

# <파이썬 알고리즘 스터디\_1>

\* Big O notation  $O()$

- 복잡도의 정근적 상한을 나타냄

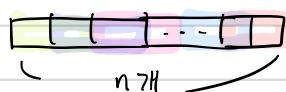
-  $O(g(n))$  이 정근적 상한이라는 뜻 =  $\frac{g(n)}{n \cdot n^2 \dots}$  보다 큰 모든 n에 대해 항상  $O(g(n))$  보다 크다.  
 $n \cdot n^2 \dots$  임의의 상수  
최고차항의 차수  $\leq g(n)$ 인 것들의 합함

## <정렬>

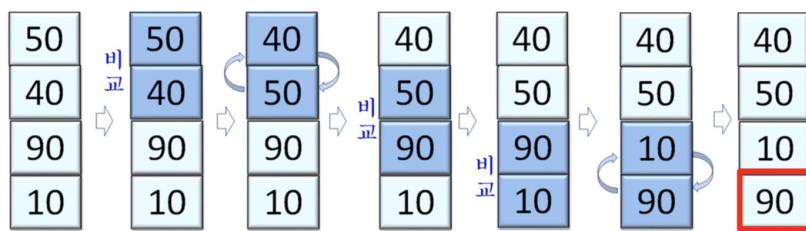
### 1. 버블정렬 (Bubble sort)

- 아웃하는 숫자를 비교하여 작은 수를 앞쪽으로 이동시키는 과정·반복·정렬

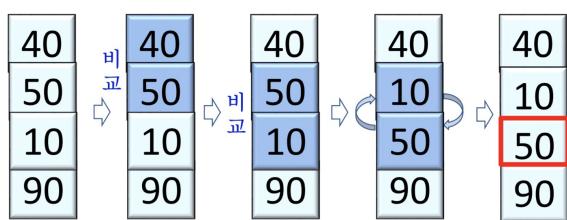
- pass : 두 개씩 양옆의 숫자를 비교하는 과정을 배열 전체에 수행하는 것. 1번째 pass는 n-1 번 비교  
2번째 pass는 n-2 번 비교  
⋮



- 1번째 pass



- 2번째 pass



- 시간복잡도

$$\text{비교횟수} = (n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2} = O(n^2)$$

비교시 자리바꿈 시간 =  $O(1)$

$$\rightarrow O(n^2) \times O(1) = O(n^2)$$

- pseudo code

input : 크기가 n인 배열 A

output : 정렬된 배열 A

for pass : 1 to n-1

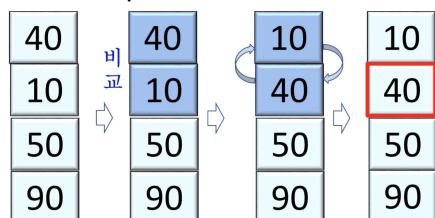
    for i : 1 to n - pass

        if (A[i-1] > A[i]) :

            A[i-1] ↔ A[i]

return 배열 A

- 3번째 pass



Worst case : 악운

장렬 순서에 장관없이 항상 일정한 시간복잡도

= 입력에 민감하지 않은 (input insensitive) 알고리즘

## 2. 선택정렬 (Selection Sort)

- 최댓값을 선택하여 제일 끝쪽의 것과 바꿔줌. 반복 (1번째에서는 9개중 제일 작은 값 1번째 위치)

2번째에는  $n-1$ 개 중 제일 작은 값 2번째 위치)  
:  $A[1] \sim A[n-1]$

	A[0]	40	70	60	30	10	50	A[5]
m=6								
①		40	70	60	30	10	50	최솟값
②		10	70	60	30	40	50	최솟값
③		10	70	60	30	40	50	최솟값
④		10	30	60	70	40	50	최솟값
⑤		10	30	60	70	40	50	최솟값
⑥		10	30	40	70	60	50	최솟값
⑦		10	30	40	50	60	70	최솟값
⑧		10	30	40	50	60	70	

