SORTING ALGORITHM

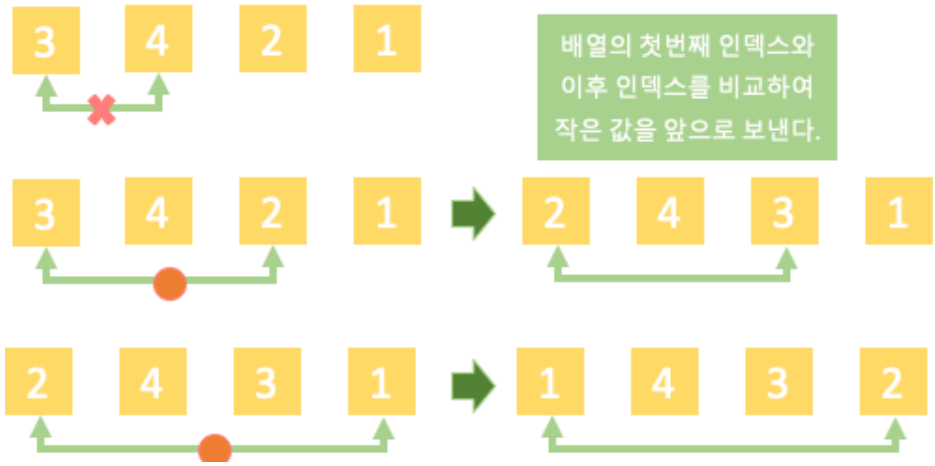
김희조 (20181020)

성신여자대학교 정보시스템공학과

Introduction – 정렬 알고리즘 소개

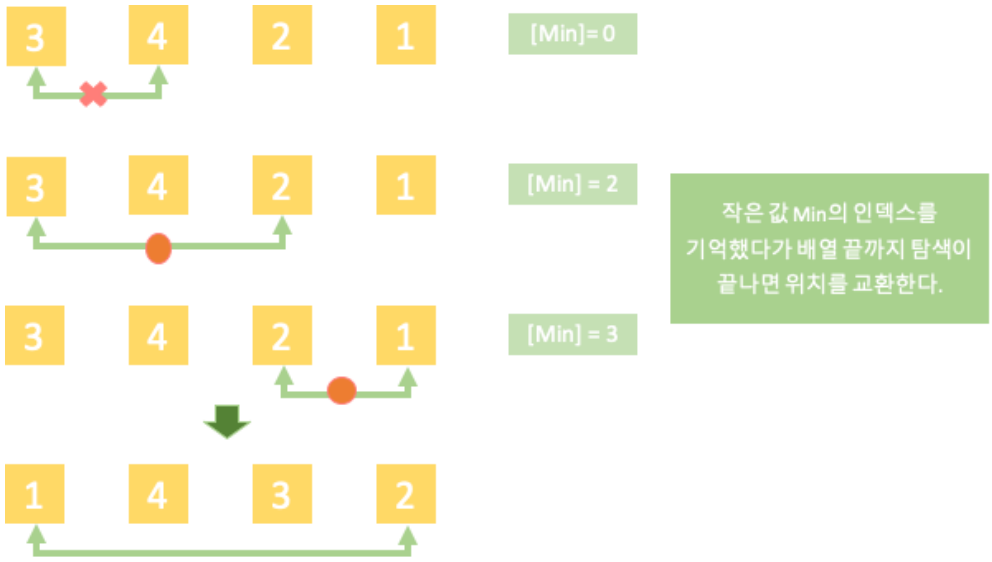
입력된 n개의 정수를 비내림차순으로 정렬한다.

교환 정렬[1] (Exchange Sort)



첫 번째 인덱스(1)와 이후 인덱스들(2~n)을 전부 순차적으로 비교하여 더 작은 값을 갖는 인덱스를 첫 번째 인덱스에 배치한다. 이후 이미 정렬한 인덱스를 제외하고 나머지들을 같은 방법으로 정렬하는 알고리즘이다.

