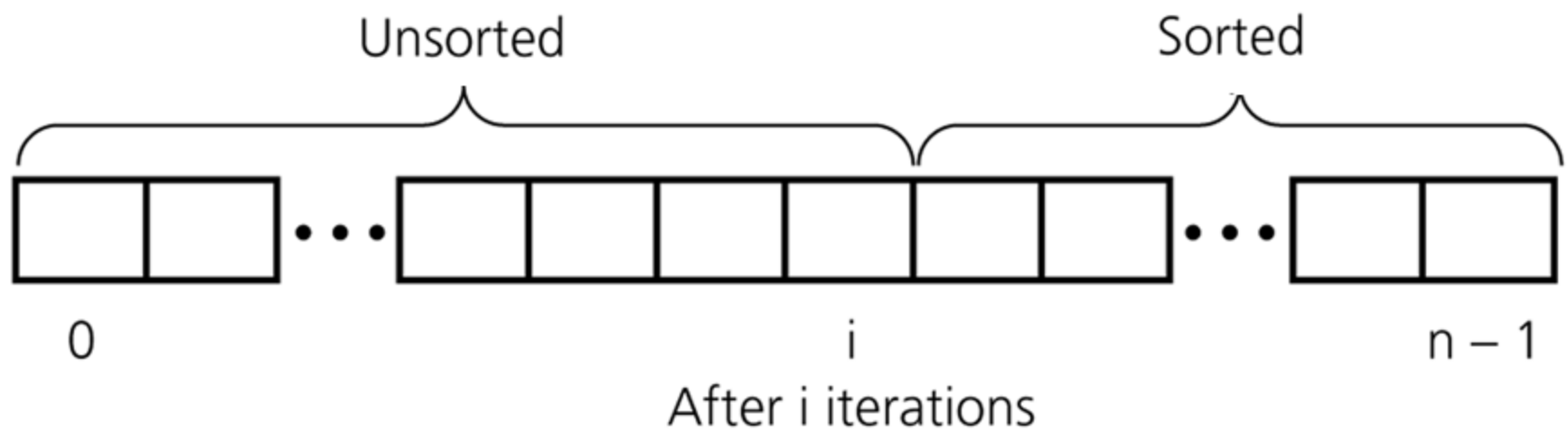
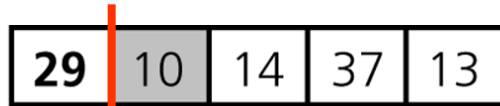


Insertion Sort(삽입 정렬)

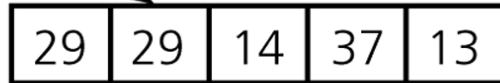


# Insertion Sort

Initial array:



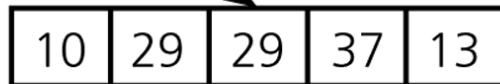
Copy 10



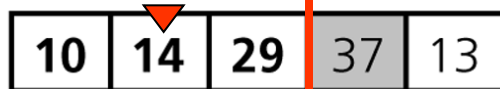
Shift 29



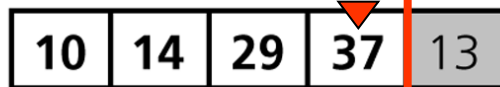
Insert 10; copy 14



Shift 29



Insert 14; copy 37, insert 37 on top of itself

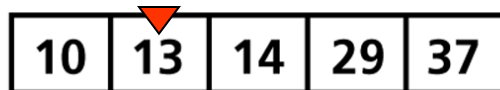


Copy 13



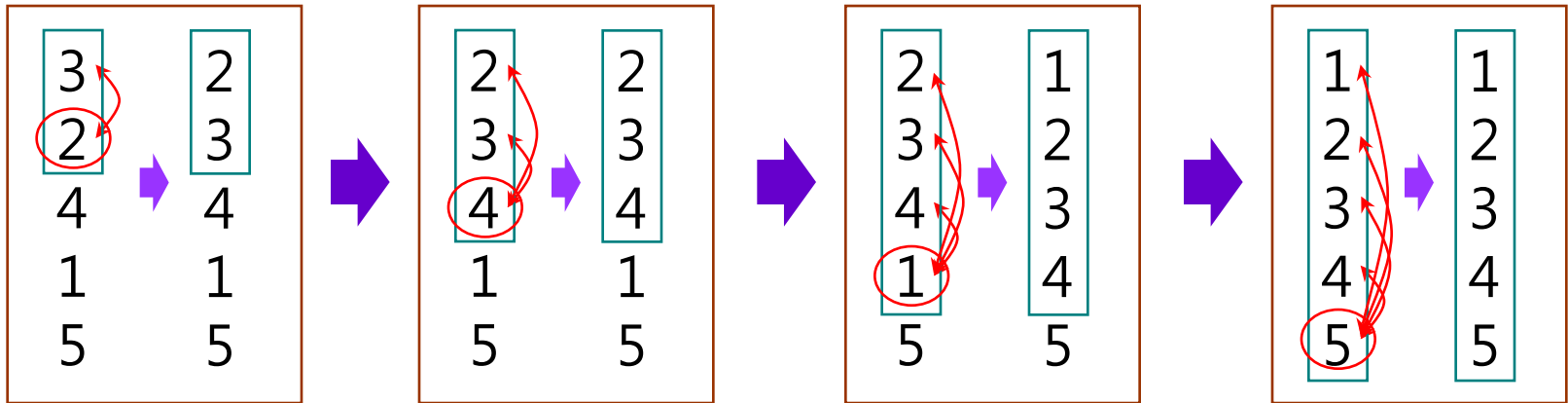
Shift 37, 29, 14

Sorted array:



Insert 13

# Insertion Sort



insertionSort(int\* A, n)    ▷ 배열 A을 정렬한다

{

    int i;

**for** (i = 1 ; i < n ; i++ )

        A[0]...A[i-1]의 적당한 자리에 A[i]를 삽입한다;

}

insertionSort(int\* A, n)    ▷ 배열 A를 정렬한다

{

    int i;

**for** ( i = 1 ; i < n ; i++ )

        A[0]...A[i-1]의 적당한 자리에 A[i]를 삽입한다;

}

**A[i]가 들어갈 적당한 자리부터...**

        ( j = 0 ; j < i ; j++ )

        if( A[j] > A[i] ) break;

        // loop가 끝나면 A[j] 위치에 A[i]가

## A[i]가 들어갈 적당한 자리부터...

```
for (j = 0; j < i; j++)  
    if( A[j] > A[i] ) break;
```

// loop가 끝나면 A[j] 위치에 A[i]가 들어가야 함을 알 수 있음

1 2 4 5 3 7

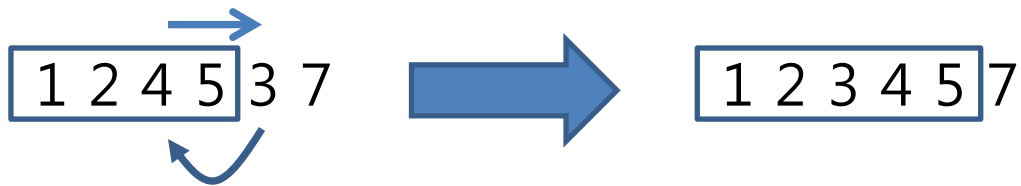
i=4, A[i] = 3; 인 상황

```
for(j=0 ; j < 4 ; j++ ) // 비교를 최대 A[3]까지 함.  
    if ( A[j] > A[4] ) break;
```

이 부분을 수행하면


j ← 2 이고 A[4]는 A[2]에 들어가야 한다는 뜻  
A[2], A[3]은 뒤로 이동하고

.




$i=4$ ,  $A[i] = 3$ ; 인 상황.,  $j = 2$

먼저  $A[4]$ 를 적당한 변수  $temp$ 에 저장;  $temp = A[i]$ ;


$A[3] \rightarrow A[4]$   1 2 4 5 5 7


$A[2] \rightarrow A[3]$   1 2 4 4 5 7


의 순서로 뒤로 밀어야 함.

그리고  $A[2]$ 에  $m$ 을 대입;  $A[2] = temp$ ;  1 2 3 4 5 7

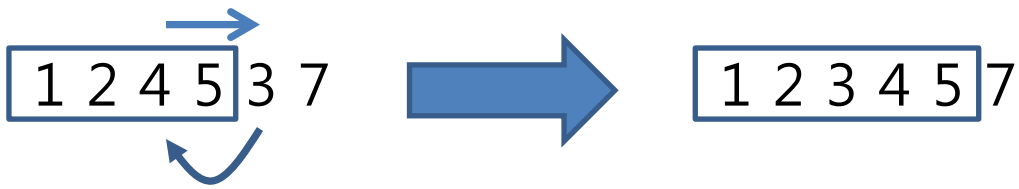
만약 순서를 이상하게 하면 이상한 일이 발생.

