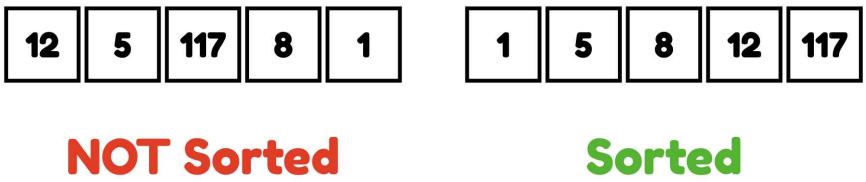
기술면접 – 정렬(Sort)

정렬

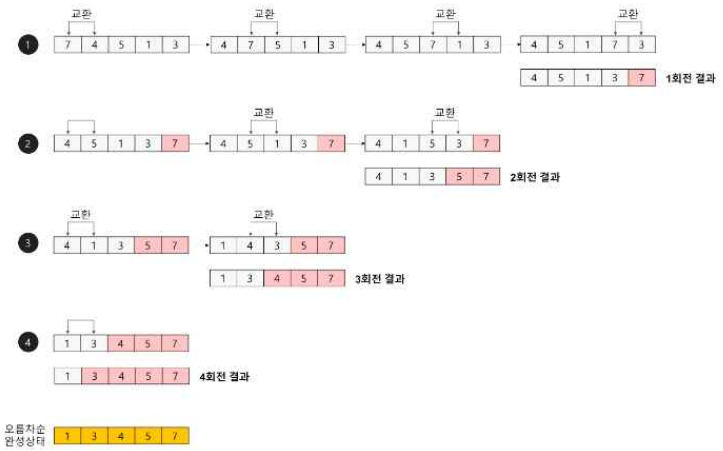
- "뭔가를 정리하는 것"
- 사전처럼 A부터 Z까지 기준으로 정렬하거나 큰 수에서 작은 수 기준으로 정렬할 수도 있다.
- 이진검색처럼 빠른 알고리즘을 사용하려면 일단 먼저 배열 정렬을 해야 한다. 정렬의 종류는 여러 가지 있고 밑에서부터 소개하겠습니다.



1. 버블 정렬(Bubble Sort)

- 서로 인접한 두 원소를 검사하여 정렬하는 알고리즘
- 인접한 2개의 레코드를 비교하여 크기가 순서대로 되어 있지 않으면 서로 교환한다.
- 선택 정렬과 기본 개념이 유사하다.

배열에 7, 4, 5, 1, 3이 저장되어 있다고 가정하고 자료를 오름차순으로 정렬해 보자.



- 버블정렬 시간복잡도

비교 횟수 : 최상, 평균, 최악 모두 일정 n-1, n-2, … , 2, 1 번 = n(n-1)/2
교환 횟수 : 입력 자료가 역순으로 정렬되어 있는 최악의 경우, 한 번 교환하기 위하여 3번의 이동(SWAP 함수의 작업)이 필요하므로 (비교 횟수 * 3) 번 = 3n(n-1)/2
입력 자료가 이미 정렬되어 있는 최상의 경우, 자료의 이동이 발생하지 않는다.
시간복잡도 : T(n) = $O(n^2)$

- 버블정렬 장단점, 특징

장점 : 구현이 매우 간단하다.

단점 :

순서에 맞지 않은 요소를 인접한 요소와 교환한다.

하나의 요소가 가장 왼쪽에서 가장 오른쪽으로 이동하기 위해서는 배열에서 모든 다른 요소들과 교환되어야 한다.

특히 특정 요소가 최종 정렬 위치에 이미 있는 경우라도 교환되는 일이 일어난다.

일반적으로 자료의 교환 작업(SWAP)이 자료의 이동 작업(MOVE)보다 더 복잡하기 때문에 버블 정렬은 단순성에도 불구하고 거의 쓰이지 않는다.

2. 선택 정렬(Selection Sort)

- 제자리 정렬(in-place sorting) 알고리즘의 하나, 입력 배열(정렬되지 않은 값들) 이외에 다른 추가 메모리를 요구하지 않는 정렬 방법
- 해당 순서에 원소를 넣을 위치는 이미 정해져 있고, 어떤 원소를 넣을지 선택하는 알고리즘
- 첫 번째 순서에는 첫 번째 위치에 가장 최솟값을 넣는다. 두 번째 순서에는 두 번째 위치에 남은 값 중에서의 최솟값을 넣는다.

배열에 9, 6, 7, 3, 5가 저장되어 있다고 가정하고 자료를 오름차순으로 정렬해 보자.

