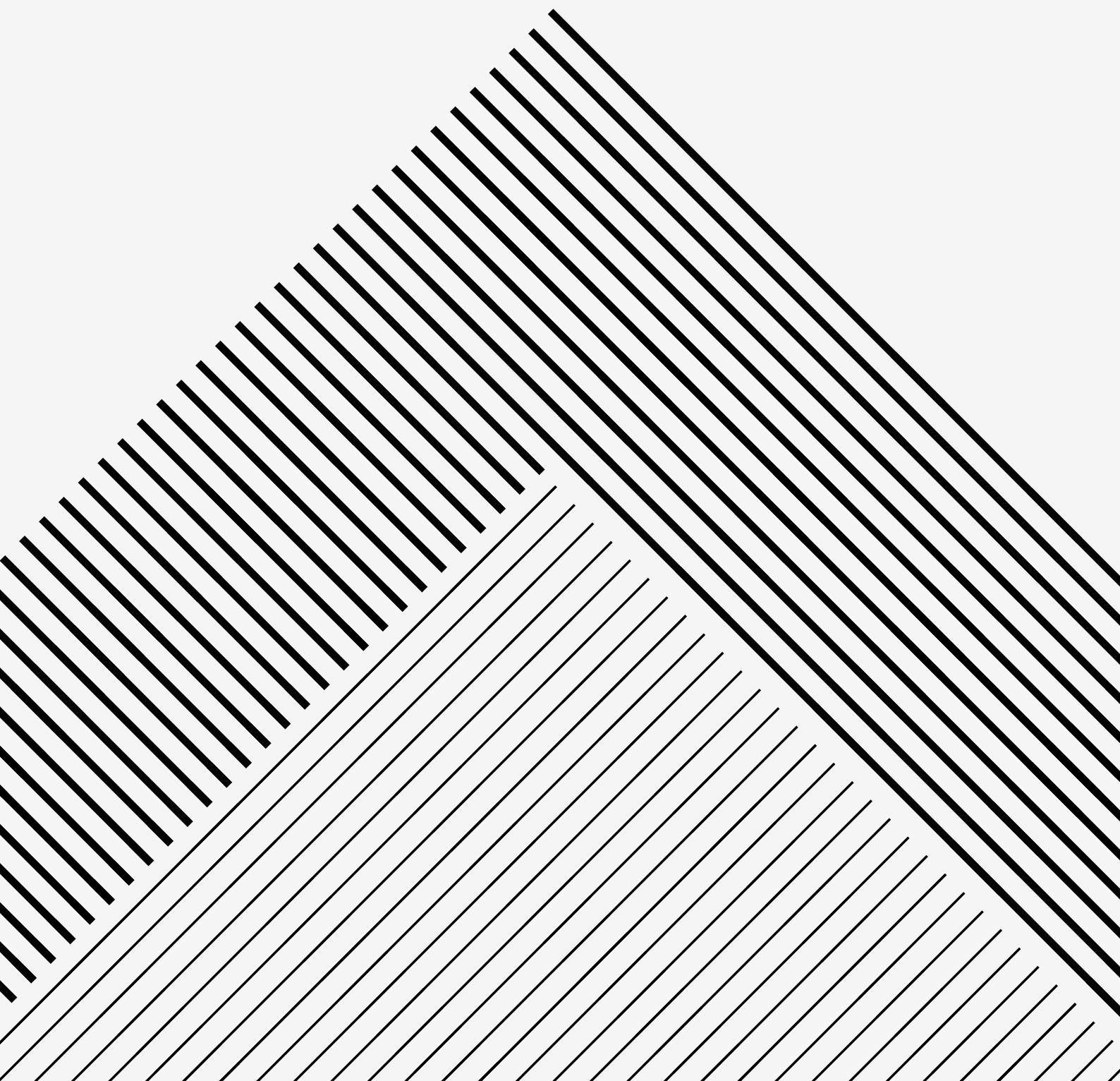


Sorting Algorithms



Sorting Algorithms.

pairwise vs distribute

internal sort vs external sort

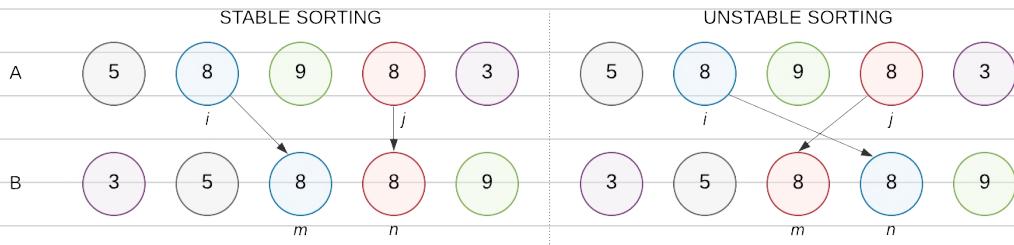
bubble sort
selection sort
insertion sort
shell sort
heap sort
merge sort
quick sort

bucket sort
counting sort
radix sort

* 모든 정렬은 원소를 가진 알고리즘.

