알고리즘 : 정렬 탐색 힙 해시테이블 그래프 문자열검색?

탐색

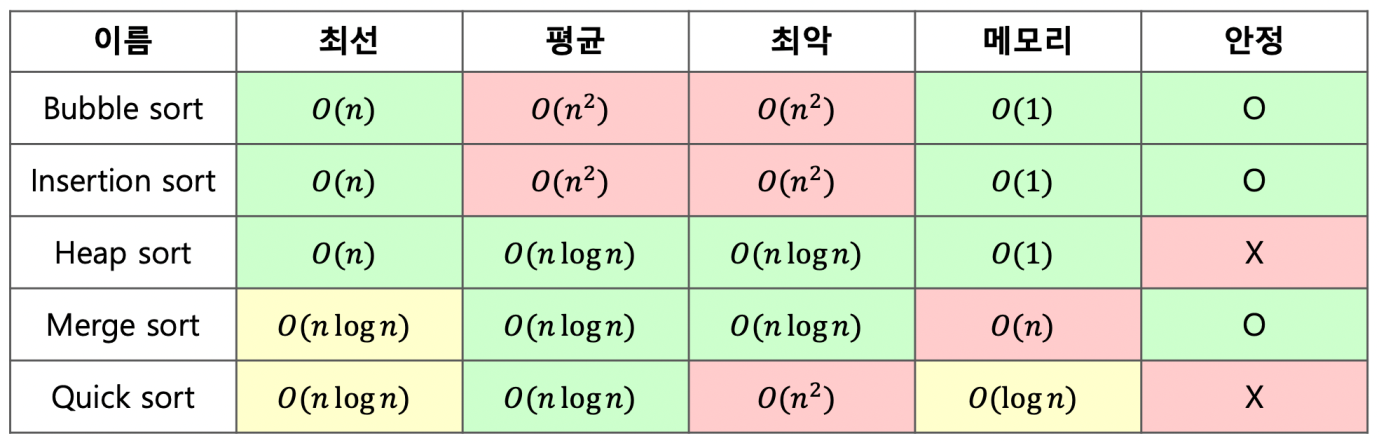
순차탐색: 데이터 no정렬시 앞에서부터 하나씩 살펴봄. 버그적고 쉬우나 비효율적. O(n)

이진탐색: 데이터가 미리 오름차순으로 정렬 되어 있어야한다. 반으로 나누어 한쪽씩 고르는 방식.

중앙 고르고 작으면 왼쪽 크면 오른쪽. 반복. O(log n).

이렇게 이진탐색쓰려면 정렬이 필요함

정렬 : 데이터셋을 정해진 순서로 재배치



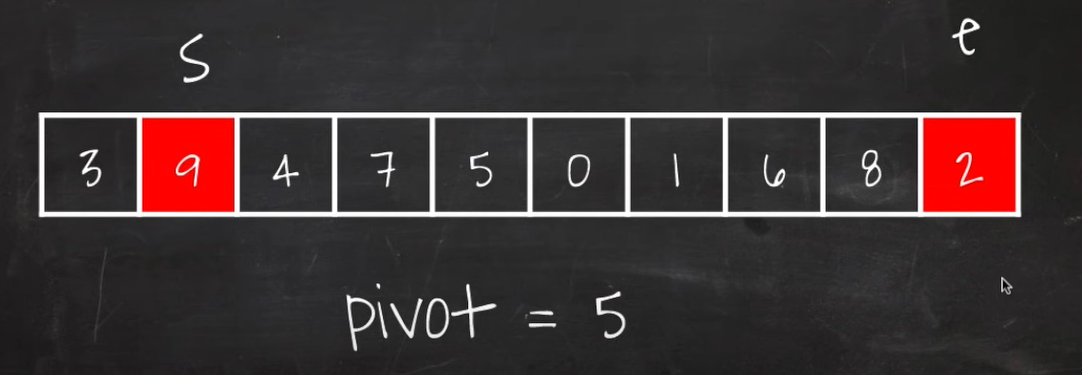
1. 버블정렬: 자주안씀. 앞에서부터 2개씩 비교하면서 큰걸 뒤로 옮김. - 1사이클. 끝날때까지 사이클 무한 반복. 1사이클이 n-1번. 최악은 N번만큼 사이클 돌릴 수 있으므로 **시간복잡도는 O(n²)**

2. 선택정렬: 현재위치에 맞는 자료 찾고 거기로 옮김. -1사이클. 정렬다 될때까지 사이클 반복. 버블정렬만큼 최악은 아니지만 모든데이터를 스캔하므로 최선이든 최악이든 **시간복잡도는 O(n²).**

3. 삽입정렬: 필요한 것만 스캔. 왼쪽부터 하나찝고 앞에꺼랑 비교하면서 작으면 앞으로 땡김. 선택보다 빠르지만 그래도 **시간복잡도는 O(n²).** 최선일때는 전체 한번만 순회해도되니 O(n)