선택 정렬[2] (Selection Sort)



배열에 있는 모든 인덱스 값 중에서 가장 작은 값을 첫 번째 인덱스 값과 교환한다. 이후 이미 정렬한 인덱스를 제외하고 가장 작은 값을 다음 인덱스에 배치하는 정렬 알고리즘이다. 작은 값의 인덱스를 기억했다가 마지막에 교환한다.

교환 정렬은 비교할 때마다 더 작은 값을 앞쪽 인덱스에 배치하는 데 비해 선택 정렬은 가장 작은 값과 한번만 교환하는 점에서 차이를 둔다. 따라서 교환 정렬과 시간복잡도가 같지만, 레코드의 저장 횟수가 더 적다.

---

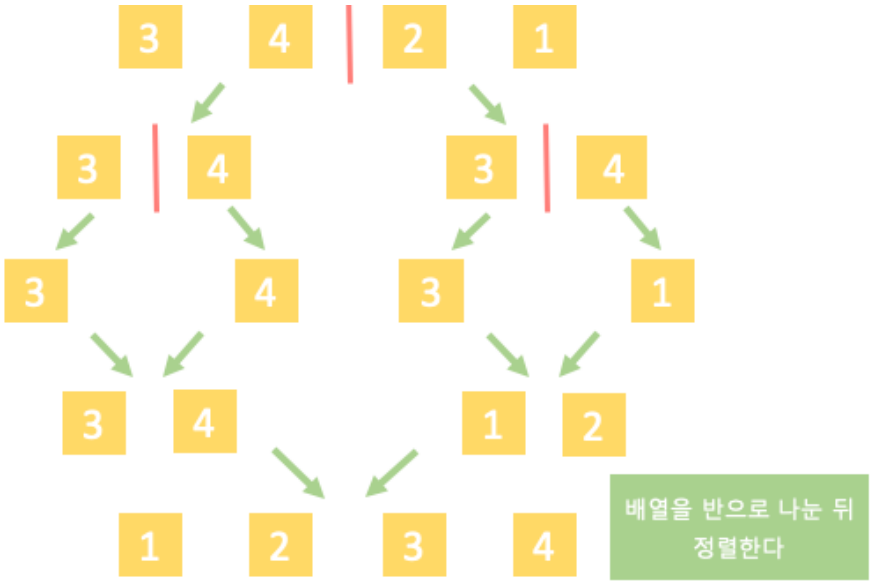
[1] 알고리즘 기초 7pg

[2] 알고리즘 기초 275pg

삽입 정렬[3] (Insertion Sort)

정렬된 배열에 값을 추가하여 정렬하는 알고리즘이다. 0개의 정렬된 배열에 주어진 배열의 첫 번째 인덱스를 추가하여 정렬한다. 이후 인덱스를 하나씩 추가하여 정렬하는 방법이다.

합병 정렬[4] (Merge Sort)



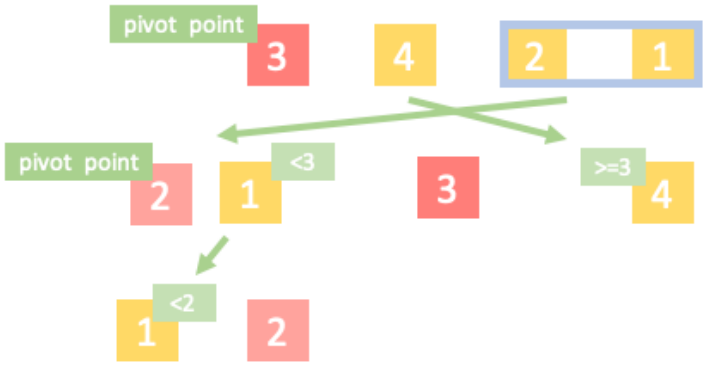
주어진 배열을 인덱스 개수가 1이 될 때까지 반으로 분할한 다음 각각 따로 정렬한 뒤 합병을 반복하는 알고리즘이다.

