Algorithm
Sorting Comparison

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Introduction

본 문서는 컴퓨터 과학에서 사용되는 정렬 알고리즘들에 대한 큰 특징들을 살펴보며 각 알고리즘 별로 공간 및 시간 복잡도에 대해서 서로 비교를 해봅니다.

Sorting Algorithms

정렬 알고리즘은 크게 보면 형태나 방법에 따라 아래와 같이 분류될 수 있습니다.

안전 정렬 (stable)	불안전 정렬 (not stable)
같은 값(key)의 위치가 정렬 과정에서 바뀌지 않는 것	같은 값(key)의 위치가 정렬 과정에서 바뀌는 것

내부 정렬 (Internal sorting)	외부 정렬 (External sorting)
데이터의 크기가 주 기억장소 용량보다 적을 경우 기억장소를 활용하여 정렬	데이터의 크기가 주 기억장소 용량보다 클 경우 외부 기억장치를 사용하여 정렬

Sorting Kinds

버블정렬(Bubble Sort)

특성	설명
형태	안정 정렬, 내부 정렬
정렬 방법	이중 루프를 돌며 모든 인접값끼리 비교
장점	코드 구현이 쉽고 직관적이다.
단점	모든 인접값을 비교하므로 굉장히 비효율적
공간 복잡도	O(1)
시간 복잡도	최악 : O(n^2), 최선 : Ω(n), 평균 : Θ(n^2)

Sorting Kinds

선택정렬(Selection Sort)

특성	설명
형태	불안정 정렬, 내부 정렬
정렬 방법	데이터 중 최소값을 선택한 후 제일 앞자리와 바꾸며 정렬
장점	코드 구현이 쉽고 데이터가 적을때 비교적 효율적, 데이터 크기와 상관 없이 일정한 시간 복잡도를 가짐
단점	버블 정렬보다는 일반적으로 2-3배 빠르지만 그래도 느림
공간 복잡도	O(1)
시간 복잡도	최악 : O(n^2), 최선 : Ω(n^2), 평균 : Θ(n^2)

Sorting Kinds

삽입정렬(Insertion Sort)

특성	설명
형태	안정 정렬, 내부 정렬
정렬 방법	특정 데이터 선택 후 별도 공간에 삽입하면서 데이터 비교
장점	최선의 경우는 O(n)의 빠른 효율성을 지님
단점	데이터의 상태와 크기에 따라서 최악의 경우 O(n^2) 속도를 보임
공간 복잡도	O(1)
시간 복잡도	최악 : O(n^2), 최선 : Ω(n), 평균 : Θ(n^2)

Sorting Kinds

퀵 정렬(Quick Sort)

특성	설명
형태	불안정 정렬, 내부 정렬
정렬 방법	기준값(Pivot)을 정하고 분할, 정복을 통하여 정렬
장점	실제 사용 환경에서 가장 빠르다고 알려져 많이 사용됨
단점	최악의 경우는 n^2의 속도를 가짐, Pivot 선택에 따른 편차가 큼
공간 복잡도	O(nlogn)
시간 복잡도	최악 : O(n^2), 최선 : Ω(nlogn), 평균 : Θ(nlogn)

Sorting Kinds

병합정렬(Merge Sort)

특성	설명
형태	안정 정렬, 외부 정렬
정렬 방법	데이터를 분할 정복을 활용하여 정렬, 대표적인 분할정복 정렬
장점	퀵 정렬과 비슷한 O(logn)의 시간이 걸림, Pivot 선택 과정이 없으므로 항상 일정한 속도를 가짐, 가장 많이 사용되는 알고리즘
단점	추가적인 메모리가 많이 필요함
공간 복잡도	O(n)
시간 복잡도	최악 : O(nlogn), 최선 : Ω(nlogn), 평균 : Θ(nlogn)

Sorting Kinds

힙 정렬(Heap Sort)

특성	설명
형태	불안정 정렬, 내부 정렬
정렬 방법	최대힙 트리를 사용한 정렬 알고리즘
장점	추가적인 공간을 필요로하지 않으면서 항상 O(nlogn)의 시간복잡도를 가짐
단점	데이터 상태에 따라서 퀵정렬 등 다른 정렬법보다 조금 느린편이다. 또한 안정성도 보장받지 못하는 단점이 있음
공간 복잡도	O(1)
시간 복잡도	최악 : O(nlogn), 최선 : Ω(nlogn), 평균 : Θ(nlogn)

Sorting Kinds

쉘 정렬(Shell Sort)

