3.3 Sorting

값들의 원소를 (ascending order 오름차순 / decending order 내림차순) 으로 정렬.

Sorting Algorithms

① O(n²)

- Bubble sort : pair로 비교하여 왼쪽 index의 원소가 오른쪽보다 크/작으면 swap
- Insertion sort : index [i]의 원소는 [0 ~ i-1] 원소들 비교하여 알맞은 자리에 넣음
- Selection sort : 배열의 원소들 중 가장 큰/작은 원소의 index 찾아 [0~n] index랑 swap

② 더 빠른 Algorithm. ⇒ Divide & conquer

- Merge sort
- Quick sort

03-1. Merge Sort

주어진 n개의 원소로 이루어진 집합을 2개의 set으로 분할, 각 set을 sort 하고 merge
⇒ 1개의 정렬된 원소의 집합으로 n개의 데이터를 봄!

Three steps

- Divide
  : 데이터를 2개로 split
- Conquer
  : 각 반반씩 재귀로 sort
- Combine
  : 두 정렬된 sub-리스트를 merge

merge sort algorithms

```
void msort( int s, int e, int A[] )
{
    if ( s == e )
        return;
    m ← (s+e)/2;
    msort ( s, m, A );
    msort ( m+1, e, A );
    merge ( s, m, m+1, e, A );
}
```

degererate case : 더이상 쪼개질 수 X

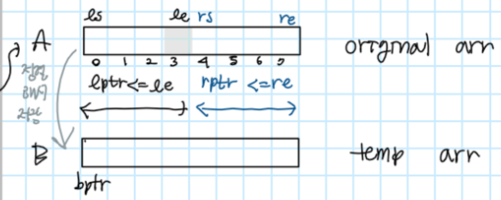
divide → O(1)

conquer → 2T(n/2)

combine → O(n)

```
void merge( int ls, int le, int rs, int re, int A[] )
{
    int lptr = ls, rptr = rs, bptr = 0;
    int *B = (int *) calloc ((le - ls)+(re - rs)+2, sizeof(int));

    while ( lptr <= le && rptr <= re ) {
        if ( A[lptr] < A[rptr] )
            B[bptr++] = A[lptr++];
        else
            B[bptr++] = A[rptr++];
    }
    if ( lptr > le )      left index가 끝까지 안갔을 경우.
        for ( int i = rptr; i <= re; i++ )
            B[bptr++] = A[i];
    if ( rptr > re )      right index가 끝까지 안갔을 경우.
        for ( int i = lptr; i <= le; i++ )
            B[bptr++] = A[i];
    A ← B; // B에 쌓은 값 A로 Copy!
}
```



performance analysis.

In master theorem

$T(n) = 2T(\frac{n}{2}) + O(n) + O(1)$

$a=2 \quad b=2 \quad d=1$

$1 = \log_2 2$

$O(n^1 \log n)$

$\therefore T(n) = O(n\log n)$

In repeated substitution

$\rightarrow T(n) = 2T(\frac{n}{2}) + n$

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

$2^2 T(\frac{n}{2^2}) = 2^3 T(\frac{n}{2^3}) + 2^2\frac{n}{2^2}$

⋮ (k번)

$+\ 2^{k-1} \cdot T(2) = 2^k T(1) + 2 \cdot 2^{k-1}$ $\rightarrow (2^k = n,\ k = \log_2 n)$, $T(1)=1$

