힙 정렬 (Heap Sort)

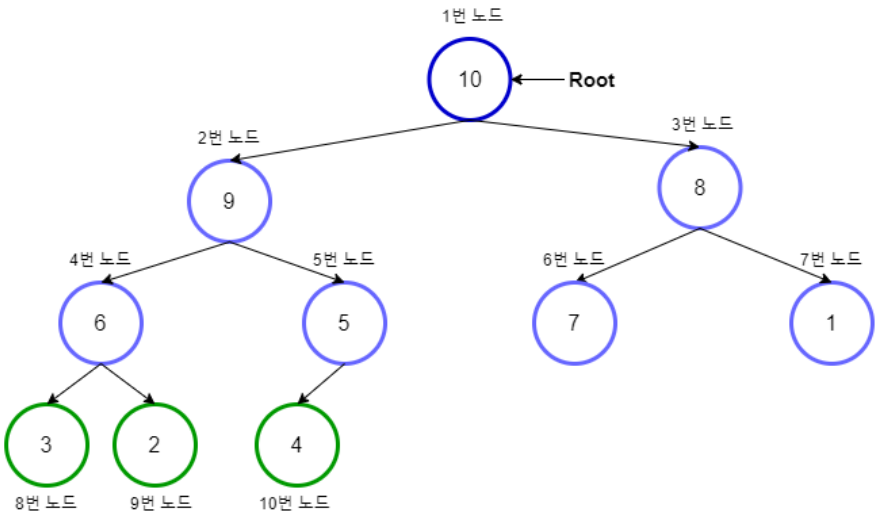
> 퀵 소트, 머지 소트 만큼 빠름
>
> 힙 트리 구조 (Heap Tree Structure)을 이용하는 정렬 방법

힙

Q1. 스택과 큐, 힙

- 개념
  - 완전 이진트리 구조
    - 자식 노드가 2개 이하 임을 보장
    - 중간 노드가 비어있지 않고 빽빽히 차있다
      - 자식노드 기준 왼쪽, 오른쪽 순서로 들어옴
  - 힙을 통해 최대 값과 최소 값을 빠르게 찾고자 함.
  - 우선순위 큐를 위해 만들어진 자료구조
    - 우선 순위 큐
      - 데이터들이 우선순위를 가지고 있다
      - 우선순위가 높은 데이터가 먼저 나온다
      - CPU 작업 스케줄링, 시뮬레이션 시스템에서 사용
  - 일종의 트리로 여러 개의 값들 중에서 최댓값이나 최솟값을 빠르게 찾아내도록 만들어진 자료구조
  - O(log n)
- 종류
  - 최대 힙(max heap)
    - 부모 노드의 키 값이 자식 노드의 키 값보다 크거나 같은 완전 이진트리
    - key(부모 노드) ≥ key(자식 노드)

    - 최소 힙(min heap)
        - 부모 노드의 키 값이 자식 노드의 키 값보다 작거나 같은 완전 이진트리
        - key(부모 노드) ≥ key(자식 노드)
- 특징
    - 힙을 저장하는 표준적인 자료구조는 “배열”
    - 중복 값 허용
    - 반 정렬 상태를 유지
        - 느슨한 정렬 상태
        - 반 정렬 상태라는 것은 트리 구조를 배열에 저장한 것이므로 배열로만 보았을 때는 완전히 정렬된 상태로 보이지 않는 것



Heap 자료구조 Tree 형태로 보기

Heap 자료구조 배열로 보기

- 연산
    - 삽입
        - 힙에 새로운 요소가 들어오면, 일단 새로운 노드를 힙의 마지막 노드에 삽입

        - 새로운 노드를 부모 노드들과 교환
    - 삭제
        - 최대 힙에서 최댓값은 루트 노드이므로 루트 노드가 삭제됨
            - 최대 힙에서 삭제 연산은 최댓값 요소를 삭제하는 것)
        - 삭제된 루트 노드에는 힙의 마지막 노드를 가져옴
        - 힙을 재구성

진행 순서

최대 힙이라 가정

넣을 때

1. 최대(or 최소) 힙 구성
2. 마지막 리프노드 자리에 새로운 원소를 넣고
3. 부모 노드와 비교, 자식이 더 크면(or 작으면) swap
4. 루트노드까지 위 과정 반복

꺼낼 때

1. 최대(or 최소) 힙 구성
2. 루트값과 마지막 리프노드 값 swap
3. 이전 루트값을 꺼냄
4. 루트 노드부터 자식노드보다 작으면 두 자식노드 중 더 큰 노드와 swap
5. n / 2까지 반복

코드

```
for (int i = 1; i<N; i++) {
    int c = i;
    while (c != 0) {
        int root = (c - 1) / 2;
        if (arr[root] < arr[c]) swap(&arr[root], &arr[c]);
```

```
        c = root;
    }
}


for (int i = number - 1; i>=0; i--) {
    swap(&arr[0], &arr[i])

    int root = 0;
    int c = 1;
    while ( c < i ) {
        c = 2 * root + 1;
        if (arr[c] < arr[c+1] && c<(i-1) c++;
        if (arr[root] < arr[c] && c < i) swap(&arr[root], &ar
        root = c;
    }
}
```

}

시간 복잡도

최선	평균	최악
$O(NlogN)$	$O(NlogN)$	$O(NlogN)$

장점

- 병합 정렬과 다르게 별도로 추가적인 배열이 필요하지 않다.
  - 메모리 측면에서 효율적
- 항상 O(n * log n) 보장