$A[2] \rightarrow A[3]$   1 2 4 4 3 7

$A[3] \rightarrow A[4]$   1 2 4 4 4 7







$i=4$ ,  $A[i] = 3$ ; 인 상황.,  $j = 2$

먼저  $A[4]$ 를 적당한 변수  $temp$ 에 저장;  $temp = A[i];$

$A[3] \rightarrow A[4]$  → 1 2 4 5 5 7

$A[2] \rightarrow A[3]$  → 1 2 4 4 5 7

//오른쪽으로 옮기기

$A[j] = temp;$

insertionSort(int\* A, n)      ▷ 배열 A를 정렬한다

```
{  
    int i;  
    for (int i = 1 ; i < n ; i++)  
        A[0]...A[i-1]의 적당한 자리에 A[i]를 삽입한다;  
}
```

insertionSort(int\* A, n) //배열 A를 정렬한다

```
{  
    int i;  
    int j, k, temp;  
  
    for (i = 1 ; i < n ; i++)  
    {  
        for (j = 0; j < i; j++)  
            if (A[j] > A[i]) break;  
  
        temp = A[i];  
        for (k = i; k > j; k--)  
            A[k] = A[k-1];  
        A[j] = temp;  
    }  
}
```

# Inductive Verification of Insertion Sort

- 배열  $A[0]$ 만 놓고 보면
    - 정렬되어 있음
  - 배열  $A[0 \dots k]$ 까지 정렬되어 있다면
    - ② 행의 삽입에 의해  $A[0 \dots k+1]$ 까지 정렬된다
- ✓ 고등학교에서 배운 수학적 귀납법과 다를 바 없음

Merge Sort(합병 정렬)

# 각개격파

## ■ Divide & Conquer

- 알고리즘 구현 패턴
  - 큰 문제를 두 개 혹은 그 상의 부분 문제로 분할하여
  - 부분 문제들을 같은 함수/프로시저/알고리즘으로 풀어내고
  - 부분 문제들의 솔루션으로 큰 문제의 솔루션을 도출하는 패턴
- 
- 작은 문제들로 분할할 때 일반적으로 분할된 범위가 겹치지 않음.

Examples)

```
binary_search(int *A, int l, int r)  
→ Call binary_search(A, l, m-1) or  
binary_search(A, m+1, r)
```

```
max_subarray(int *A, int l, int r)  
→ Call max_subarray(A, l, m) and  
max_subarray(A, m, r) and some additional computation
```



■ 각개격파로 정렬을 할 수 있을까?

- 만약 각개격파가 가능하다면...

l		m						r	
31	3	65	73	8	11	20	29	48	15



Recursive calls.  
Easy tasks to us.

3	8	31	65	73	11	15	20	29	48
---	---	----	----	----	----	----	----	----	----



합치자. (merge)

3	8	11	15	20	29	31	48	65	73
---	---	----	----	----	----	----	----	----	----

# Mergesort

mergeSort(A[ ], p, r) // A[p ... r]을 정렬한다

{

  if (p < r) then {

    q ← (p+r)/2;

// ① ▷ p, r의 중간 지점 계산

    mergeSort(A, p, q);

// ② ▷ 전반부 정렬

    mergeSort(A, q+1, r);

// ③ ▷ 후반부 정렬

    merge(A, p, q, r);

// ④ ▷ 병합

  }

}

merge(A[ ], p, q, r)

{

  정렬되어 있는 두 배열 A[p ... q]와 A[q+1 ... r]을 합하여  
  정렬된 하나의 배열 A[p ... r]을 만든다.

}

이것이 Recursion!



## mergeSort의 작동 예

정렬할 배열이 주어짐

31	3	65	73	8	11	20	29	48	15
----	---	----	----	---	----	----	----	----	----

배열을 반반으로 나눈다

31	3	65	73	8	11	20	29	48	15
----	---	----	----	---	----	----	----	----	----

— (1)