정렬의 안정성

key:value쌍을 가진 객체 사이에서 같은 키를 갖는 객체 간 순서가 유지될 때 \rightarrow 안정성 Stability.

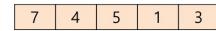


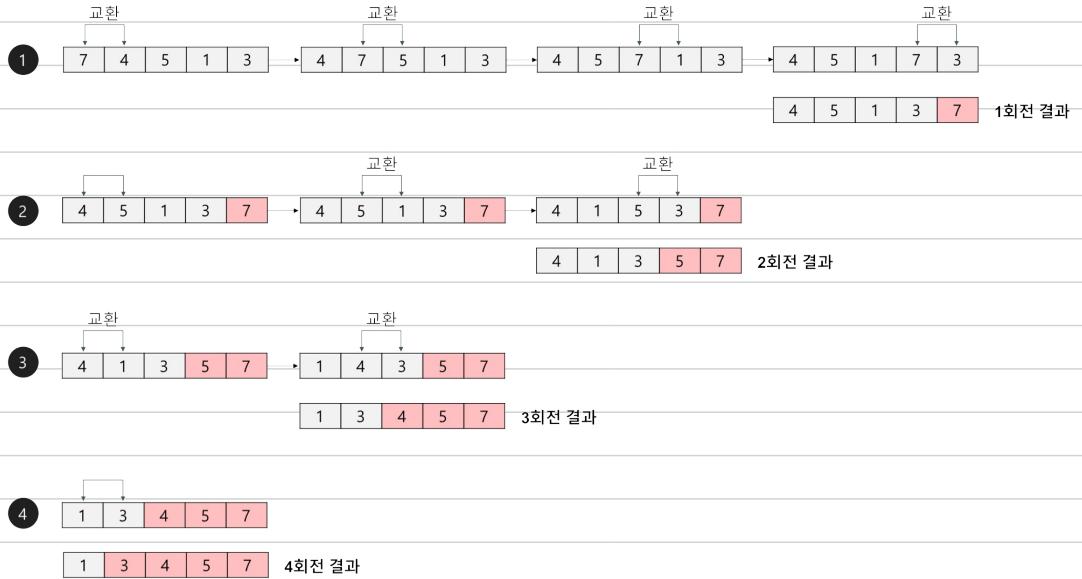
- 안정성 정렬
bubble sort, insertion sort, merge sort
- 불안정성 정렬
selection sort, quick sort, heap sort, shell sort

Bubble Sort

인접한 두 element를 비교, swap 하는 방식의 정렬 알고리즘

오름차순 정렬인 경우...

초기상태 



오름차순 완성상태 

시간 복잡도 $T(n) = O(n^2)$

$$\text{비교 횟수} : \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

swap 횟수 $\lceil \text{최대} : 3 \cdot \frac{n(n-1)}{2} \rceil$ ($\because \text{swap} \rightarrow 3\text{번의 이동 필요}$)
 카운트: 0

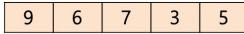
```
Swap(a, b)
tmp = a
a = b
b = tmp
```

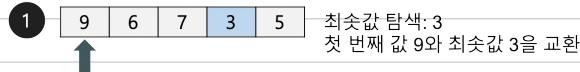
Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
삽입정렬	n	n^2	n^2	7.438
선택정렬	n^2	n^2	n^2	10.842
버블정렬	n^2	n^2	n^2	22.894
셀 정렬	n	$n^{1.5}$	n^2	0.056
퀵 정렬	$n \log_2 n$	$n \log_2 n$	n^2	0.014
힙 정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.034
병합정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.026

Selection Sort

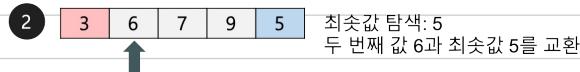
In-place sorting algorithm. → 입력받은 배열의 최적적인 메모리 요구 X

각 순서의 위치는 정해져 있고, 어떤 원소를 넣을지 selection하는 algorithm

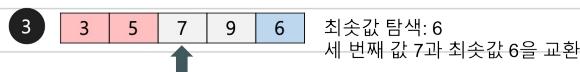
초기상태 

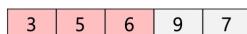


 1회전 결과



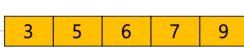
 2회전 결과



 3회전 결과



 4회전 결과

오름차순
완성상태 

* 자료 이동 횟수가 미리 정해짐.