- 선택정렬 시간복잡도

비교 횟수 : 두 개의 for 루프의 실행 횟수

외부 루프: (n-1)번, 내부 루프(최솟값 찾기): n-1, n-2, … , 2, 1 번

교환 횟수 : 외부 루프의 실행 횟수와 동일. 즉, 상수 시간 작업

한 번 교환하기 위하여 3번의 이동(SWAP 함수의 작업)이 필요하므로 3(n-1)번

시간복잡도 : $T(n) = O(n^2)$

- 선택정렬 장단점, 특징

장점 :

자료 이동 횟수가 미리 결정된다.

버블 정렬과 다르게 N 번의 교환을 하지 않는다. 선택 정렬에서는 매번 싸이클마다 한 번의 교환만 하면 된다. 즉, 선택 정렬은 버블 정렬보다 훨씬 낫다

단점 :

안정성을 만족하지 않는다. 즉, 값이 같은 레코드가 있는 경우에 상대적인 위치가 변경될 수 있다.

3. 삽입 정렬(Insertion Sort)

- 자료 배열의 모든 요소를 앞에서부터 차례대로 이미 정렬된 배열 부분과 비교 하여, 자신의 위치를 찾아 삽입함으로써 정렬을 완성하는 알고리즘
- 매 순서마다 해당 원소를 삽입할 수 있는 위치를 찾아 해당 위치에 넣는다.
- 삽입 정렬은 두 번째 자료부터 시작하여 그 앞(왼쪽)의 자료들과 비교하여 삽입할 위치를 지정한 후 자료를 뒤로 옮기고 지정한 자리에 자료를 삽입하여 정렬하는 알고리즘이다.
- 즉, 두 번째 자료는 첫 번째 자료, 세 번째 자료는 두 번째와 첫 번째 자료, 네 번째 자료는 세 번째, 두 번째, 첫 번째 자료와 비교한 후 자료가 삽입될 위치를 찾는다. 자료가 삽입될 위치를 찾았다면 그 위치에 자료를 삽입하기 위해 자료를 한 칸씩 뒤로 이동시킨다.
- 처음 Key 값은 두 번째 자료부터 시작한다.

배열에 8, 5, 6, 2, 4가 저장되어 있다고 가정하고 자료를 오름차순으로 정렬해 보자.

- 삽입정렬 시간복잡도

최선의 경우

비교 횟수 : 이동 없이 각 요소들이 1번의 비교만 이루어진다. Best T(n) = O(n)

최악의 경우(입력 자료가 역순일 경우)

비교 횟수 : 외부 루프 안의 각 반복마다 i번의 비교 수행

외부 루프 : (n-1) + (n-2) + … + 2 + 1 = n(n-1)/2 = $O(n^2)$

교환 횟수 : 외부 루프의 각 단계마다 (i+2)번의 이동 발생

n(n-1)/2 + 2(n-1) = (n^2+3n-4)/2 = $O(n^2)$. Worst T(n) = $O(n^2)$

- 삽입정렬 장단점, 특징

장점

안정한 정렬 방법

레코드의 수가 적을 경우 알고리즘 자체가 매우 간단하므로 다른 복잡한 정렬 방법보다 유리할 수 있다.

대부분위 레코드가 이미 정렬되어 있는 경우에 매우 효율적일 수 있다.

단점

비교적 많은 레코드들의 이동을 포함한다.

레코드 수가 많고 레코드 크기가 클 경우에 적합하지 않다.

버블 정렬 vs 선택 정렬 vs 삽입정렬

버블 정렬

인접한 원소끼리 비교하여 교환하는 방식

셋 중에 제일 느리지만 단순함

선택 정렬

최솟값을 찾아서 맨 앞으로 이동하는 방식

버블 정렬보다 좋음

삽입 정렬

앞에서부터 차례대로 이미 정렬된 부분과 비교하여 교환하는 방식

셋 중에 제일 빠르지만 배열이 길어질수록 효율성이 떨어짐

모두 시간 복잡도는 $O(n^2)$

선택 정렬과 삽입 정렬은 사용할 메모리가 제한적인 경우에 사용하면 좋음

4. 퀵 정렬(Quick Sort)

- 퀵 정렬(Quick Sort)은 기준값(피봇; Pivot)을 중심으로 자료를 왼쪽 부분집합과 오른쪽 부분집합으로 분할한다. 왼쪽 부분집합으로 기준값보다 작은 원소를 이동시키고, 오른쪽 부분집합으로 기준값보다 큰 원소를 이동시킨다.
- 퀵 정렬(Quick Sort)은 분할과 정복(Divide and Conquer)라는 작업을 반복하여 수행한다.
- 동일한 값에 대해 기존의 순서가 유지되지 않는 불안정 정렬이다.