### - 시간복잡도

**비교횟수** : 첫번째. 최솟값 찾는데  $(\eta_1)$ 번, 바꾸는데  $O(1)$  시간  
두번째. 최솟값 찾는데  $(\eta_2)$ 번, 바꾸는데  $O(1)$  시간

$$\rightarrow (n-1) + (n-2) + \dots + 1 = \frac{(n-1)n}{2} = O(n^2) \times O(1) = O(n^2)$$

- pseudo code

input : 크기가 n인 배열 A

Output : 정렬된 배열 A

Acid

for i : 0 to n-2    {

$\min = i$  # 최솟값의 위치

for  $j = i+1$  to  $n-1$  {

if ( $A[j] < \min = i$ )

$A[i] \leftrightarrow A[min]$  # 앞  $\leftrightarrow$  최솟값

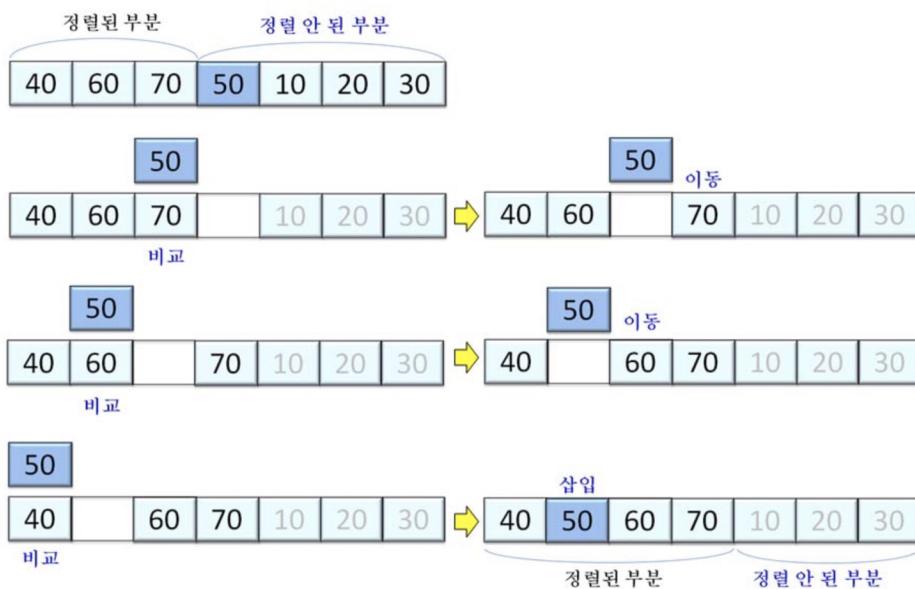
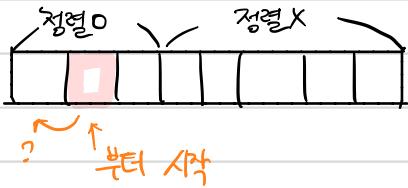
3  
return 배열 A.

입력 상태에 따라 수행 시간이 달라짐.

/ 나거의 정렬된 상태일 때 성능 제일 good.

### 3. 삽입정렬 (insertion sort)

- 배열을 정렬된 부분(앞), 정렬되지 않은 부분(뒤)로 나눈 후, 정렬되지 않은 부분의 제일 왼쪽 원소를 정렬된 부분의 가장 오른쪽 부분부터 왼쪽으로 움직이면서 가장 적절한 위치로 옮기는 정렬. 반복



#### - 시간 복잡도

- 첫번째 loop - 최대 1번 비교
  - 두번째 loop - 최대 2번 비교
  - ⋮
  - 마지막 loop - 최대  $n-1$  번 비교
- $\leftarrow \frac{(n-1)n}{2} = O(n^2) \times O(1) = O(n^2)$   
비교는데 걸리는 시간