* 암정당 확보 X

$$\text{시간 복잡도. } T(n) = \frac{n-1}{1} i = \frac{n(n-1)}{2} = O(n^2)$$

비교횟수: $\text{이중 반복문 } \dots \text{ 외부 loop } (n-1) \text{ 번.}$
 $\text{내부 loop } (n-1), (n-2), \dots, 1 \text{ 번}$

swap 횟수: $(n-1) \text{ 번 교환. }$ 상수간작법.

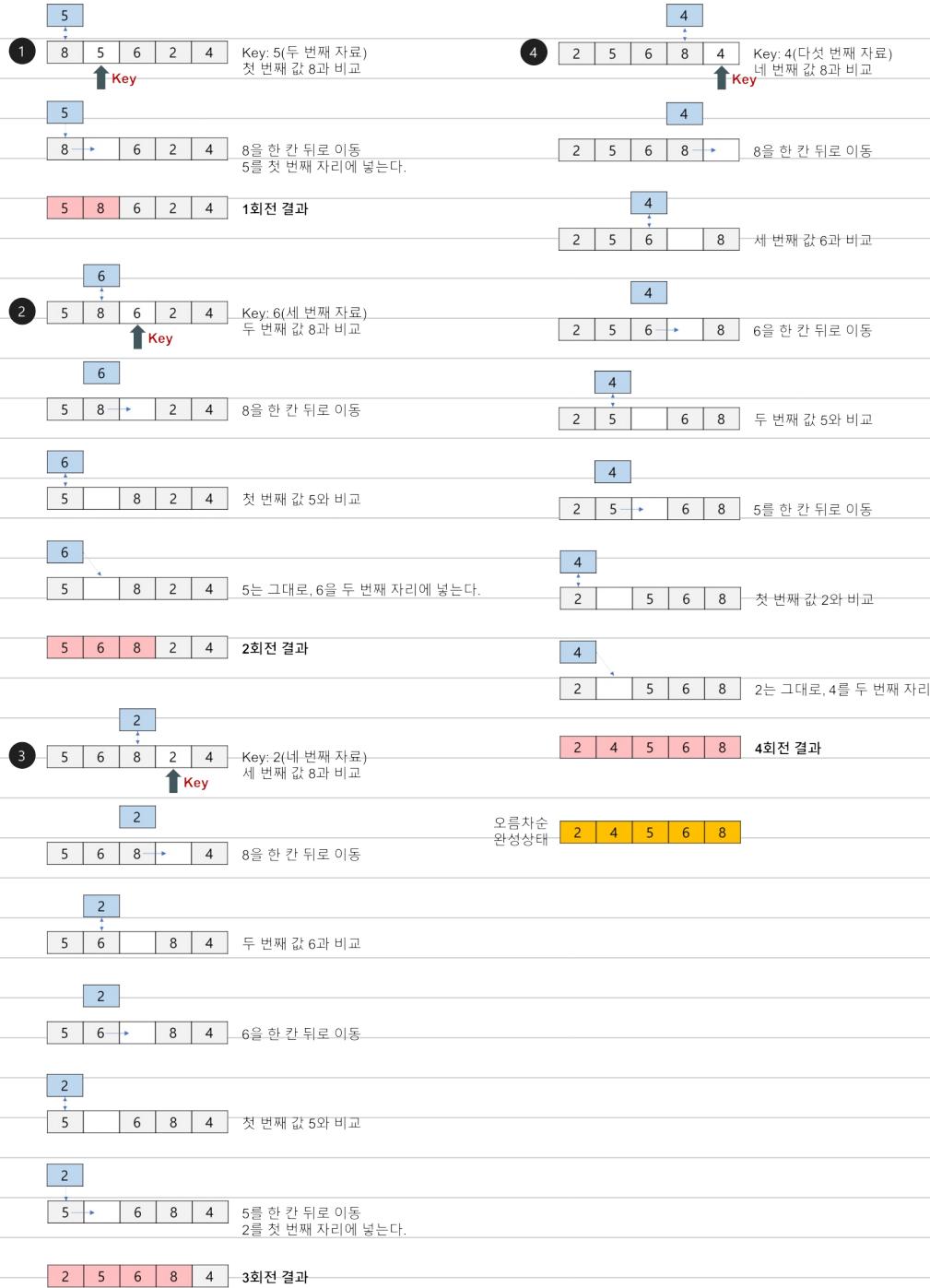
→ $3(n-1)$ 번 이동.

Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
삽입정렬	n	n^2	n^2	7.438
선택정렬	n^2	n^2	n^2	10.842
버블정렬	n^2	n^2	n^2	22.894
셀 정렬	n	$n^{1.5}$	n^2	0.056
퀵 정렬	$n \log_2 n$	$n \log_2 n$	n^2	0.014
힙 정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.034
병합정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.026

Insertion Sort

자료 배열의 모든 요소는 앞에서부터 차례대로. 이미 정렬된 부분과 비교, 자신의 위치를 찾아 삽입해 정렬
매 순서마다 해당 원소 삽입할 수 있는 위치 찾아 삽입.
⇒ index 1 (2nd element) 부터 시작!

초기상태 8 5 6 2 4



* 안정 정렬.

* 이미 정렬된 경우 + 작은 데이터셋일 경우. \Rightarrow 퀵.

* 코드 이동 \uparrow . 러코드 크기 및 양 많을 경우 \Rightarrow 볼드.

시간 복잡도.

• 첫번. \downarrow 비교 $(n-1)$ 번

 | 이동 X

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

• 첫번. (여러일 경우) { 비교 | 왼쪽 loop \rightarrow 오른 { $n-1, n-2, \dots, 2, 1$ } ... $O(n^2)$ }
 | swap | 오른 loop 각 단계별 $(i+1)$ 번 ... $O(n^2)$

$$\Rightarrow T(n) = O(n^2)$$

Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
삽입정렬	n	n^2	n^2	7.438
선택정렬	n^2	n^2	n^2	10.842
버블정렬	n^2	n^2	n^2	22.894
셀 정렬	n	$n^{1.5}$	n^2	0.056
퀵 정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.014
힙 정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.034
병합정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.026

Shell Sort

삽입 정렬 보완

전체를 한 번에 정렬하지 않는다. derived from 빠른 정렬속도 of 삽입정렬 at 어느정도 정렬된 배열
⇒ 전체를 비연속적인 부분으로 나누어서 부분적 삽입 정렬 시행.

부분리스트의 개수 줄여가며 삽입정렬. until 부분리스트가 1개, 즉 전체가 끝 때까지!!

$$\text{Skip number or gap} = \frac{(\text{정렬할 값의 수})}{2}$$

혹수가 좋다!

초기상태

10	8	6	20	4	3	22	1	0	15	16
----	---	---	----	---	---	----	---	---	----	----

 정렬할 값의 수: 10
간격(gap) k의 초기값: $10/2 = 5$

1

간격 $k=5$ 일 때의 부분 리스트들

10				3					16
8					22				
	6					1			
		20					0		
			4					15	

하나의 부분 리스트

간격 $k=5$ 일 때의 부분 리스트를
각각 삽입 정렬로 정렬

3				10				16
8					22			
	1					6		
		0					20	
			4					15

1회전 결과

3	8	1	0	4	10	22	6	20	15	16
---	---	---	---	---	----	----	---	----	----	----

다음 k의 값: $(5/2)+1 = 3$

2

간격 $k=3$ 일 때의 부분 리스트들

3			0			22			15	
8				4			6			16
	1				10			20		

간격 $k=3$ 일 때의 부분 리스트를
각각 삽입 정렬로 정렬

0			3			15		22		
4				6			8			16
	1				10			20		

2회전 결과

0	4	1	3	6	10	15	8	20	22	16
---	---	---	---	---	----	----	---	----	----	----

다음 k의 값: $3/2 = 1$

3

간격 $k=1$ 일 때의 부분 리스트들

0	4	1	3	6	10	15	8	20	22	16
---	---	---	---	---	----	----	---	----	----	----

간격 $k=1$ 일 때의 부분 리스트를
각각 삽입 정렬로 정렬

0	1	3	4	6	8	10	15	16	20	22
---	---	---	---	---	---	----	----	----	----	----

3회전 결과

0	1	3	4	6	8	10	15	16	20	22
---	---	---	---	---	---	----	----	----	----	----

오름차순
완성상태

0	1	3	4	6	8	10	15	16	20	22
---	---	---	---	---	---	----	----	----	----	----

* 비연속적 부분리스트에서의 이동 \Rightarrow 더 긴 거리 이동 ... 최종 위치와 일정한 가능성이 ↑

* 삽입정렬보다 빠르다.

* 간단한 구조.

시간 복잡도

Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
삽입정렬	n	n^2	n^2	7.438
선택정렬	n^2	n^2	n^2	10.842
버블정렬	n^2	n^2	n^2	22.894
셸 정렬	n	$n^{1.5}$	n^2	0.056
퀵정렬	$n \log n$	$n \log n$	n^2	0.014
힙 정렬	$n \log n$	$n \log n$	$n \log n$	0.034
병합정렬	$n \log n$	$n \log n$	$n \log n$	0.026

heap sort

* 내림차순 가정.