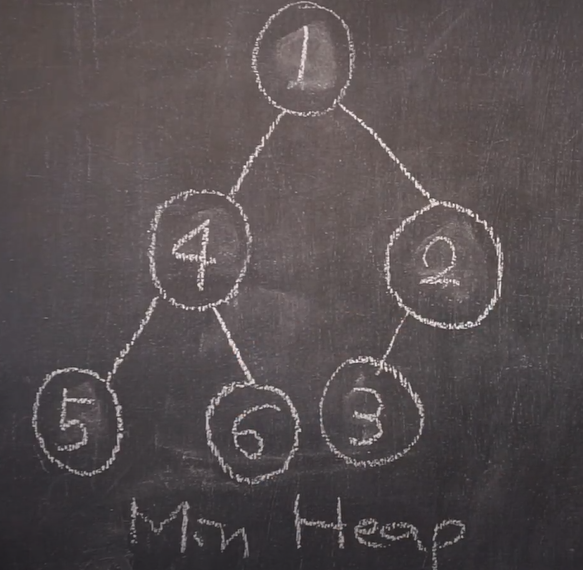
4. 병합정렬: 분할정복알고리즘. 모든 숫자를 다 나눈다음에 병합하는 방식. 분할해서 정렬 후 병합해서 정렬. 재귀적으로 점점 더 크게 병합. n개만큼 log n번 돌리므로 **시간복잡도는 O(n log n).** 하지만 실행시 별도 저장공간 필요. 공간쓸수없을땐 퀵소트 써야.

5. 퀵정렬: 아무거나 값을 하나 잡고 작은애들 왼쪽 큰애들 오른쪽으로. 그리고 작은쪽에서 하나 또 잡고 작은거 왼쪽 큰거 오른쪽으로 쪼개서 정렬. 큰쪽도 마찬가지. **평균적으로 O(n log n)이지만 최악의 경우 O(n²).** n번 파티션 나누는데 나눌때마다 데이터가 절반씩 줄어드니 log n번이라.



젤먼저 기준값(피벗)을 정해야되는데 일반적으로 중간에 있는거 찝음. 양쪽끝에서 작고 큰거끼리 스왑. 분할과 정복 반복수행. 메모리 부족하고(병합정렬 사용불가), 역정렬(최악의경우)일 가능성이 없을 때 사용. 메모리 효율위해 퀵정렬 사용.

$$ 힙-최대값or최소값을 빠르게 구하기 위해 고안된 완전이진트리. 최소힙은 부모노드가 자식노드보다 항상작다. 최대힙은 부모노드가 자식노드보다 항상 크다. 최소힙에서 노드 삽입시 맨 마지막에 넣고 부모힙과 비교해서 작으면 위로 올리면 된다. **삽입제거시 시간복잡도O(log n)**



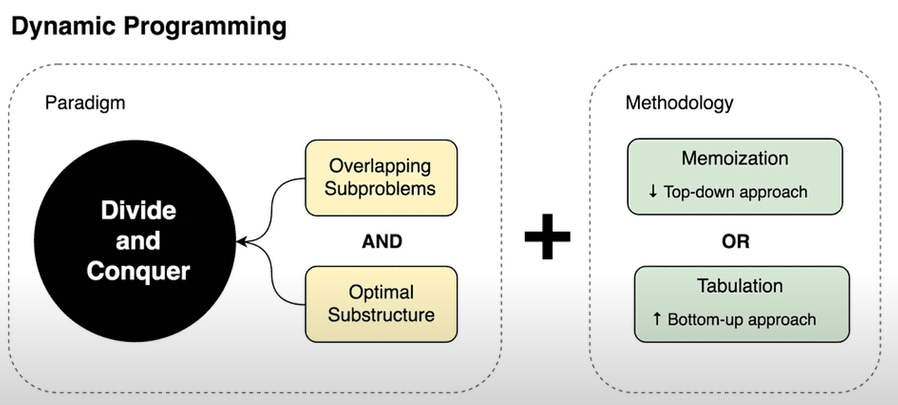
6. 힙정렬: 최악의경우 O(n log n)이고 최대최소값 찾을시 O(1) 퀵보다 평균적으로 느리다.

모든 노드가 힙속성을 만족하도록 재귀적으로 트리구조를 만들어 정렬을 완성.

메모리가 부족하고(병합사용x), 배열이 이미 정렬(역정렬)되어 퀵정렬 사용힘들고 원소 위치가 바바껴 될 때 사용 가능.

분할정복법(divide and conquer) : 문제를 분할해서 합침

동적계획법(dynamic programming)



(피보나치수열같이 재귀적) 문제에서 중복부분문제 발생 or 최적부분구조가 있을 때

Memorization : 탑다운으로 푸는데 이미 저장한건 바로 리턴하자.

Or Tabulation : 그럴필요없이 바텀업으로 저장한걸 쌓아올라가면서 풀자

재귀알고리즘(recursion) : 자신이 자신을 호출하는 함수. 일반적으로 반복문보다 느리지만 코드가 깔끔해짐. 함수를 반복호출해 시간복잡도가 늘어나고 스택오버플로우 발생가능.

Ex) 피보나치 수열, 하노이의 탑.

꼬리재귀 : 반환부에 연산 없에서 스택오버플로우 방지

탐욕알고리즘 : 근사적 방법으로 최적해 구해서 그 순간 최적이라 생각하는걸 고르는 것. 지역적으로는 최적이지만 최종적으로 최적이라는 보장 x. 조건이 맞을 때만 사용가능.