---

[3] 알고리즘 기초 272pg

[4] 알고리즘 기초 54pg

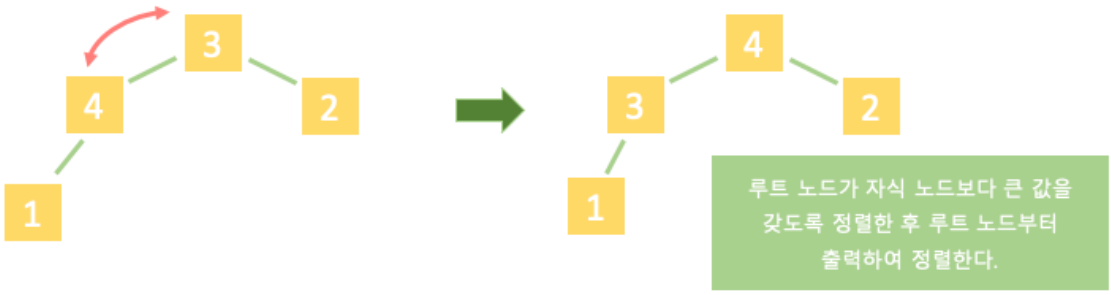
빠른 정렬[5] (Quick Sort)



기준 키(pivot item)를 기준으로 작은 값은 기준 키의 인덱스(pivot point) 앞으로, 크거나 같은 값은 pivot point 뒤로 가도록 하여 기준점보다 작거나 큰 값으로 분리해가며 정렬하는 알고리즘이다.

합병 정렬은 배열을 최소한으로 분할한 뒤 정렬하는 반면, 빠른 정렬은 기준점보다 크거나 작은 값으로 분리해가며 정렬한다는 점에서 차이를 둔다.

힙 정렬[6] (Heap Sort)



주어진 배열을 완전한 이진 트리에 정렬하여 할당한 후 루트 노드에서 키를 제거해가며 정렬하는 알고리즘이다. 해당 알고리즘은 제자리 정렬이기에 힙에 저장하는 방법이 아닌 배열로 힙을 구현하는 방법을 사용한다. 따라서 추가적인 메모리가 필요하지 않는다.

---

[5] 알고리즘 기초 63pg

[6] 알고리즘 기초 287pg

컴퓨터 사양

컴퓨터 기종

Laptop (Macbook pro)

CPU 사양

Intel core i9-9880H 2.3GHz 8-core, turbo boost 4.8GKz

RAM 크기

16 GB

RAM 속도

2400 MHz

운영체제

Windows 10 Home

개발 소프트웨어

Microsoft Visual Studio 2019

Result - Table

임의의 정수 n개를 정렬하는 알고리즘들의 실행 결과이다. 아래 테이블들의 값은 각 알고리즘마다 n개의 정수를 정렬하는 데 걸린 시간(sec)이다. n개 정수의 정렬을 10번씩 반복하여 평균을 구해 오차범위를 줄였다.

교환·선택·삽입 정렬의 경우 n=1000000의 경우 프로그램을 돌리는 데 너무 오래 걸리므로 제외하였다.