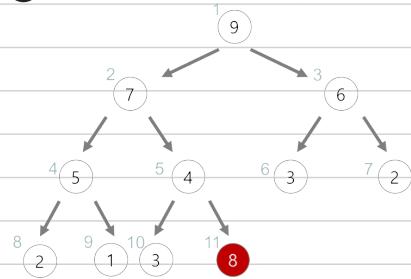
(MAX heap) or (min heap) 구조 \Rightarrow 가능!

내림차순

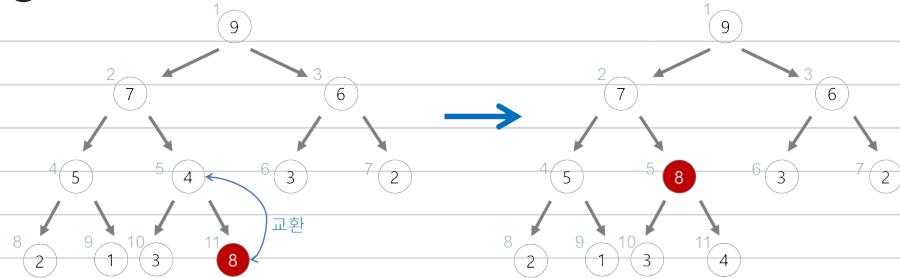
오름차순

heap의 추가

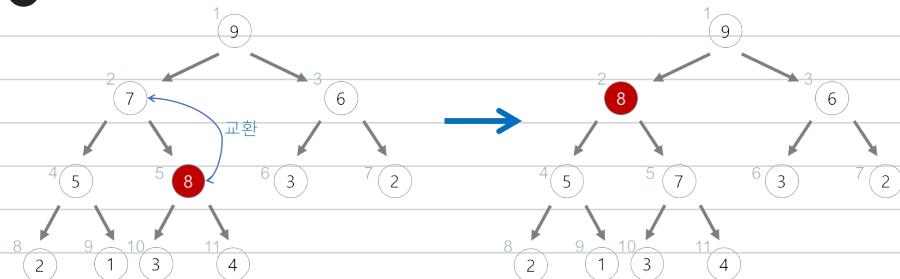
- 인덱스 순으로 가장 마지막 위치에 이어서 새로운 요소 8을 삽입



- 부모 노드 4 < 삽입 노드 8 이므로 서로 교환



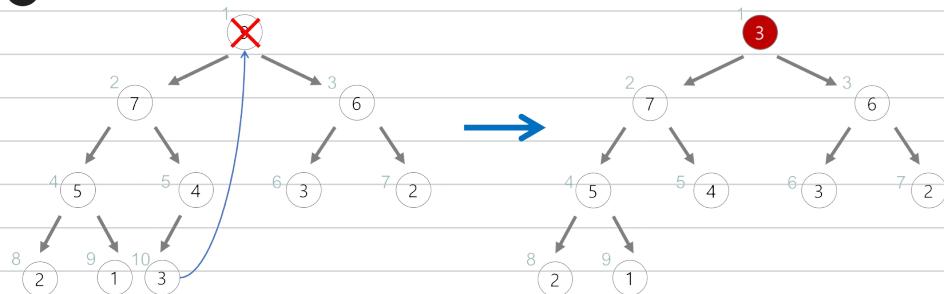
- 부모 노드 7 < 삽입 노드 8 이므로 서로 교환



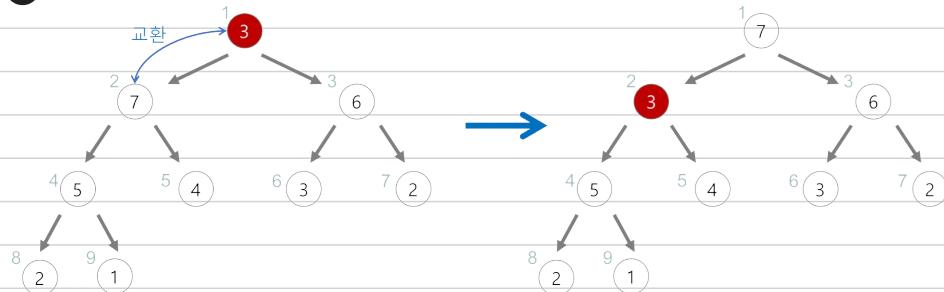
- 부모 노드 9 > 삽입 노드 8 이므로 더 이상 교환하지 않는다.