각각 독립적으로 정렬한다

3	8	31	65	73	11	15	20	29	48
---	---	----	----	----	----	----	----	----	----

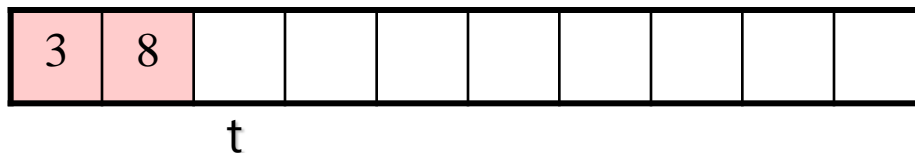
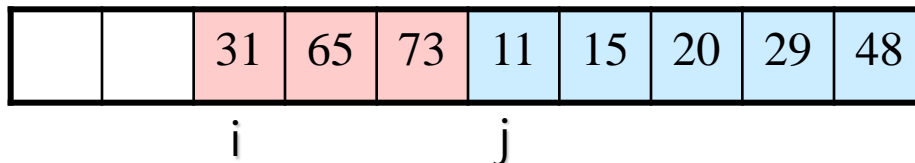
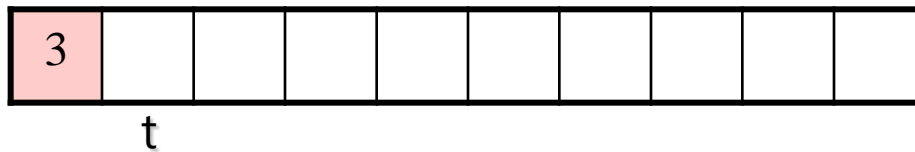
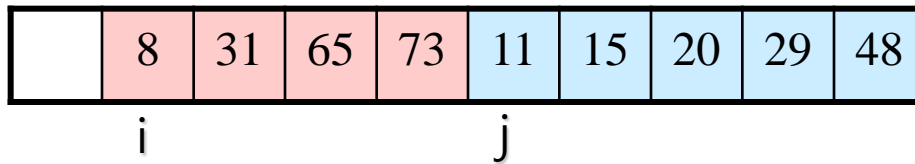
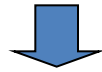
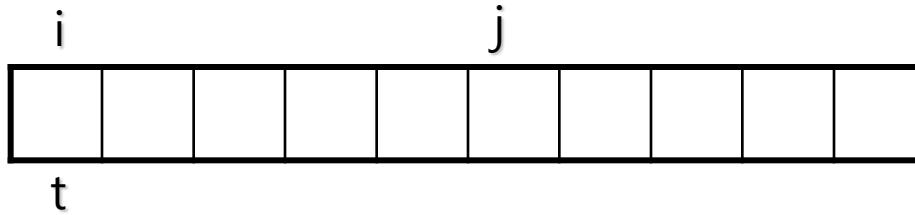
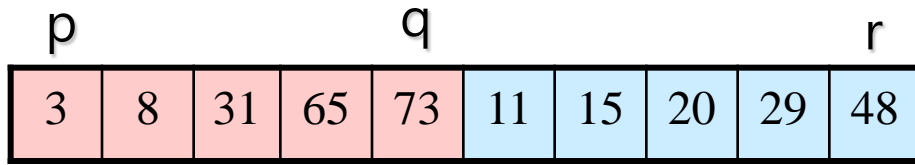
— (2) (3)

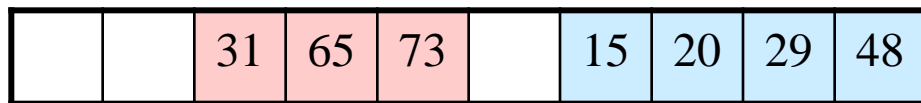
병합한다 (정렬완료)

3	8	11	15	20	29	31	48	65	73
---	---	----	----	----	----	----	----	----	----

— (4)

