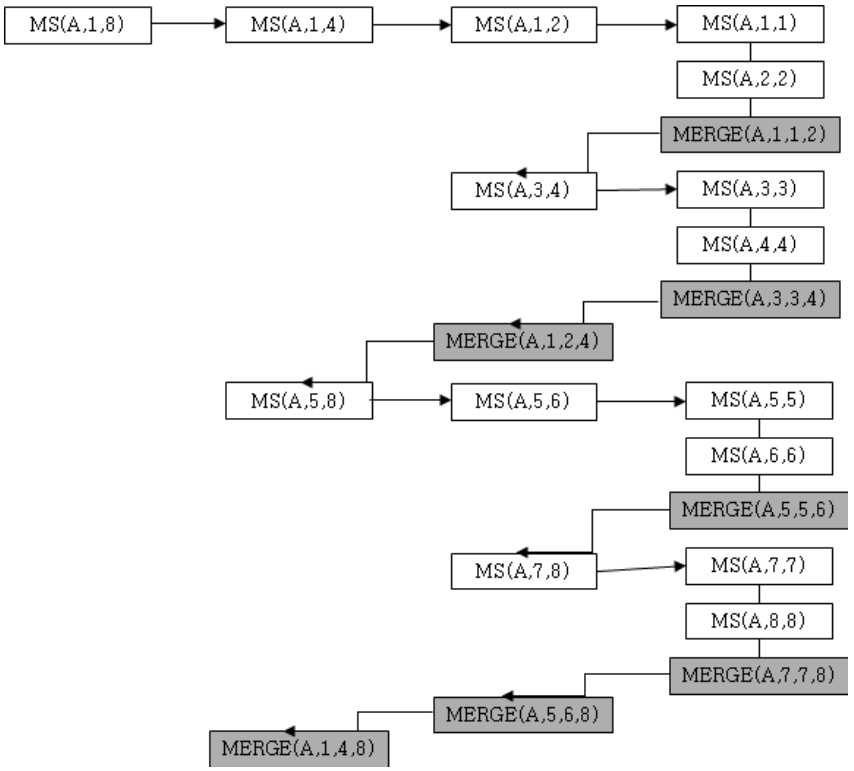
- Algorithm

```
MERGE-SORT(A, p, r)
1 if p < r
2    then q ← ⌊(p+r) / 2⌋
3         MERGE-SORT(A, p, q)
4         MERGE-SORT(A, q+1, r)
5         MERGE(A, p, q, r)
```

- Operation

  A = <5,2,4,7,1,3,2,6>

  p=1, r=8 경우



Merge Sort

(Q) A = <3, 5, 9, 4, 8, 7, 1, 2>

MERGE-SORT(A,1,8) 수행 시 A 의 변화 과정 ?

#	3	5	9	4	8	7	1	2
1								
2								
3								
4								
5								
6								
7								

Analysis

- Running Time

$$T(n) = \begin{cases} c & (n = 1) \\ 2T(n/2) + cn & (n > 1) \end{cases}$$

```
MERGE-SORT(A, p, r)
1 if p < r
2    then q ← ⌊(p+r) / 2⌋
3         MERGE-SORT(A, p, q)
4         MERGE-SORT(A, q+1, r)
5         MERGE(A, p, q, r)
```

For n>1 $T(n) = 2T(\frac{n}{2}) + cn$

$$= 2(2T(\frac{n}{4}) + c \cdot \frac{n}{2}) + cn = 4T(\frac{n}{4}) + 2cn$$

$$= 4 \cdot (2 \cdot T(\frac{n}{8}) + c \cdot \frac{n}{4}) + 2cn = 8T(\frac{n}{8}) + 3cn$$

$$\vdots$$

$$= 2^k T(\frac{n}{2^k}) + kcn$$

Let $n = 2^k \quad (k = \lg n)$

$$T(n) = nT(1) + cn \lg n$$

$$= cn + cn \lg n$$

$$= \Theta(n \lg n)$$

Analysis

- Memory
  - Not in-place sorting
  - 입력 배열 (A) 이외의 별도 메모리 사용: L, R
  - Size = *n*

Divide-and-Conquer

- Divide-and-conquer approach (분할 정복)
  - Divide: 원 문제를 몇 개의 소 문제로 분할
    (ex) n 원소 정렬 문제를 2개의 n/2 원소 정렬문제로 분할
  - Conquer: 각 소 문제를 재귀적으로 해결
    (ex) 2개의 n/2 원소 정렬문제를 각각 recursive 알고리즘으로 해결
  - Combine: 각 소 문제의 해로부터 원 문제의 해를 구함
    (ex) 2개의 정렬된 원소를 Merge
  - ❖ Recursive structure

(cf) Incremental approach

Example

- Binary search algorithm
  1) Problem

     input : A = $<a_1, a_2, \ldots, a_n>$ and a value v ($a_1 \le a_2 \le \ldots \le a_n$)

     output : an index i such that $v = A[i]$ or

     NIL if v does not appear in A

  2) Operation
     - 중간 값을 탐색하여, v 보다 작으면 오른쪽으로, 크면 왼쪽으로 진행
     - divide-and-conquer approach

Example

- Binary search algorithm
  3) Algorithm

  4) Analysis
     - Running time