heap에서 삭제

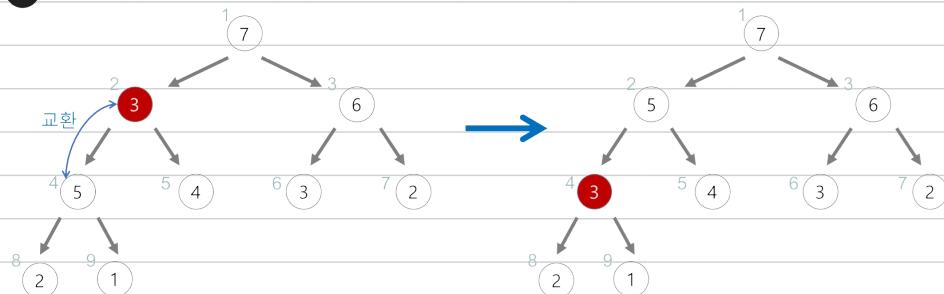
1. 최댓값인 루트 노드 9를 삭제. (빈자리에는 최대 힙의 마지막 노드를 가져온다.)



2. 삽입 노드와 자식 노드를 비교. 자식 노드 중 더 큰 값과 교환. (자식 노드 7 > 삽입 노드 3 이므로 서로 교환)



3. 삽입 노드와 더 큰 값의 자식 노드를 비교. 자식 노드 5 > 삽입 노드 3 이므로 서로 교환



4. 자식 노드 1, 2 < 삽입 노드 3 이므로 더 이상 교환하지 않는다.

시간복잡도 : $T(n) = O(n \log n)$

Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
삽입정렬	n	n^2	n^2	7.438
선택정렬	n^2	n^2	n^2	10.842
버블정렬	n^2	n^2	n^2	22.894
셀정렬	n	$n^{1.5}$	n^2	0.056
퀵정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.014
힙정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.034
병합정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.026

Merge Sort

Divide & Conquer

→ 문제를 작은 두 문제로 나누고 각각을 해결. 그 결과를 종합해 원래 문제 해결.

divide → conquer → combine

recursively call **d & c** until size of partial array is enough small.

divide : 입력 배열을 같은 크기의 부분배열들로 나누는 과정.

conquer : 부분 배열 정복. 부분배열이 충분히 작지 않다면 재귀호출(재귀)

→ 다시 부분 정복 정복.