#### - Pseudo code

input : 크기가 n인 배열, output : 정렬된 배열 A

for i = 1 to n-1 #  $i =$  정렬되지 않은 부분의 제일 왼쪽 원소

    current element = A[i]

    j = i-1 #  $j =$  정렬된 부분의 제일 오른쪽 원소

    while ( $j >= 0$ ) and ( $A[j] >$  current element) #  $j$ 가 배열 범위 안 + 정렬된 것 같음

        A[j+1] = A[j] # 비교원소와 오른쪽 원소 바꿔줌

        j  $\leftarrow j-1$

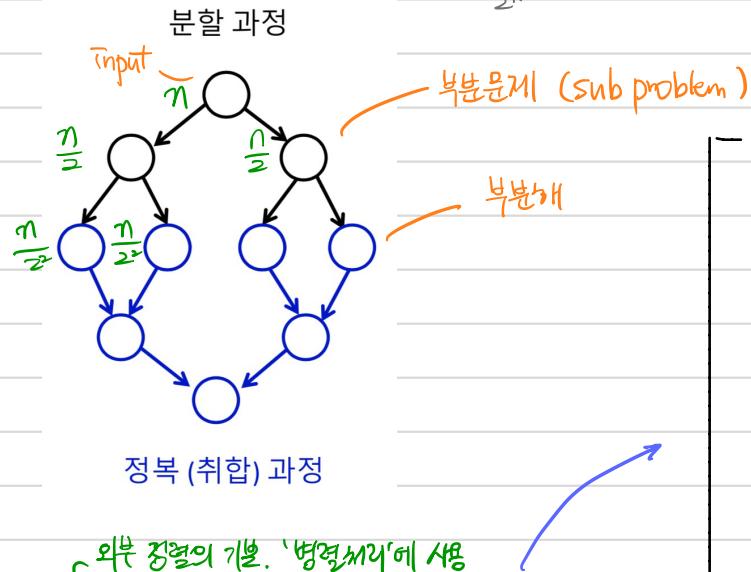
    # 앞쪽 계속 옮겨가며 위치 비교

    A[j+1]  $\leftarrow$  current element

}  
return A

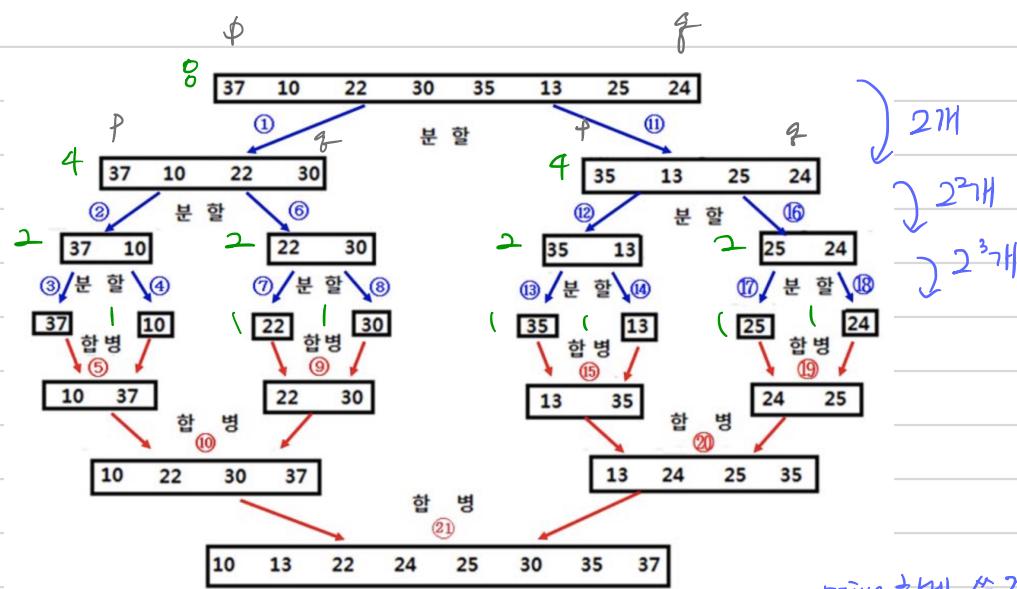
## <분할정복 (Divide & Conquer) >

- 주어진 문제의 Input 을 분할하여 문제를 해결하는 방식
- input 크기가  $n$  일 때,  $\log_2 n$  번 분할해야 더 이상 분할할 수 없는 크기 됨 (부분해의 크기는  $\frac{n}{2^k}$ )  
 $\frac{n}{2^k} = 1$  일 때 분할 불가.