특성	설명
형태	불안정 정렬, 내부 정렬
정렬 방법	삽입 정렬의 단점을 보완해서 만든 정렬 알고리즘, 일정한 간격의 그룹을 만든 후 삽입 정렬을 통해 정렬
장점	삽입 정렬보다 훨씬 뛰어난 성능을 가짐
단점	간격에 따라서 시간복잡도가 결정되며 간격 설정에 따라서 성능이 안 좋을 수 있음
공간 복잡도	O(1)
시간 복잡도	최악 : O(nlogn), 최선 : Ω(nlogn), 평균 : Θ(nlogn)

Sorting Kinds

기수정렬(Radix Sort)

특성	설명
형태	안정 정렬, 내부 정렬
정렬 방법	데이터의 자릿수(Radix) 기준으로 정렬하는 알고리즘
장점	데이터 비교를 하지 않고 O(n)의 속도를 가짐
단점	버킷이라는 추가적인 메모리가 필요함. 또한 데이터 타입이 일정한 경우에만 사용 가능
공간 복잡도	O(n+k)
시간 복잡도	최악 : O(nk), 최선 : Ω(nk), 평균 : Θ(nk)

Sorting Kinds

카운팅정렬(Counting Sort)

특성	설명
형태	안정 정렬, 내부 정렬
정렬 방법	데이터 비교 없이 데이터의 최대값의 수만큼의 카운팅 배열을 만들고 카운트를 통하여 정렬하는 알고리즘
장점	데이터 비교를 하지 않고 O(n)의 속도를 가짐
단점	기본적으로 숫자의 최대 수만큼의 별도의 추가 공간이 필요함. 데이터에 따라서 비효율적인 공간 낭비가 있을 수 있음.
공간 복잡도	O(k)
시간 복잡도	최악 : O(n+k), 최선 : Ω(n+k), 평균 : Θ(n+k)

Space Complexity

지금까지 살펴본 정렬 알고리즘들의 공간 복잡도를 다시 한번 비교 해보겠습니다.

특성	공간복잡도
버블정렬	O(1)
선택정렬	O(1)
삽입정렬	O(1)
퀵 정렬	O(nlogn)
병합정렬	O(n)
힙 정렬	O(1)
쉘 정렬	O(1)
기수정렬	O(n+k)
카운팅정렬	O(k)

Time Complexity

마찬가지로 지금까지 살펴본 정렬 알고리즘들의 시간 복잡도를 다시 한번 비교해보겠습니다.

특성	최악(Worst)	평균(Average)	최선(Best)
버블정렬	O(n^2)	O(n^2)	O(n^2)
선택정렬	O(n^2)	O(n^2)	O(n^2)
삽입정렬	O(n^2)	O(n^2)	O(n)
퀵 정렬	O(n^2)	O(nlogn)	O(nlogn)
병합정렬	O(nlogn)	O(nlogn)	O(nlogn)
힙 정렬	O(nlogn)	O(nlogn)	O(nlogn)
쉘 정렬	O(n^2)	O(nk) = O(n)	O(nlogn)
기수정렬	O(nk) = O(n)	O(nk) = O(n)	O(nk) = O(n)
카운팅정렬	O(n+k) = O(n)	O(n+k) = O(n)	O(n+k) = O(n)

References

[1] 정렬 알고리즘 비교 : https://ratsgo.github.io/data%20structure&algorithm/2017/10/19/sort/

[2] [정렬] 정렬별 장단점 및 시간복잡도 : https://yabmoons.tistory.com/250

[3] [C언어] 여러가지 정렬 속도 비교(정렬의 시간복잡도) : https://coding-factory.tistory.com/138

[4] 정렬 알고리즘의 종류 및 속도 비교 : https://moyaria.tistory.com/1399

[5] 정렬 알고리즘(Sorting Algorithm) | 종류 및 속도 비교 : https://m.blog.naver.com/go2604/221824669394

References

[6] 정렬 알고리즘의 비교 : https://wonjayk.tistory.com/225

[7] 정렬 알고리즘 비교 : https://defacto-standard.tistory.com/37

[8] 정렬 알고리즘 (Sorting Algorithm) : https://2heedu.tistory.com/160

[9] 정렬 알고리즘 정리 : https://medium.com/@wooder2050/정렬-알고리즘-정리-54349222f432

[10] 제 2 강. 알고리즘과 알고리즘의 성능
 : http://dblab.duksung.ac.kr/ds/pdf/Chap02.pdf

References

[11] [Algorithm] Time Complexity : https://velog.io/@junyong92/TIL-Time-Complexity

[12] [알고리즘] 점근적 표기법 : https://satisfactoryplace.tistory.com/70

Thank you!