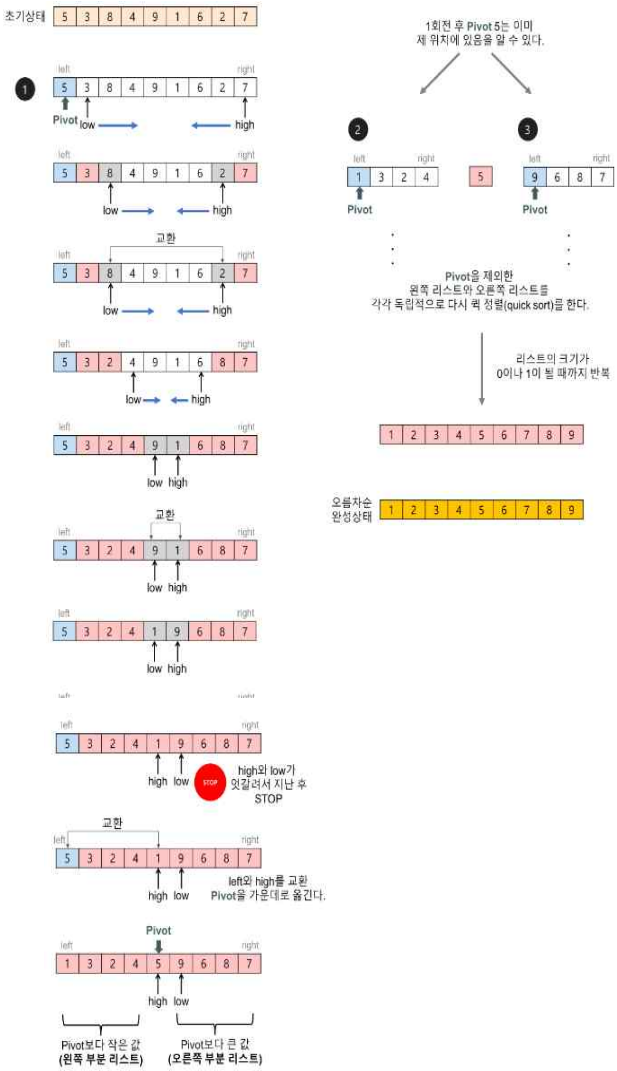
- 퀵 정렬 과정

분할(Divide): 입력 배열을 피벗을 기준으로 비균등하게 2개의 부분 배열(피벗을 중심으로 왼쪽: 피벗보다 작은 요소들, 오른쪽: 피벗보다 큰 요소들)로 분할한다.
정복(Conquer): 부분 배열을 정렬한다. 부분 배열의 크기가 충분히 작지 않으면 순환 호출을 이용하여 다시 분할 정복 방법을 적용한다.
결합(Combine): 정렬된 부분 배열들을 하나의 배열에 합병한다.
순환 호출이 한번 진행될 때마다 최소한 하나의 원소(피벗)는 최종적으로 위치가 정해지므로, 이 알고리즘은 반드시 끝난다는 것을 보장할 수 있다.

(1) 왼쪽 끝에서 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소를 찾아 L로 표시
(2) 오른쪽 끝에서 왼쪽으로 움직이며 피봇보다 작은 원소를 찾아 R로 표시
(3) L과 R을 서로 교환하고 (1) 과 (2) 를 반복 수행
(4) L과 R이 만나는 경우 멈추고 R과 피봇의 원소를 교환 (5) 피봇을 다시 정하고 (1) ~ (2) 반복
(6) 모든 부분집합의 크기가 1 이하가 되면 퀵 정렬 종료