- 시간복잡도
- 분할 : 그번의 재귀 호출 =  $O(1)$
- 합병 - 각 층에서의 비교 횟수 =  $n+m-1 = O(n)$   
앞의 정렬된 배열 | 뒤의 정렬된 배열
- 층의 개수 :  $\log_2 n$   
 $\rightarrow \log_2 n \times O(n) = O(n \log n)$

### 1. 합병정렬 (merge sort)



recursive하게 쓸 것이다 때문에.

- Pseudo Code

input :  $A[p] \sim A[q]$ , output : 정렬된  $A[p] \sim A[q]$

Mergesort ( $A, p, q$ )

if  $(p < q)$  { # 배열의 원소 수 2개 이상일 때

$k = \lfloor (p+q)/2 \rfloor$  #  $k$  = 절반 인덱스

    Mergesort ( $A, p, k$ ) # 앞부분 재귀 호출

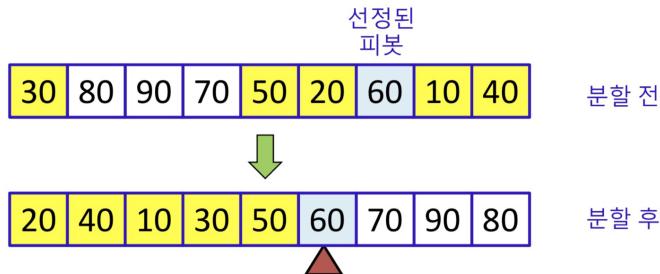
    Mergesort ( $A, k+1, q$ ) # 뒷부분 재귀 호출

$A[p] \sim A[k]$  와  $A[k+1] \sim A[q]$  합병

input 크기가 크면 삽입정렬과 같이 사용해서 성능을 높임

## 2. 퀵정렬 (Quick sort)

- 2개의 부분 문제로 분할하는데, 각 부분문제의 크기가 일정하지 않은 형태의 알고리즘
- 선택된 **pivot**을 중심으로 작은건 왼쪽, 큰건 오른쪽으로 정렬  
(1) 랜덤 추출, 2) 가장 왼쪽 원소, 중간 원소, 가장 오른쪽 원소의 중앙값



0	1	2	3	4	5	6	7	8	9	10	11
8	3	11	9	12	2	6	15	18	10	7	14

0	1	2	3	4	5	6	7	8	9	10	11
2	3	7	6	8	12	9	15	18	10	11	14

· pivot을 제일 왼쪽으로 옮겨놓고  
pivot을 기준으로 작은 수와 큰수를 각각 교환  
후 pivot 제일 작은수와 교환

### - 시간 복잡도

- Merge sort 와 동일  $O(n \log n)$  - 최선경우

- 최악의 경우 (pivot이 제일 큰 수자 or 제일 작은 수자) 하나하나 다 비교  $\rightarrow (n-1) + (n-2) + \dots + 1 = \frac{(n-1)n}{2} = O(n^2)$

### - Pseudo code

Quicksort (A, left, right)

input : 배열 A [left] ~ A [right]

output : 정렬된 배열 A [left] ~ A [right]

if (left < right) {

    pivot = A [left] ~ A [right] 중 하나 선택.

    A [left]  $\leftrightarrow$  pivot

    for i : 0 to right - left

        if (pivot < A[i]) : A [left] ~ A [p-1]로 옮김

        else : A [p+1] ~ A [right]로 옮김

    pivot = A [p]

    l = Quicksort (A, left, p-1) # pivot보다 작은 그룹 재귀호출

    r = Quicksort (A, p+1, right)

    return l + A [p] + r