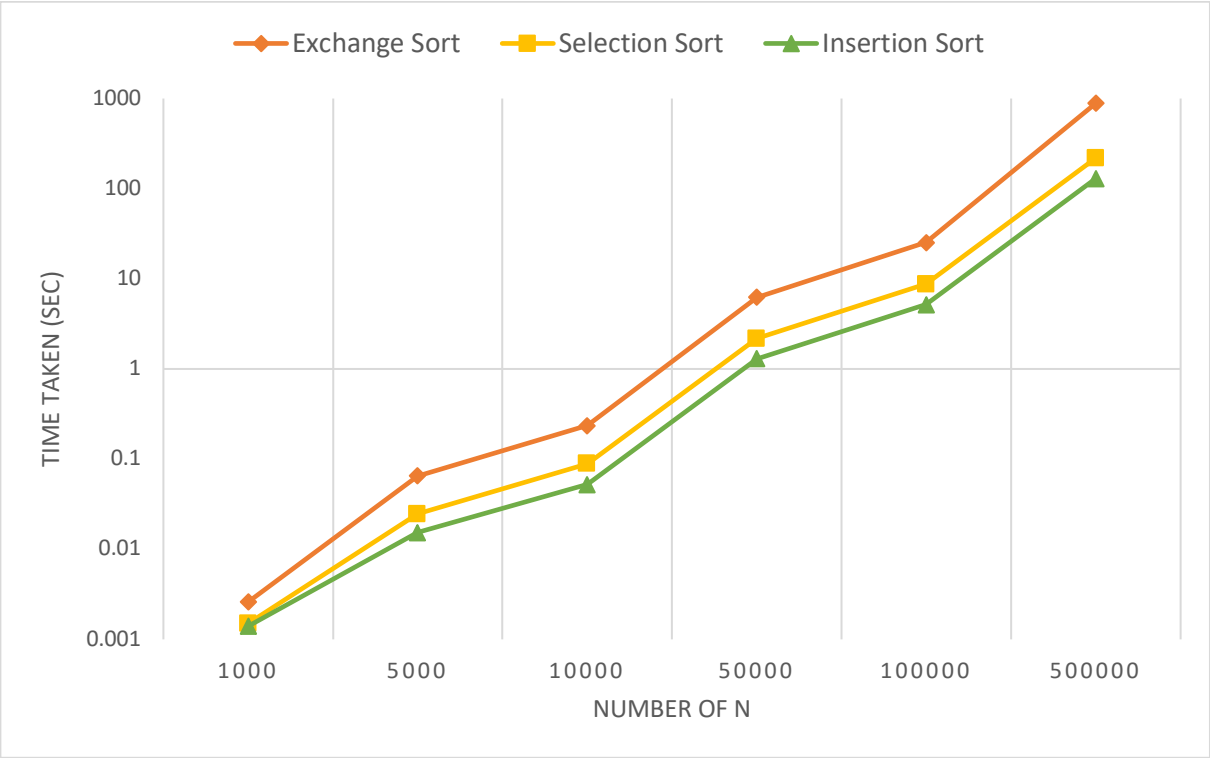
Table 1: $O(n^2)$

n	1000	5000	10000	50000	100000	500000
Exchange Sort	0.0026	0.0650	0.2348	6.2543	25.2027	886.2750
Selection Sort	0.0015	0.0246	0.0888	2.1769	8.7288	218.8800
Insertion Sort	0.0014	0.0153	0.0523	1.2960	5.1762	129.7280

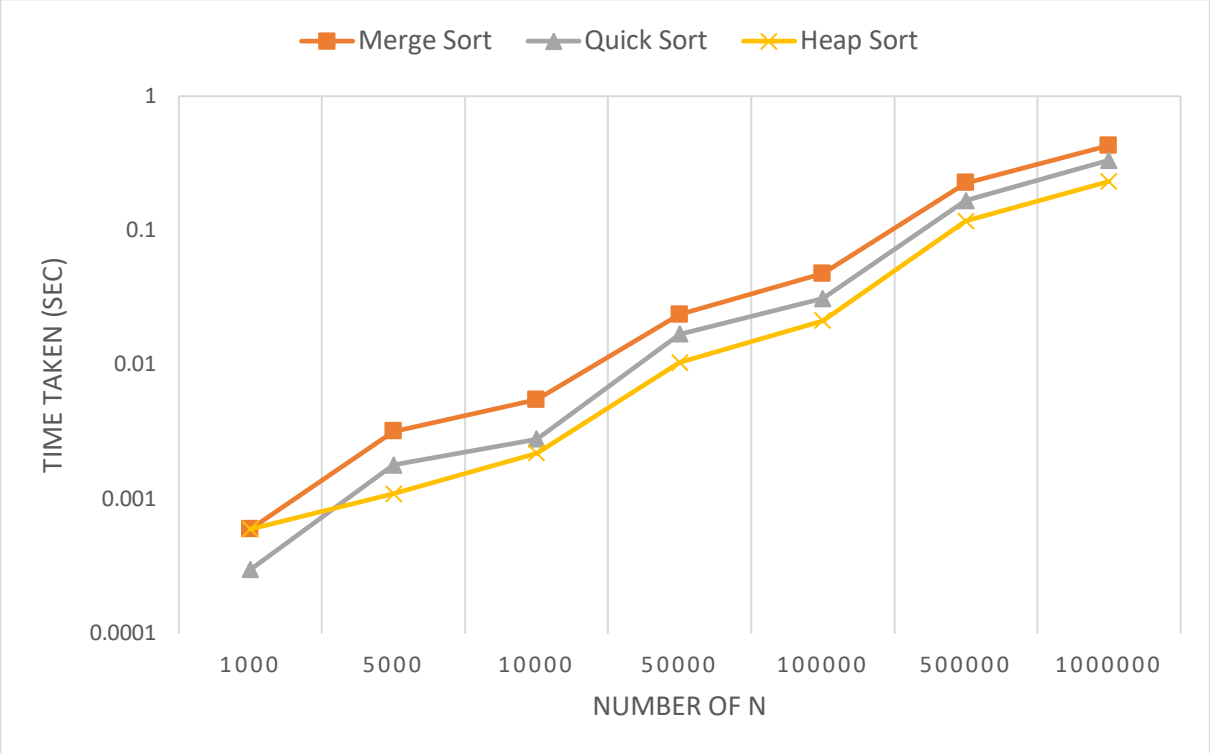
Table 2: O(nlogn)