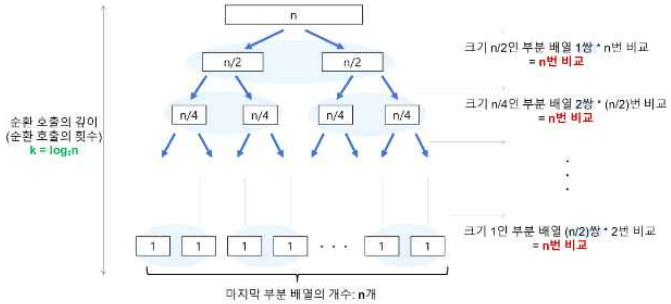
- 퀵 정렬 예시

배열에 5, 3, 8, 4, 9, 1, 6, 2, 7이 저장되어 있다고 가정하고 자료를 오름차순으로 정렬해 보자



1. 피벗 값을 입력 리스트의 첫 번째 데이터로 하자. (다른 임의의 값이어도 상관없다.)
2개의 인덱스 변수(low, high)를 이용해서 리스트를 두 개의 부분 리스트로 나눈다.
2. 1회전: 피벗이 5인 경우,
    low는 왼쪽에서 오른쪽으로 탐색해가다가 피벗보다 큰 데이터(8)을 찾으면 멈춘다.
    high는 오른쪽에서 왼쪽으로 탐색해가다가 피벗보다 작은 데이터(2)를 찾으면 멈춘다.
    low와 high가 가리키는 두 데이터를 서로 교환한다.
    이 탐색-교환 과정은 low와 high가 엇갈릴 때까지 반복한다.
3. 2회전: 피벗(1회전의 왼쪽 부분리스트의 첫 번째 데이터)이 1인 경우,
위와 동일한 방법으로 반복한다.
4. 3회전: 피벗(1회전의 오른쪽 부분리스트의 첫 번째 데이터)이 9인 경우,
위와 동일한 방법으로 반복한다.

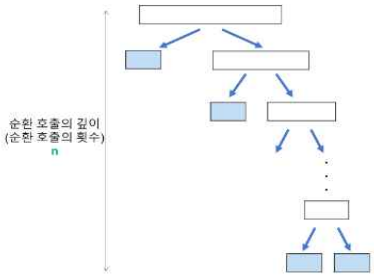
- 퀵 정렬 시간복잡도

- 최선의 경우
  - 비교 횟수
    - 

    - *순환 호출의 깊이*
      - 레코드의 개수 n이 2의 거듭제곱이라고 가정(n=2^k)했을 때, n=2^3의 경우, 2^3 -> 2^2 -> 2^1 -> 2^0 순으로 줄어들어 순환 호출의 깊이가 3임을 알 수 있다. 이것을 **일반화하면** n=2^k의 경우, k(k=log2n)임을 알 수 있다.
      - $k=\log_2 n$
    - *각 순환 호출 단계의 비교 연산*
      - 각 순환 호출에서는 전체 리스트의 대부분의 레코드를 비교해야 하므로 평균 n번 정도의 비교가 이루어진다.
      - 평균 n번
    - 순환 호출의 깊이 * 각 순환 호출 단계의 비교 연산 = $\mathbf{n\log_2 n}$

이동 횟수 : 비교 횟수보다 적으므로 무시할 수 있다.

최선의 경우 T(n) = O(nlog2n)

- 최악의 경우
  - 리스트가 계속 불균형하게 나누어지는 경우 (특히, 이미 정렬된 리스트에 대하여 퀵 정렬을 실행하는 경우)
    - 

  - 비교 횟수
    - *순환 호출의 깊이*
      - 레코드의 개수 n이 2의 거듭제곱이라고 가정(n=2^k)했을 때, 순환 호출의 깊이는 n임을 알 수 있다.
      - n
    - *각 순환 호출 단계의 비교 연산*
      - 각 순환 호출에서는 전체 리스트의 대부분의 레코드를 비교해야 하므로 평균 n번 정도의 비교가 이루어진다.
      - 평균 n번
    - 순환 호출의 깊이 * 각 순환 호출 단계의 비교 연산 = **n^2**
  - 이동 횟수
    - 비교 횟수보다 적으므로 무시할 수 있다.
  - 최악의 경우 T(n) = **O(n^2)**
- 평균
  - 평균 T(n) = **O(nlog2n)**
  - 시간 복잡도가 O(nlog2n)를 가지는 다른 정렬 알고리즘과 비교했을 때도 가장 빠르다.
  - 퀵 정렬이 불필요한 데이터의 이동을 줄이고 먼 거리의 데이터를 교환할 뿐만 아니라, 한 번 결정된 피벗들이 추후 연산에서 제외되는 특성 때문이다.

공간 복잡도 : 주어진 배열 내에서 교환을 하며 수행하므로 O(n)이다.

- 퀵 정렬 장단점, 특징

장점

속도가 빠르다.

시간 복잡도가 O(nlog₂n)를 가지는 다른 정렬 알고리즘과 비교했을 때도 가장 빠르다.

추가 메모리 공간을 필요로 하지 않는다.

퀵 정렬은 O(log n)만큼의 메모리를 필요로 한다.

단점

정렬된 리스트에 대해서는 퀵 정렬의 불균형 분할에 의해 오히려 수행시간이 더 많이 걸린다.

퀵 정렬의 불균형 분할을 방지하기 위하여 피벗을 선택할 때 더욱 리스트를 균등하게 분할할 수 있는 데이터를 선택한다.