$T(n) = 2^k T(1) + nk$

$= \log n + n\log n$

$= n(1+\log n)$

$\therefore T(n) = O(n\log n)$

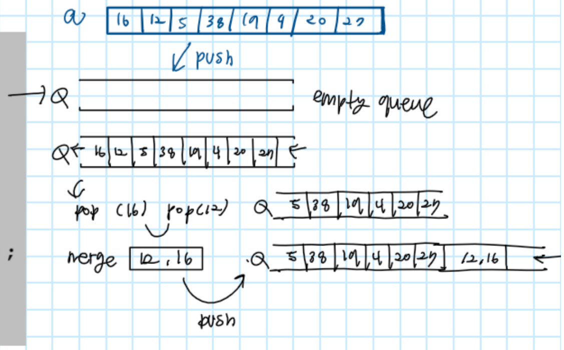
Iterative merge sort

```
int  *iteractive_mergesort ( int n, int *a )
{
    Q <- empty Queue;

    for ( int i = 0; i < n; i++ )
        Q.push ( a[i] )

    while ( Q.count ( ) > 1 ) {
        Q.push ( merge ( Q.pop ( ), Q.pop ( ) ) );

    return Q.pop ( );
}
```



성능 왜 O(nlogn)?

while ① : n개의 Input을 pair로 묶어 $\frac{n}{2}$ 회씩 merge 수행. → O(n)

↓

while문에서 Q의 길이가 $\frac{1}{2}$ 줄어듦: $n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \rightarrow \cdots \rightarrow O(\log n)$

$\therefore O(n\log n)$

summary.

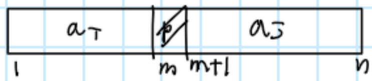
	degenerate case	divide	conquer	combine	performance
merge sort	n = 1 (s = e)	m = (s+e)/2	ms (s, m); ms (m+1, e);	merge (s, m, e);	2T(n/2) + O(n) = O(n log n)

03-2. Quick sort

\# Quick sort란? n개의 원소로 이루어진 sequence를 두개의 set으로 split함.

\# 원소의 rearrange 이기 때문에, 이후 merging의 과정 필요X.

\# partitioning 후에



$a_i \le a_j$ 꼴로 partition 됨.

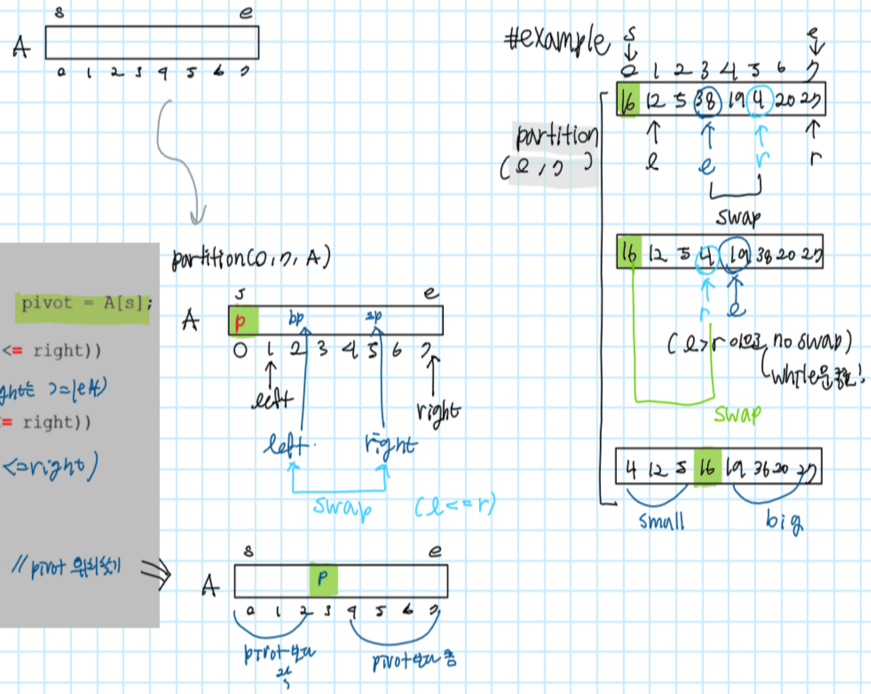
\# Two step.

- divide : 2개의 list로 split. (with partitioning)
- conquer : 각 반이 recursively sort 됨.

