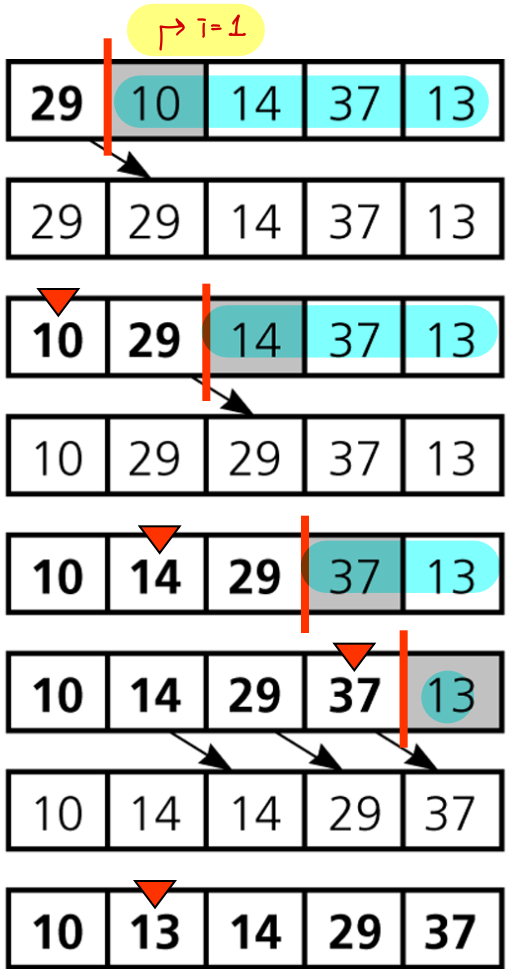
Insertion Sort(삽입 정렬)

Insertion Sort



Copy 10

Shift 29 → 그 앞을 뒤로 미룬다.

Insert 10; copy 14

↳ 원래 그 앞의 자리에 10을 넣는다.

Shift 29 → 그 앞을 뒤로 미룬다.

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

Copy 13

Shift 37, 29, 14

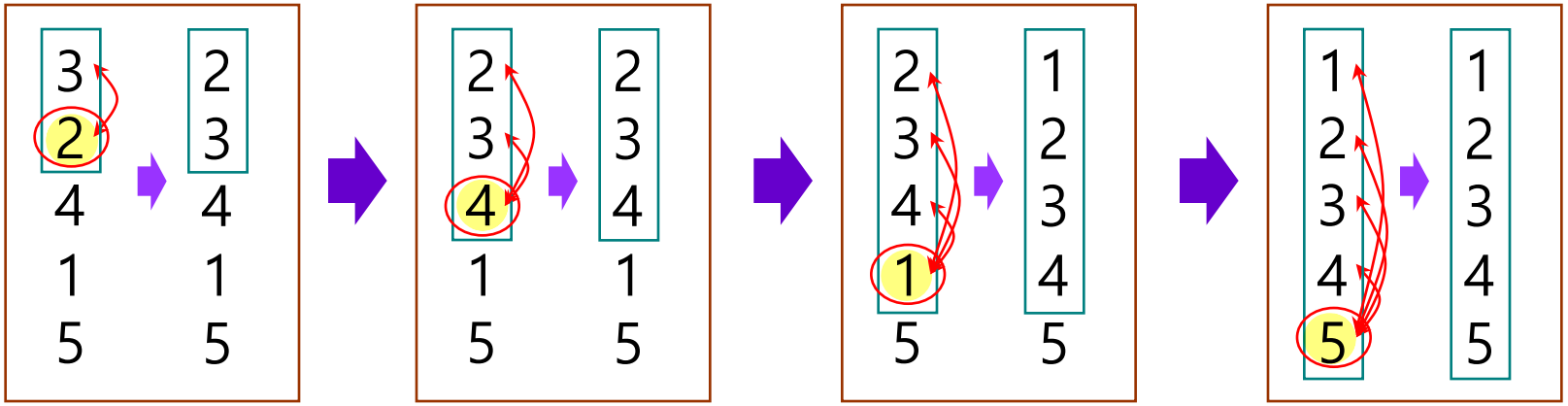
Insert 13

```
for (int i = 1; i<n; i++)
```

a[i] 의
- 위치 찾기
- 뒤로 한 칸씩

Insertion Sort

i=0 인 3은 이미 정렬되어 있음.
i=1 인 2부터 시작



a[i] 의 위치 구하기.