EX) 리스트 내의 몇 개의 데이터 중에서 크기순으로 중간 값(medium)을 피벗으로 선택한다

5. 합병 정렬(Merge Sort)

- 병합 정렬(Merge Sort)은 여러 개의 정렬된 집합을 병합하여 하나의 정렬된 집합으로 만드는 정렬 방법이다. 자료를 부분집합으로 분할 하고 부분집합에 대해 작업을 정복 하고 부분집합들을 다시 결합 하는 분할과 정복(Divide and Conquer) 방법을 사용한다.
- 하나의 리스트를 두 개의 균등한 크기로 분할하고 분할된 부분 리스트를 정렬한 다음, 두 개의 정렬된 부분 리스트를 합하여 전체가 정렬된 리스트가 되게 하는 방법이다
- 병합 정렬은 순차적인 비교를 통해 정렬하므로, LinkedList의 정렬에 효율적이다.
- 동일한 값에 대해 기존의 순서가 유지되는 안정 정렬이다.

- 합병 정렬 과정

분할(Divide): 입력 배열을 같은 크기의 2개의 부분 배열로 분할한다.

정복(Conquer): 부분 배열을 정렬한다. 부분 배열의 크기가 충분히 작지 않으면 순환 호출을 이용하여 다시 분할 정복 방법을 적용한다.

결합(Combine): 정렬된 부분 배열들을 하나의 배열에 합병한다.

(1) 2개의 리스트의 값들을 처음부터 하나씩 비교하여 두 개의 리스트의 값 중에서 더 작은 값을 새로운 리스트(sorted)로 옮긴다.

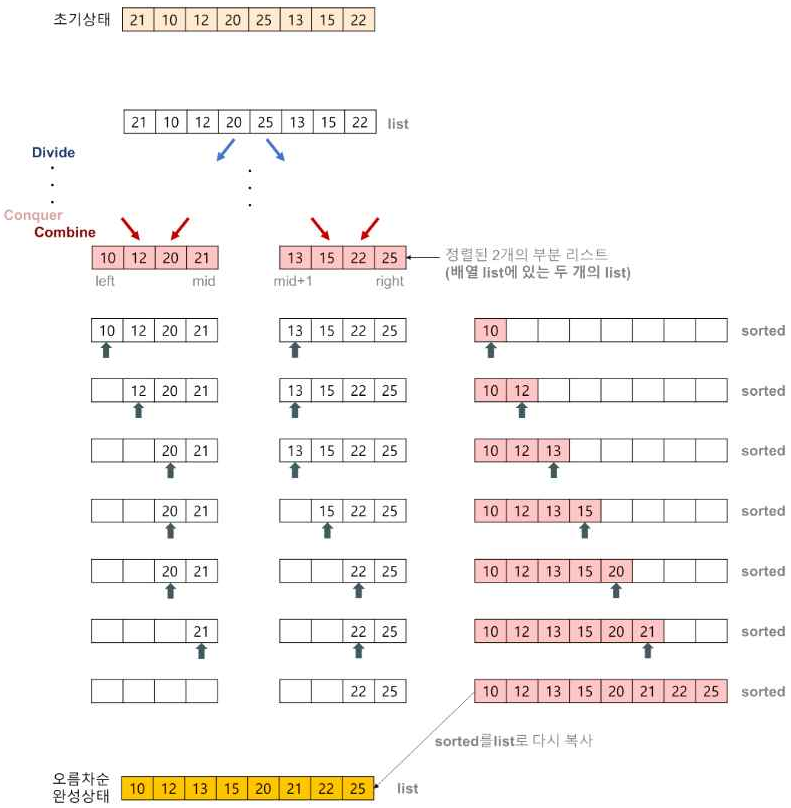
(2) 둘 중에서 하나가 끝날 때까지 이 과정을 되풀이한다.

(3) 만약 둘 중에서 하나의 리스트가 먼저 끝나게 되면 나머지 리스트의 값들을 전부 새로운 리스트(sorted)로 복사한다.

(4) 새로운 리스트(sorted)를 원래의 리스트(list)로 옮긴다.

- 합병 정렬 예시

배열에 27, 10, 12, 20, 25, 13, 15, 22이 저장되어 있다고 가정하고 자료를 오름차순으로 정렬해 보자.