\# quick sort Algorithms

```
void quick_sort( int s, int e, int A[] )
{
    if ( s >= e )      ] degenerate case : 정렬
        return;
    int m = partition ( s, e, A ); //divided
    quick_sort ( s, m-1, A );      ] conquer
    quick_sort ( m+1, e, A );
}   Combination X.
```

```
int partition ( int s, int e, int A[] )
{   int pivot, left, right;
    left = s+1;          right = e;          pivot = A[s];
    while (left <= right) {
        while ((A[right] >= pivot) && (left <= right))
            right--;
        // pivot 보다 작으면 right stop (물론 right는 >=left)
        while ((A[left] <= pivot) && (left <= right))
            left++;
        // pivot 보다 크면 left stop (물론 left<=right)
        if ( left <= right )
            swap ( A[left], A[right] );
    }
    swap ( A[right], A[s] ); // A[s]: pivot; //pivot 위치찾기
    return right; //pivot index 반환
}
```



\# performance analysis ?

$$T(n) = (n+1) + \frac{1}{n}\sum_{k=1}^{n}\{T(k-1) + T(n-k)\}$$

Quick sort vs Merge Sort

	Example	Quick sort	Merge sort
Best case	5 1 6 3 4 8 7 2	O(n log n)	O(n log n)
Worst case	1 2 3 4 5 6 7 8 8 7 6 5 4 3 2 1	$O(n^2)$	O(n log n)
Average case	4 1 3 6 5 2 9 8	O(n log n)	O(n log n)

$T(n) = T(n-1) + O(n)$: 한쪽으로 쏠려있음

summary

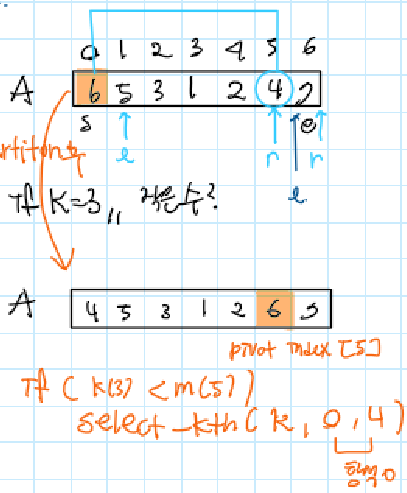
	degenerate case	divide	conquer	combine	performance
quick sort	n = 1 (s>=e)	m = partition ( );	qs (s, m-1); qs (m+1, e);	-	O(n log n)
merge sort	n = 1 (s = e)	m = (s+e)/2	ms (s, m); ms (m+1, e);	merge (s, m, e);	2T(n/2) + O(n) = O(n log n)

3.4 Medians

Quick sort 의 pivot 왼쪽은 다 작은수 / 오른쪽은 다 큰수 라는 사실에 확인

- select (DnC) : 정렬되지 않은 list에서 k번째로 작은 값을 찾아라.

```
void select_kth ( int k, int s, int e )
{
    if ( s == e )
        return A[s];            ) degenerate case
                       1번째 partition 후 pivot의 index.
O(n) ( int (m) = partition ( s, e ); // divided
                       Quicksort pivot partition.
if/else  if ( k < m )
T(n/2)  (   select_kth ( k, s, m - 1 );      ]
         else if ( k > m )                    ] conquer
            select_kth ( k-m, m + 1, e );    ]
         else
            return A[k];
}
```



performance analysis.

$T(n) = T(\frac{n}{2}) + O(n) = T(\frac{n}{2}) + n$

$a = 1 \quad b = 2 \quad d = 1$

$1 > \log_2 1$

$\therefore O(n^d) = O(n^1)$

$(\because)\ T(n) = O(n)$

summary.

	degenerate case	divide	conquer	combine	performance
median (find k-th)	n = 1 (s>=e)	m = partition ( );	select (k, s, m-1); or select(k-m, m+1, e);	-	T(n/2) + O(n) = O(n)