n	1000	5000	10000	50000	100000	500000	1000000
Merge Sort	0.0006	0.0032	0.0055	0.0237	0.0478	0.2254	0.4298
Quick Sort	0.0003	0.0018	0.0028	0.017	0.0312	0.1667	0.3341
Heap Sort	0.0006	0.0011	0.0022	0.0104	0.0214	0.1175	0.2324

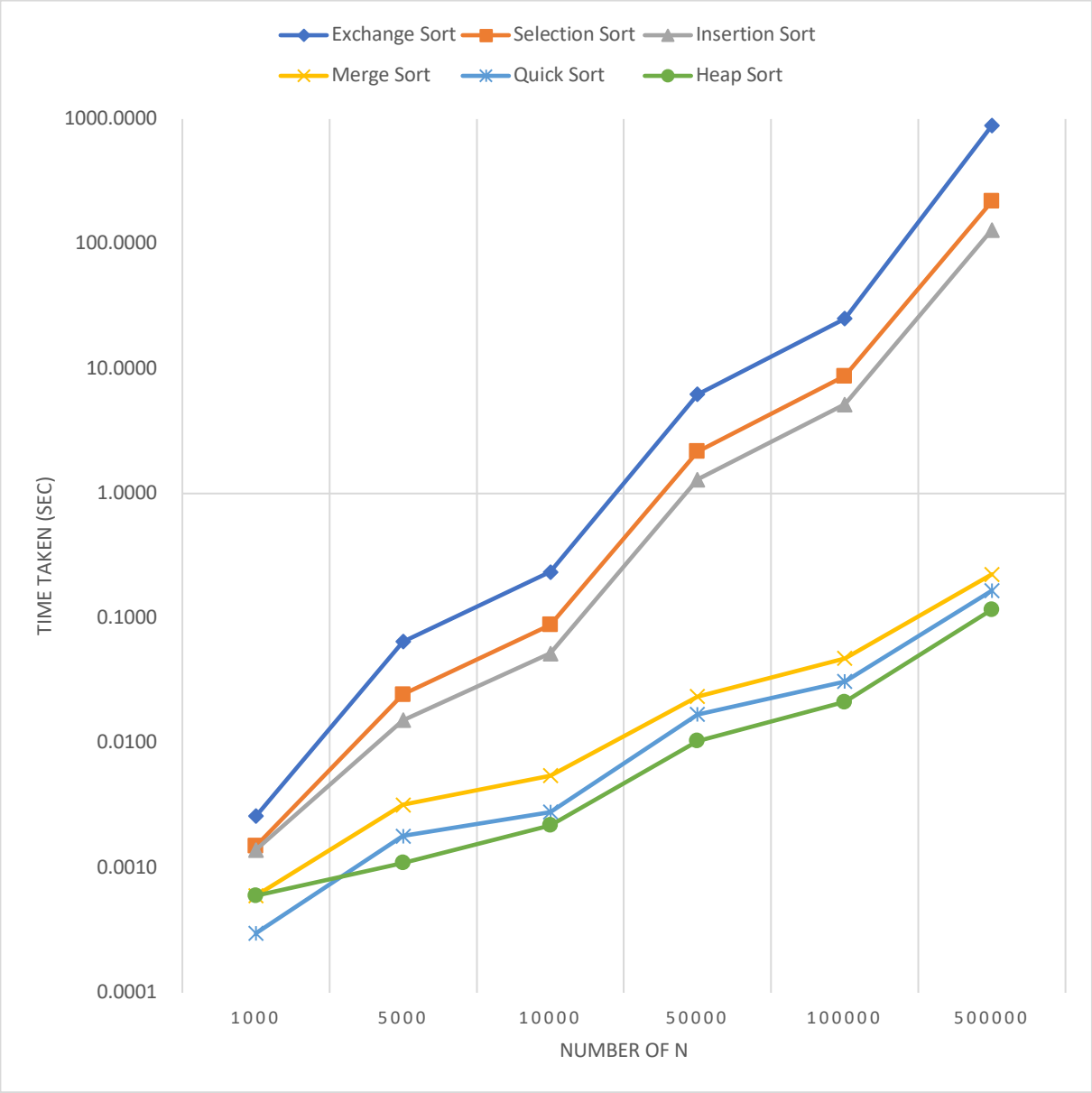
Analysis – Graph



Graph 1: $O(n^2)$



Graph 2: O(nlogn)



Graph 3: Sorting Algorithms

절반의 알고리즘이 n=1000000일 때의 데이터가 없어서 해당 데이터들을 제외하여 작성했다.

Conclusion – 분석 결과

6개의 분석 알고리즘을 프로그래밍하여 분석한 결과, 알고리즘들이 데이터의 수가 적을 때는 실행 시간의 차이가 미미했던 반면, 데이터 수가 10,000개를 넘어가면서부터 큰 격차를 벌리기 시작했다. 그중에서도 급속히 느려진 알고리즘 3개와 실행속도가 비슷한 알고리즘 3개로 분류되었다. 느려진 알고리즘은 전부 시간 복잡도 $O(n^2)$를 갖는 알고리즘이었다.

O(nlogn)을 갖는 알고리즘들의 경우 실행 시간이 전부 근접했다. n=1000일 땐 빠른 정렬이 가장 빨리 실행되었지만, 데이터 값이 증가함에 따라 힙 정렬보다 느리게 실행되었다. 이론적으로 빠른 정렬이 가장 빠르다는 것을 고려하면 이는 프로그래밍 방법의 문제이거나 컴퓨터 성능의 문제라고 판단된다.

