CSE3081-03 알고리즘 설계와 분석

MP2 : Master of Sorting

Report

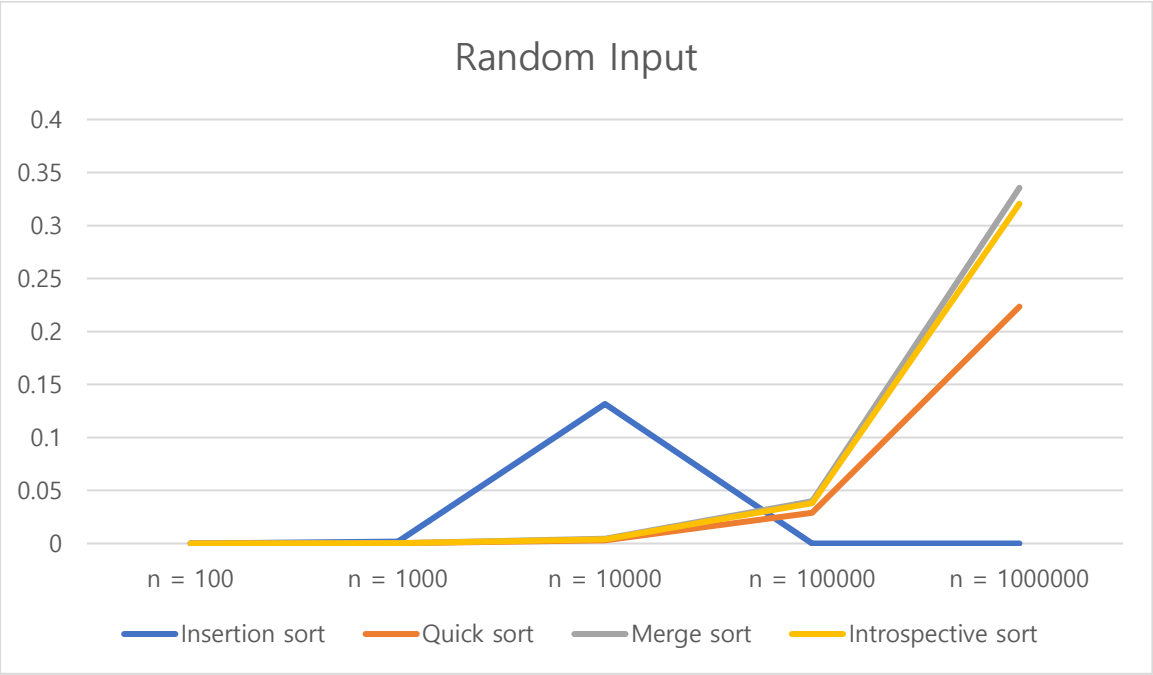
20181294 임승섭

1. Performance of four algorithm

1 - 1. Random input

각 케이스별로 5개의 test case를 만들어 평균값을 계산하였다.

n	Insertion sort	Quick sort	Merge sort	Introspective sort
100	0.000026	0.000018	0.000033	0.000040
1000	0.001796	0.000235	0.000366	0.000374
10000	0.131655	0.003085	0.004386	0.003996
100000	7.632432	0.029045	0.039822	0.037914
1000000	측정 불가	0.223450	0.335540	0.320541