→ a[i] 보다 큰 수 중, 첫 번째 수의 자리에

a[i] 를 insertion 하고 나머지 수를 뒤로 한 칸씩 밀기.

a[3] = 1보다 큰 수 (2, 3, 4)

중에서 가장 작은 수(2) 의 자리에

a[3] 이 들어가야 함.

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

A[i]보다 큰 수 중 첫번째 수가 있는 자리

A[i]가 들어갈 적당한 자리부터...

```
(j = 0; j < i; j++)
```
→ 처음부터 자기 바로 앞 정렬되어 있는 배열까지

```
if( A[j] > A[i] ) break;
```
→ 자기보다 큰 수를 만나자마자 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
```

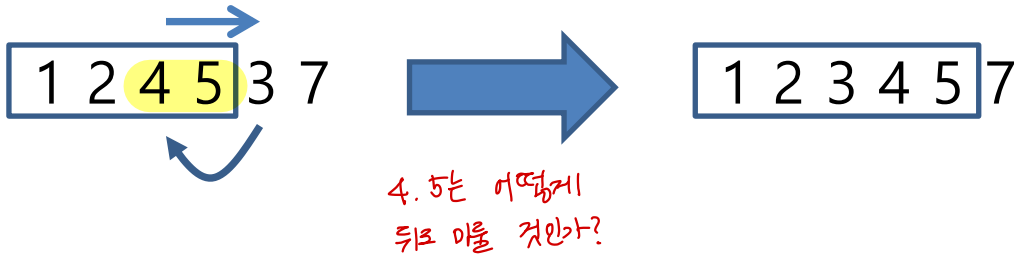
0 1 2 3 4 5

1	2	4	5	3	7

i=4, A[i] = 3; 인 상황 → outer loop의 i=4, A[4]=3

```
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 temp에 저장; `temp = A[i];`

↳ 3

A[3] → A[4] 1 2 4 5 5 7

↳ 3을 5로 바꿈.

A[2] → A[3] 1 2 4 4 5 7

↳ 5를 4로 바꿈.

temp 삽입

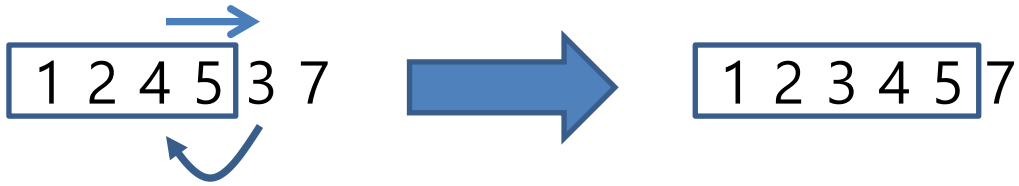
의 순서로 뒤로 밀어야 함.

그리고 A[2]에 m을 대입; `A[2] = temp;` 1 2 3 4 5 7

만약 순서를 이상하게 하면 이상한 일이 발생.

A[2] → A[3] 1 2 4 4 3 7

A[3] → A[4] 1 2 4 4 4 7



첫 번째로 만난 큰 수.
즉, 최종적으로 A[i]가 들어가야 하는 위치

i=4, A[i] = 3; 인 상황., j = 2

먼저 A[4]를 적당한 변수 temp에 저장; `temp = A[i];`

A[3] → A[4] 1 2 4 5 5 7

A[2] → A[3] 1 2 4 4 5 7

//오른쪽으로 옮기기

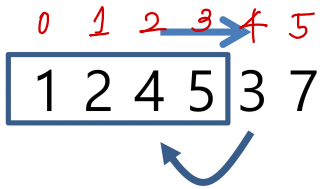
`A[j] = temp;`

최종적으로 들어갈 위치

```
insertionSort(int* A, n)        ▷ 배열 A을 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 A을 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;
                    ↳ 최종적으로 A[j]가 들어갈 위치.
          temp = A[i];
          for (k = i; k > j; k--)
               A[k] = A[k-1];
          A[j] = temp;
     }
}
```




```
for (int k=4; k>2; k--)
    A[k] = A[k-1]
→ A[4] = A[3]
  A[3] = A[2]
```

→ 최종 모양: 1 2 4 4 5

↑ 여기에 temp 값을 집어넣으면 됨.

Inductive Verification of Insertion Sort

- **배열 A[0]만 놓고 보면**
  - 정렬되어 있음
- **배열 A[0 ... k]까지 정렬되어 있다면**
  - ➔ ② 행의 삽입에 의해 A[0 ... k+1]까지 정렬된다

✓고등학교에서 배운 수학적 귀납법과 다를 바 없음