- 합병 정렬 시간복잡도

시간 복잡도

분할 : n개 원소를 두 개로 분할 -> $T(n) = O(\log_2 n)$

병합 : 최대 n번의 비교 연산 -> $T(n) = O(n)$

따라서 시간 복잡도는 $T(n) = O(n\log_2 n)$, 이과정은 퀵 정렬과 같아 자세한건 생략함

공간 복잡도 : 원래의 자료 n개에 대해 정렬할 원소를 저장할 2*n개 의 추가 공간 필요

- 합병 정렬 장단점, 특징

장점

안정적인 정렬 방법

데이터의 분포에 영향을 덜 받는다. 즉, 입력 데이터가 무엇이든 간에 정렬되는 시간은 동일하다. ($O(n\log_2 n)$로 동일)

단점

만약 레코드를 배열(Array)로 구성하면, 임시 배열이 필요하다.

제자리 정렬(in-place sorting)이 아니다.

레크드들의 크기가 큰 경우에는 이동 횟수가 많으므로 매우 큰 시간적 낭비를 초래한다.

6. 힙 정렬(Heap Sort)

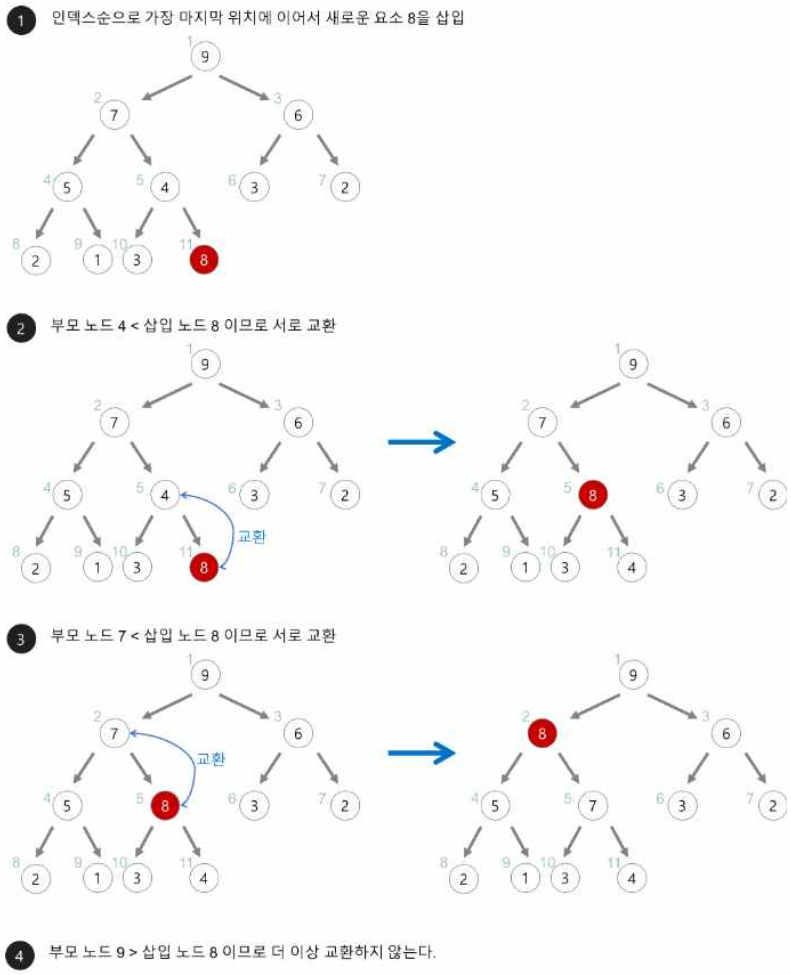
- 힙 정렬(Heap Sort)은 Heap 자료구조를 이용해 정렬하는 방법이다. 최대 히프 에서 원소 개수만큼 삭제 연산을 수행하면 내림차순으로 정렬된 원소를 얻을 수 있고, 최소 히프 에서 원소 개수만큼 삭제 연산을 수행하면 오름차순으로 정렬된 원소를 얻을 수 있다는 성질을 이용한다.
- 동일한 값에 대해 기존의 순서가 유지되지 않는 불안정 정렬이다.

- 힙 정렬 과정

(1) 정렬할 원소에 대해 삽입 연산을 통해 최대 히프 구성
(2) 최대 히프 에서 삭제 연산하여 배열의 비어있는 자리 중 마지막 자리에 저장
(3) 남은 히프를 최대 히프 로 재구성
(4) 공백 히프가 될 때 까지 (2) ~ (3) 반복