SortClass 프로그램을 통해 여러 정렬 알고리즘 간의 실행 시간을 비교해보았다.

삽입 정렬같이 입력된 데이터의 크기에 따라 성능에 편차가 심한 알고리즘이 존재하는 반면, 힙 정렬같이 최소 O(nlogn)를 보장하는 알고리즘도 있다. 따라서 최적의 결과를 내기 위해서는 특정 알고리즘만을 사용할 것이 아니라 하나 이상의 알고리즘을 섞어 사용하는 Hybrid Sorting Algorithm을 구현하여 사용할 필요가 있다.

또한, 같은 시간복잡도를 갖는 알고리즘이라도 공간복잡도를 고려하거나, 더 오래 걸리더라도 구현이 쉬운 알고리즘을 사용하고자 한다면 최적의 알고리즘은 얼마든지 변할 수 있다.

시간 복잡도만으로 어떤 알고리즘이 더 낫고 나쁜지를 정할 수는 없다. 따라서, 모든 사항을 고려하여 상황에 최적인 알고리즘을 선택해야 한다.

Reference

"Learn DS & Algorithms." *Programiz*, Parewa Labs Pvt. Ltd, www.programiz.com/dsa.

Neapolitan, Richard E. *Foundations of Algorithms*. Translated by 도경구 , Jones & Bartlett Learning, 2015.

천인국 , and 최영규 . *DATA STRUCTURES USING C++*. 생능출판, 2016.