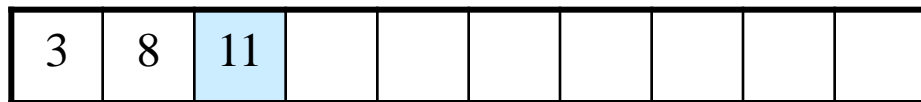
## merge의 작동 예



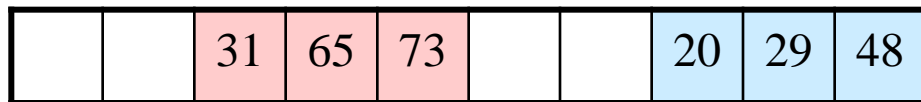


i

j



t

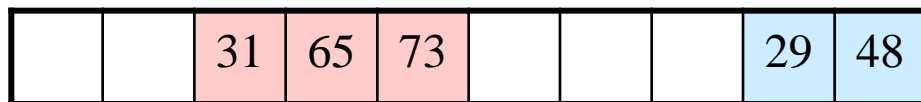


i

j

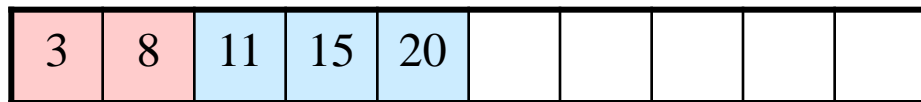


t



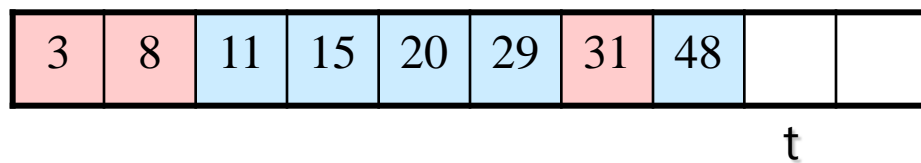
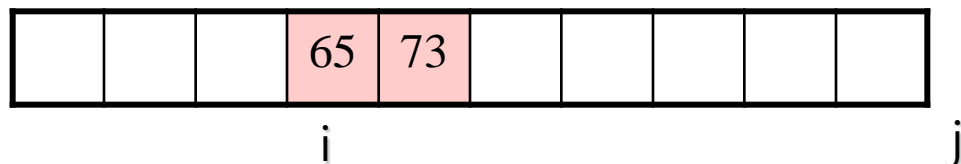
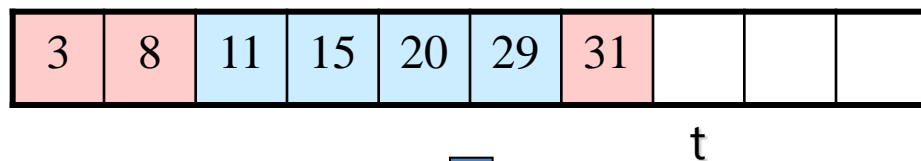
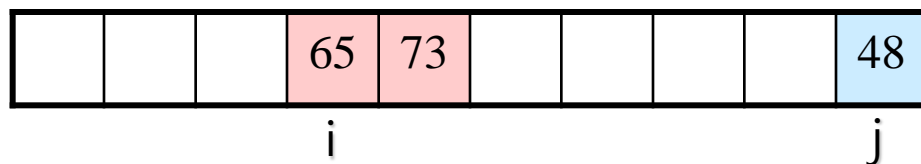
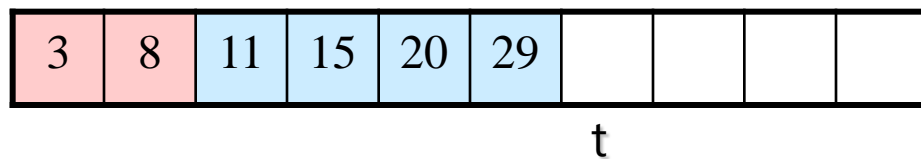
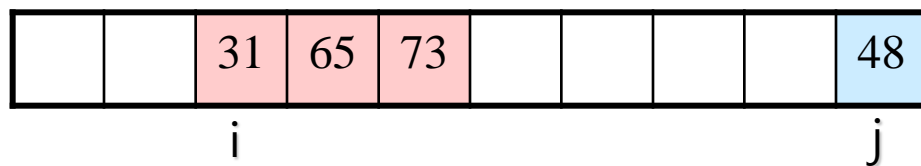
i

j



t





--	--	--	--	--	--	--	--	--	--

i

j

3	8	11	15	20	29	31	48	65	73
---	---	----	----	----	----	----	----	----	----

t

# Mergesort

mergeSort(A[ ], p, r) // A[p ... r]을 정렬한다

```
{  
    if (p < r) then {  
        q ← (p+r)/2;  
        mergeSort(A, p, q);  
        mergeSort(A, q+1, r);  
        merge(A, p, q, r);  
    }  
}
```

// ① ▷ p, r의 중간 지점 계산  
// ② ▷ 전반부 정렬  
// ③ ▷ 후반부 정렬  
// ④ ▷ 병합

merge(A[ ], p, q, r)

```
{  
    정렬되어 있는 두 배열 A[p ... q]와 A[q+1 ... r]을 합하여  
    정렬된 하나의 배열 A[p ... r]을 만든다.  
}
```

## How to code in C

# Animation (Mergesort)

1 2 3 4 6 7 8 9

✓ 수행시간:  $O(n \log n)$