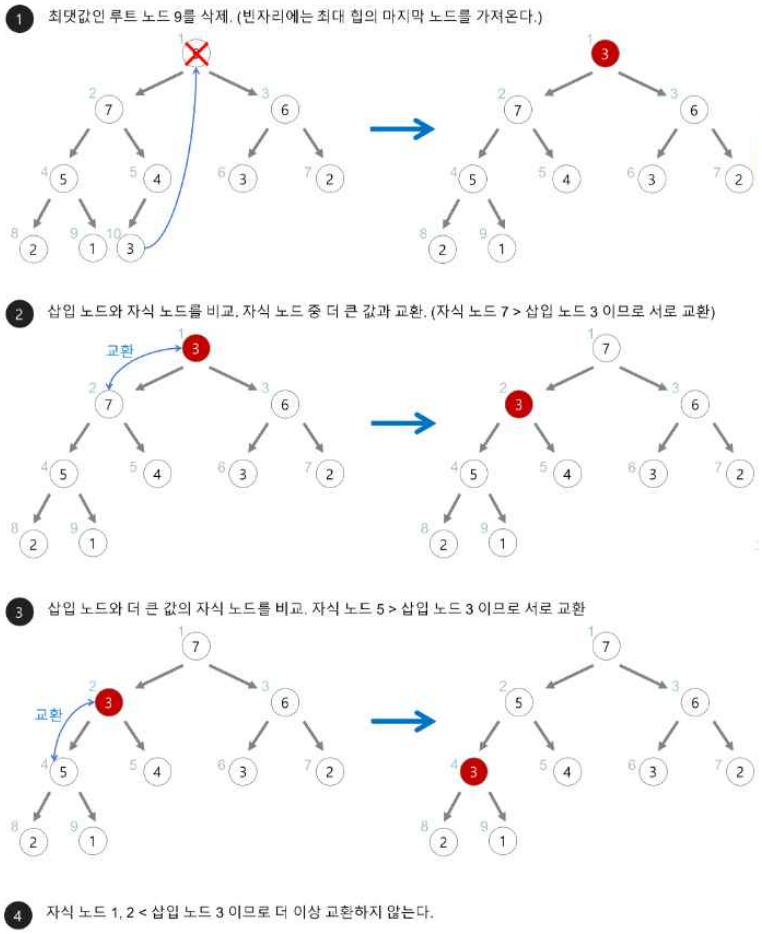
- 힙 정렬 예시 – 삽입

힙에 새로운 요소가 들어오면, 일단 새로운 노드를 힙의 마지막 노드에 이어서 삽입한다.
새로운 노드를 부모 노드들과 교환해서 힙의 성질을 만족시킨다.
아래의 최대 힙(max heap)에 새로운 요소 8을 삽입해보자.



- 힙 정렬 예시 – 삭제

1. 최대 힙에서 최댓값은 루트 노드이므로 루트 노드가 삭제된다.
2. 최대 힙(max heap)에서 삭제 연산은 최댓값을 가진 요소를 삭제하는 것이다.
삭제된 루트 노드에는 힙의 마지막 노드를 가져온다.
3. 힙을 재구성한다.
아래의 최대 힙(max heap)에서 최댓값을 삭제해보자.



위 과정들에서 삭제되는 루트들 즉 최대값이 하나씩 빠져나와 감소되는 순으로 정렬된다.

- 힙 정렬 시간복잡도

히프 재구성 : 힙 트리의 전체 높이가 거의 $\log_2 n$(완전 이진 트리이므로)이므로 하나의 요소를 힙에 삽입하거나 삭제할 때 힙을 재정비하는 시간이 $\log_2 n$만큼 소요된다.
요소의 개수가 n개 이므로 전체적으로 $O(n\log_2 n)$의 시간이 걸린다.
따라서 시간복잡도는 $T(n) = O(n\log_2 n)$

공간 복잡도 : 원래의 자료 n개에 대해 n개의 메모리 와 n개의 원소를 담을 수 있는 히프 추가 필요

- 힙 정렬 장단점, 특징

장점

시간 복잡도가 좋은편

힙 정렬이 가장 유용한 경우는 전체 자료를 정렬하는 것이 아니라 가장 큰 값 몇개만 필요할 때 이다.

단점

데이터의 상태에 따라 같은 O(nlogn) 시간 복잡도 라도 조금 느리다.

불안정 정렬이다.

정렬 알고리즘 복잡도 정리

Sorting Algorithm	공간 복잡도		시간 복잡도		
	최선	최악	최선	평균	최악
Bubble Sort	$O(1)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Heap Sort	$O(1)$	$O(n \times log\, n)$	$O(N \times log\, n)$	$O(n \times log\, n)$	$O(n \times log\, n)$
Insertion Sort	$O(1)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(1)$	$O(n \times log\, n)$	$O(n \times log\, n)$	$O(n \times log\, n)$	$O(n \times log\, n)$
Quick Sort	$O(log\, n)$	$O(n \times log\, n)$	$O(n \times log\, n)$	$O(n \times log\, n)$	$O(n^2)$
Selection Sort	$O(1)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Shell Sort	$O(1)$	$O(n)$			

정렬 알고리즘 특징 및 장단점

버블 정렬

장점 : 인접한 값만 계속해서 비교하는 방식으로 구현이 쉬우며, 코드가 직관적이다. n개 원소에 대해 n개의 메모리를 사용하기에 데이터를 하나씩 정밀 비교가 가능하다.