combine : 정복된 부분배열을 하나의 배열에 합병

초기상태

21	10	12	20	25	13	15	22
----	----	----	----	----	----	----	----



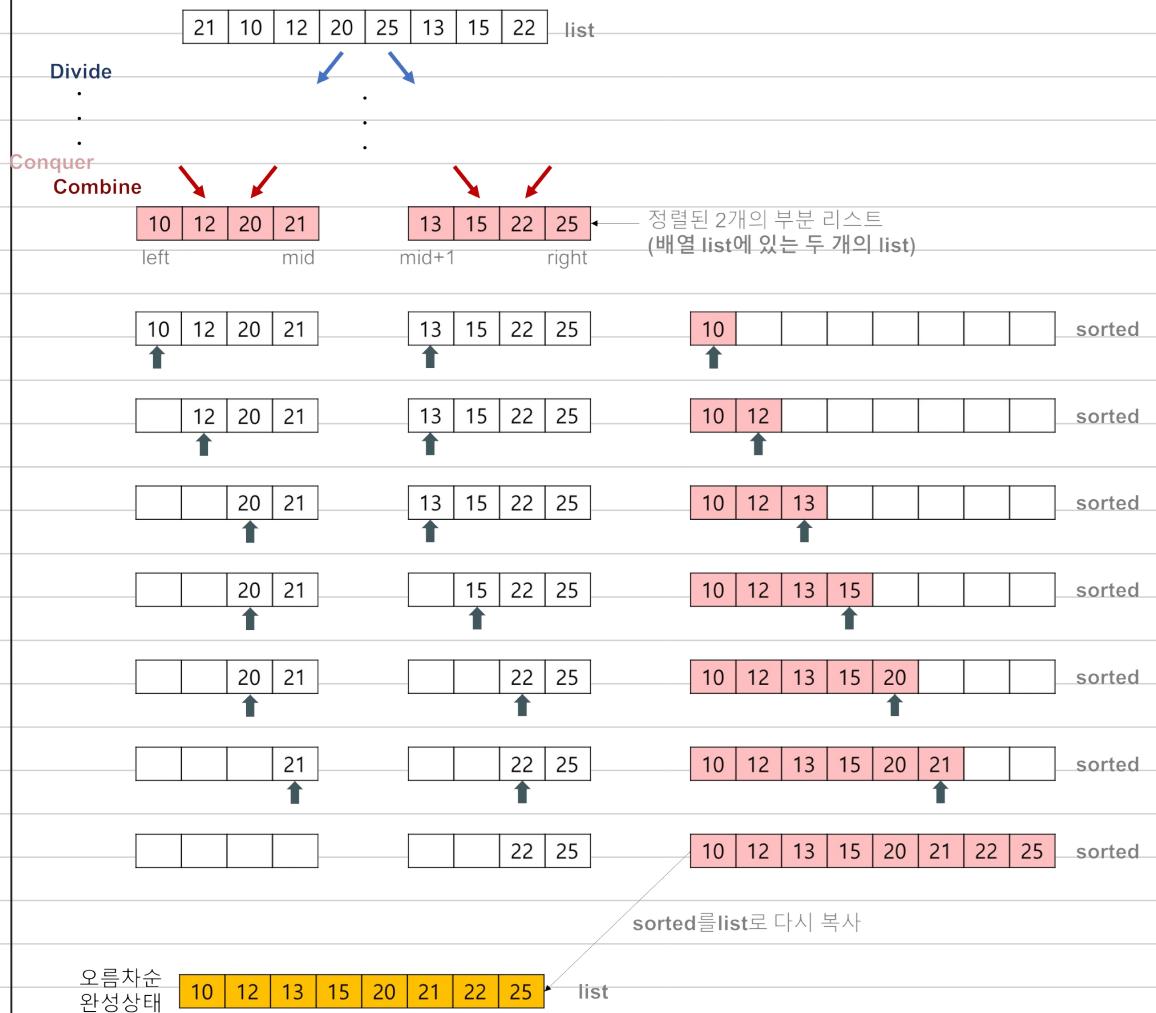
오름차순 완성상태

10	12	13	15	20	21	22	25
----	----	----	----	----	----	----	----

• 2개의 정렬된 리스트 합병

- 두 리스트 값을 하나씩 비교해 더 작은값을 새로운 리스트에 옮김
 - 둘중 하나 끝난때까지 ! 남는건 새 리스트 만들로!
 - 새 리스트를 원래 리스트에 옮기기.

초기상태 21 10 12 20 25 13 15 22



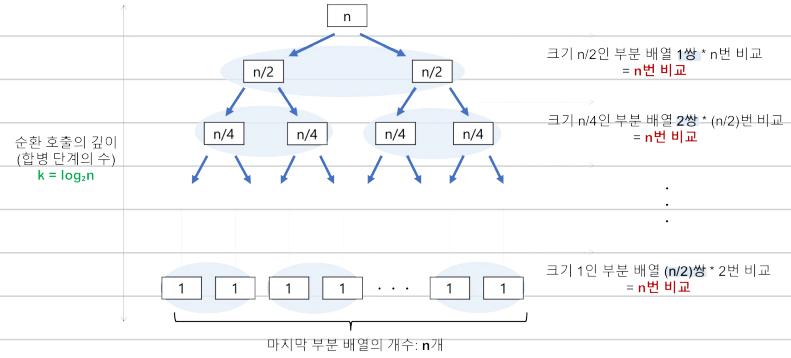
* 배열이라면 \rightarrow 임시 배열 필요.
이중 힙수 \uparrow 시간 \uparrow

- 안정정렬. 테이터 블록이 원장 틀 받는다.
 연결리스트 \rightarrow 테이터 이동 \downarrow , 제자리 정렬 (in-place sorting) 가능
 큰 데이터 \downarrow 연결리스트 \rightarrow 가장 빠름!

시간복잡도.

divide ... 비교. 이동 X

combine ... 비교.



재귀 깊이 : $n=2^k$ 개의 레벨, 깊이 = $k = \log_2 n$

하나의 합병단계 : 최대 n 번의 비교

\Rightarrow 총 $(n \log_2 n)$ 번

... 이동

재귀 깊이 : $n=2^k$ 개의 레벨, 깊이 = $k = \log_2 n$

각 합병 단계 이동 : $2n$ 번. 한 레벨에 걸친 모든 단계에서

\Rightarrow 총 $2n \log_2 n$

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

Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
삼입정렬	n	n^2	n^2	7.438
선택정렬	n^2	n^2	n^2	10.842
버블정렬	n^2	n^2	n^2	22.894
셀 정렬	n	$n^{1.5}$	n^2	0.056
퀵 정렬	$n \log_2 n$	$n \log_2 n$	n^2	0.014
힙 정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.034
병합정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.026