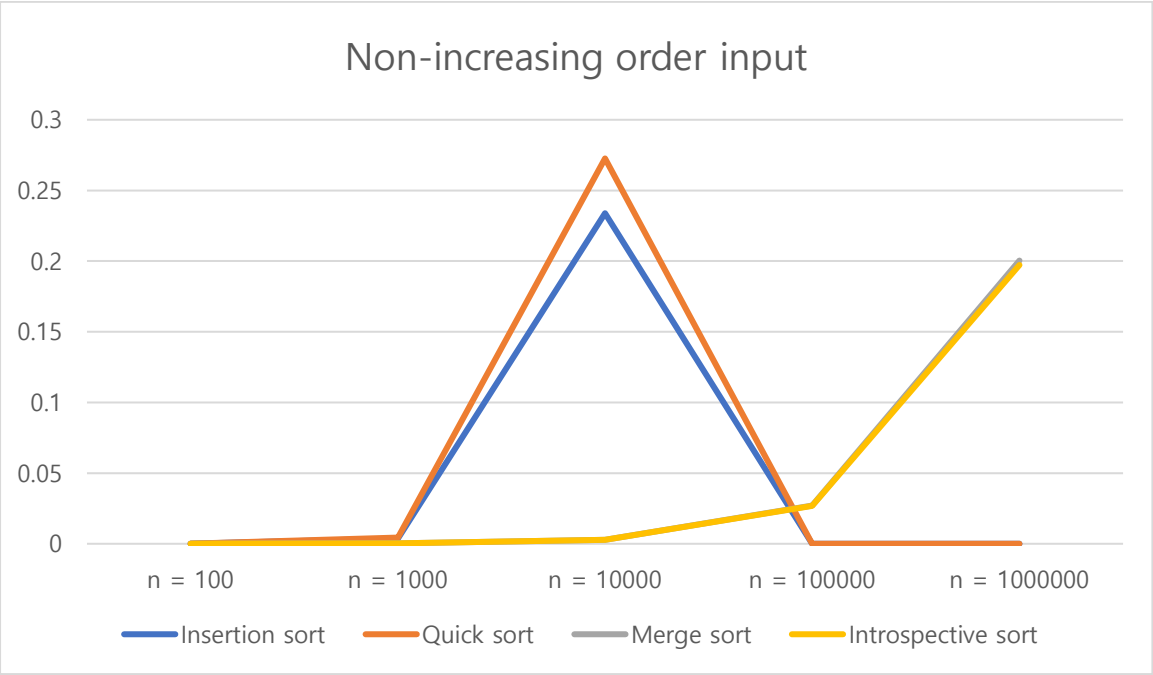


Insertion sort의 경우 n = 100000 이상일 경우 다른 케이스에 비해 압도적으로 시간이 오래 걸리기 때문에 그래프로 나타냈을 때 여러 케이스들을 비교하기에 좋지 않다. 보다 확실한 비교를 위해 그래프 상에서는 n = 100000 이상일 때 Insertion sort의 결과값을 0으로 지정하였다.

1 - 2. Non-increasing order input

n	Insertion sort	Quick sort	Merge sort	Introspective sort
100	0.000045	0.000056	0.000026	0.000057
1000	0.003776	0.004327	0.000230	0.000255
10000	0.233864	0.272625	0.002759	0.002735
100000	15.30393	17.82265	0.027014	0.026859

1000000	(측정 불가)	(측정 불가)	0.200383	0.197238



위와 마찬가지로 n = 100000 이상일 때 Insertion sort와 Quick sort의 결과값을 0으로 지정하고 그래프를 생성하였다.

2. Description of Algorithm 4

Algorithm 4에는 Introspective sorting 방법을 사용하였다.

1. 소팅해야 하는 구간의 길이가 16보다 작을 때는 insertion sort를 진행한다. 구간의 길이가 크지 않은 경우에는 재귀함수를 호출해가면서 partition 하는 것보다 곧바로 insertion sort를 해버리는 것이 더 효율적이다.

2. Median of 3 방법을 이용한 QuickSort를 사용한다. 일반적으로 사용하는 quicksort는 pivot을 맨 오른쪽 값을 사용하지만, Median of 3 방법을 이용하여 구간의 가장 왼쪽, 가운데, 오른쪽 세 값들 중 중간값을 pivot으로 사용하였다. 가장 오른쪽 값이 너무 크거나 너무 작아서 partition 하는 데 비효율적일 때 아주 유용하게 이용되는 기능이다.

3. 재귀함수를 진행할 때 depth가 log2(n)을 넘을 시, merge sort를 진행한다. 지나치게 depth가 깊어진다는 것은 partition이 잘 안된다는 뜻이고, worst case에 가까워질수록 quick sort는 O(n^2)

의 시간복잡도를 가지기 때문에 그 전에 merge sort를 진행하는 것이 더 효율적이다.

3. About Experiment

3 - 1. Experiment environment

CPU : 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz

RAM : 16.0GB

OS : Windows 10 Home. 21h2

컴파일 및 실행 : cspro

3 - 2. Experiment setup

n(소팅해야 하는 원소의 개수)를 100부터 시작하여 1000000까지 실행하였다.

random input의 경우 for loop 안에 rand() 함수를 사용하여 원소들을 생성하였고, non-increasing ordr input의 경우 for loop를 이용해서 n부터 1까지 출력하도록 만들었다.

측정 시간에는 파일을 읽거나 파일에 쓰는 시간은 따로 포함하지 않았다. 실제 소팅 함수의 실행 시간만을 측정하였다. clock() 함수를 이용하였다.

3 - 3. My comment

우선, quick sort의 시간복잡도가 worst case와 average case에서 확실히 다르다는 점을 직접 볼 수 있었다. 수식으로 증명만 하고 넘어가는 것보다 직접 시간을 눈으로 확인하니까 훨씬 인상깊게 확인할 수 있었다. 또한, algorithm 4에 사용한 Introspective sort가 case의 종류에 상관없이 평균적으로 좋은 효율을 보여주는 점 또한 직접 확인할 수 있어서 좋은 경험이었다. 하지만 다른 sorting과 엄청난 차이를 보이는 sorting 기법을 구현하고 싶었지만 효율성 차이가 눈에 보일만큼 크지 않다는 점에 약간의 아쉬움이 남는다.