단점 : 최선이든 최악이든 O(n²) 이라는 시간복잡도를 가진다. n개 원소에 대해 n개의 메모리를 사용하기에 원소의 개수가 많아지면 비교 횟수가 많아져 성능이 저하된다.

선택 정렬

장점 : 정렬을 위한 비교 횟수는 많으나 교환 횟수가 적기에, 교환이 많이 이루어져야 하는 상태에서 효율적으로 사용될 수 있으며, 이에 가장 적합한 자료상태는 역순 정렬이다. 즉, 내림차순이 정렬되어 있는 자료를 오름차순으로 재정렬할 때 적합하다.

단점 : 정렬을 위한 비교 횟수가 많으며, 이미 정렬된 상태에서 소수의 자료가 추가되면 재정렬할 때 최악의 처리 속도를 보인다.

삽입 정렬

장점 : 최선의 경우 O(n)이라는 빠른 효율성을 가지고 있다. 버블정렬의 비교횟수를 줄이기 위해 고안된 정렬이다. 버블정렬의 경우 비교대상의 메모리 값이 정렬되어 있더라도 비교연산을 진행하나, 삽입정렬은 버블정렬의 비교횟수를 줄이고 크기가 적은 데이터 집합을 정렬하는 알고리즘을 작성할 때 효율이 좋다.

단점 : 최악의 경우 $O(n^2)$이라는 시간복잡도를 가지게 된다. 즉, 데이터의 상태와 크기에 따라 성능의 편차가 큰 정렬 방법이다.

퀵 정렬

장점 : 기준값(Pivot)에 의한 분할을 통해 구현하는 정렬 방법으로, 분할 과정에서 $O(\log_2 n)$이라는 시간이 소요되며, 전체적으로 $O(n\log_2 n)$으로 준수한 편이다.

단점 : 기존값에 따라 시간복잡도가 크게 달라진다. 기준값을 이상적인 값으로 선택했다면 $O(n\log_2 n)$의 시간복잡도를 가지지만, 최악의 기준값을 선택할 경우 $O(n^2)$라는 시간복잡도를 갖게 된다. 또한 동일한 요소에 대해 불안정 정렬을 한다.

병합 정렬

장점 : 퀵 정렬과 비슷하게 원본 배열을 절반씩 분할해가면서 정렬하는 정렬법으로써 분할하는 과정에서 $O(\log_2 n)$만큼의 시간이 소요된다. 즉, 최종적으로 보게되면 $O(n\log_2 n)$이 된다. 또한 퀵 정렬과 달리 기준값을 설정하는 과정없이 무조건 절반으로 분할하기에 기준값에 따라 성능이 달라지는 경우가 없다. 따라서 항상 $O(n\log_2 n)$이라는 시간복잡도를 가지게 된다.

단점 : 병합정렬은 임시배열에 원본맵을 계속해서 옮겨주며 정렬을 하는 방식이기에 추가적인 메모리가 필요하다는 점이다. 데이터가 최악인 면을 고려하면 퀵 정렬보다는 병합정렬이 훨씬 빠르기 때문에 병합정렬을 사용하는 것이 많지만, 추가적인 메모리를 할당할 수 없다면 병합정렬을 사용할 수 없기 때문에 퀵을 사용해야 하는 것이다.

힙 정렬

장점 : 추가적인 메모리를 필요로하지 않으면서 항상 $O(n\log_2 n)$의 시간 복잡도를 가진다. 인 정렬 방법 중 가장 효율적인 정렬방법이라 할 수 있다. 퀵 정렬의 경우 효율적이나 최악의 경우 시간이 오래걸린다는 단점이 있으나 힙 정렬의 경우 항상 $O(n\log_2 n)$이 보장된다.

단점 : 이상적인 경우에 퀵정렬과 비교했을 때 똑같이 $O(n\log_2 n)$이 나오긴 하나 실제 시간을 측정하면 퀵정렬보다 느리다고 한다. 즉, 데이터의 상태에 따라서 다른 정렬들에 비해서 조금 느린편이다. 또한, 안정성을 보장받지 못한다.