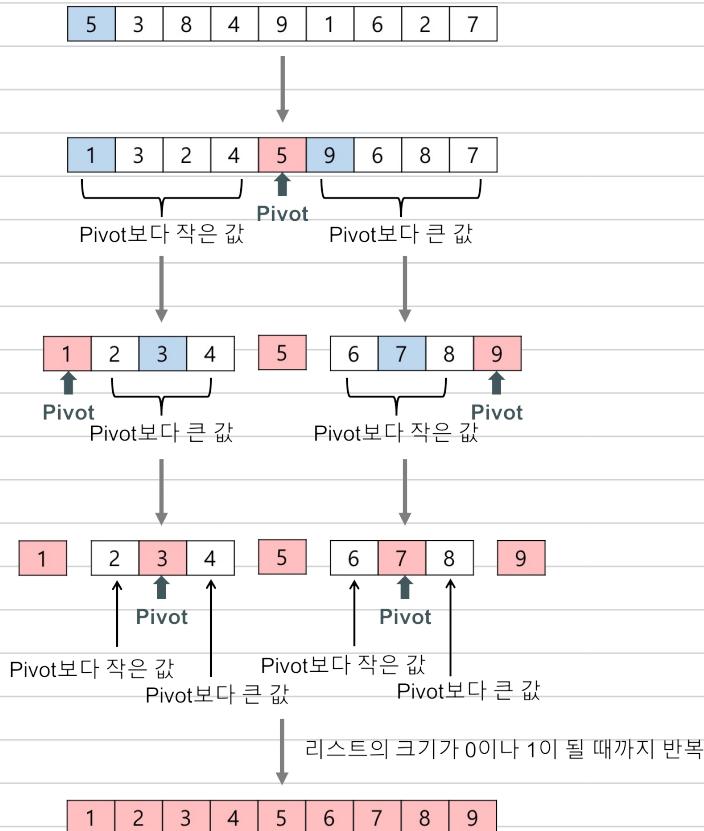
Quick Sort

Divide & Conquer Algorithm
반복 분할!

- (각트 안 한 요소 ... pivot으로 지정.
- ? < pivot \Rightarrow pivot 왼쪽
- ? > pivot \Rightarrow pivot 오른쪽.
- (pivot 왼쪽 리스트 & pivot 오른쪽 리스트 재정렬.
by 순회(재귀) until 더 이상 분할 불가
i.e., 리스트 크기 = 0 or 1

초기상태

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



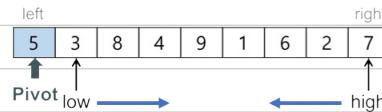
오름차순
완성상태

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

초기상태

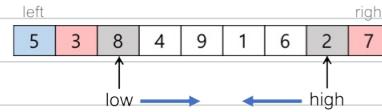
5 3 8 4 9 1 6 2 7

1

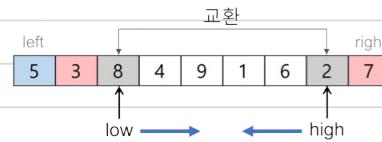
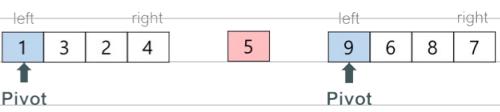


1회전 후 Pivot 5는 이미
제 위치에 있음을 알 수 있다.

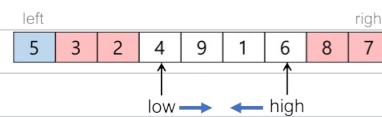
2



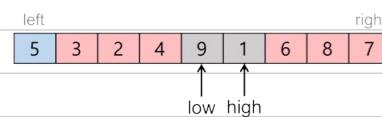
3



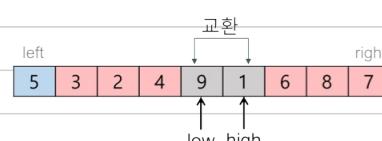
Pivot을 제외한
왼쪽 리스트와 오른쪽 리스트를
각각 독립적으로 다시 퀵 정렬(quick sort)을 한다.



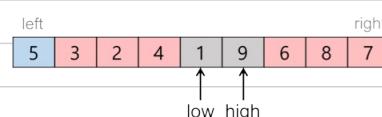
리스트의 크기가
0이나 1이 될 때까지 반복



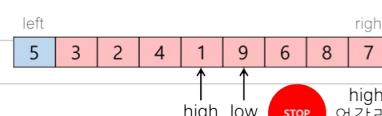
1 2 3 4 5 6 7 8 9



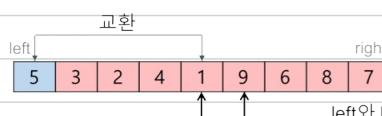
오름차순
완성상태 1 2 3 4 5 6 7 8 9



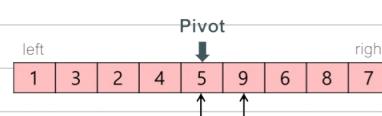
high와 low가
엇갈려서 지난 후
STOP



high와 low가
엇갈려서 지난 후
STOP



left와 high를 교환
Pivot을 가운데로 옮긴다.



Pivot보다 작은 값
(왼쪽 부분 리스트)

Pivot보다 큰 값
(오른쪽 부분 리스트)

* 배운 속도.