CSE3080-01 알고리즘설계와분석

mp2 – Master of sorting 보고서

20191172 함승우

1. 개발환경

과제 2는 VScode를 이용한 교내 cspro 리눅스 서버 환경에서 개발 되었고, testcasegen.cc 파일을 컴파일하여 random하게 나열된 testcase와, 자체 제작한 non-increasing order로 테스트를 진행했다.

과제를 진행한 추가적인 구동환경은

RAM : 8기가 바이트

CPU: intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz

System; 64bit

2. Sorting Method들끼리 비교

우선, testcase의 최대 개수는 최대 1000000개로 통일을 했다.

- **Insertion sort**

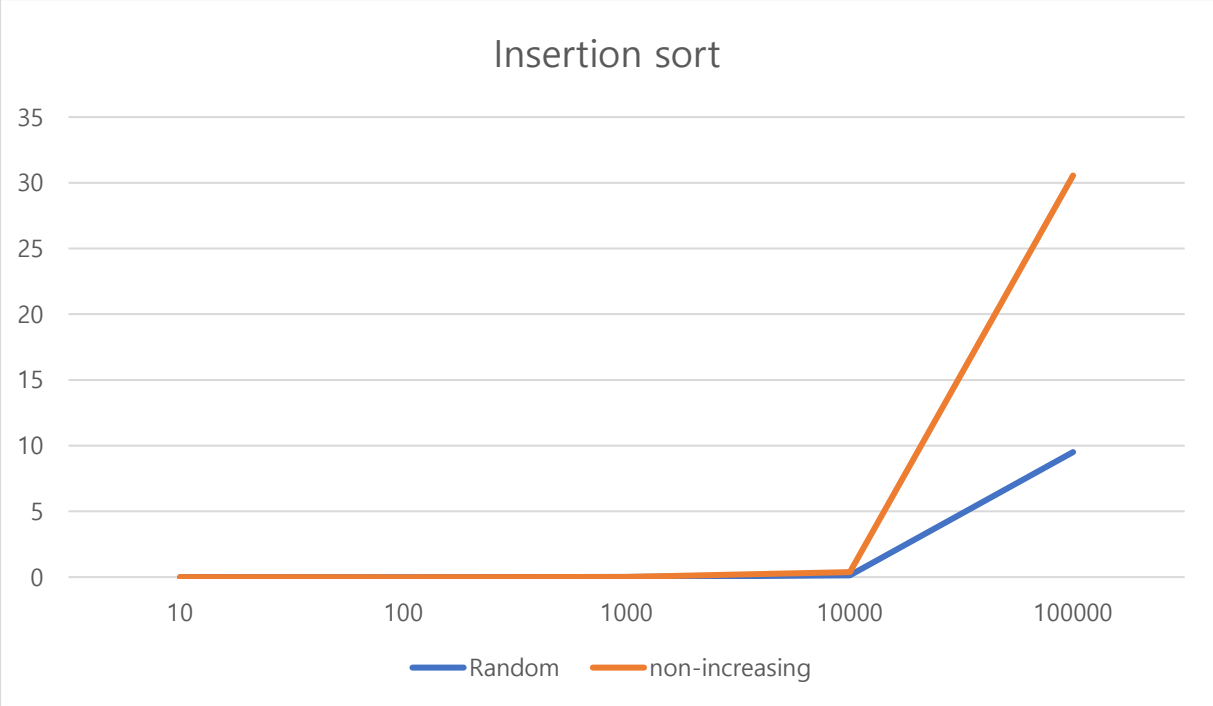
첫번째로, testcasegen으로 random한 배열 10, 100, 1000, 10000, 100000까지 test 해보았다.

n	10	100	1000	10000	100000
Random	0.000004	0.000031	0.002414	0.132572	9.511966

두번째로, for문을 사용하여 특정 n부터 1까지 출력하는 non-increasing testcase를 만들고, 이를 test 해보았다.

n	10	100	1000	10000	100000
non-increasing	0.000005	0.000052	0.004683	0.239383	21.043785

Graph(Random vs Non increasing 비교)



My comments

삽입 정렬의 경우에, 최선의 경우인 이미 제대로 sorting 되었을 때, 최선의 경우인 시간 복잡도 O(n)을 가지지만, 최악의 경우인 이미 거꾸로(역순으로 sorting)되었을 때, 시간 복잡도 O(N^2)을 가진다. worst case는 1 + 2 + 3 + … + (n-1)이지만, average case인 경우에는 1/2(1 + 2 + 3+ … (n-1))이다. 물론 둘 다, 시간 복잡도는 O(N^2)임은 변하지 않지만, 표에 있는 데이터를 통해서 2배 가까이 되는 값들을 확인할 수 있다.

-Quick sort

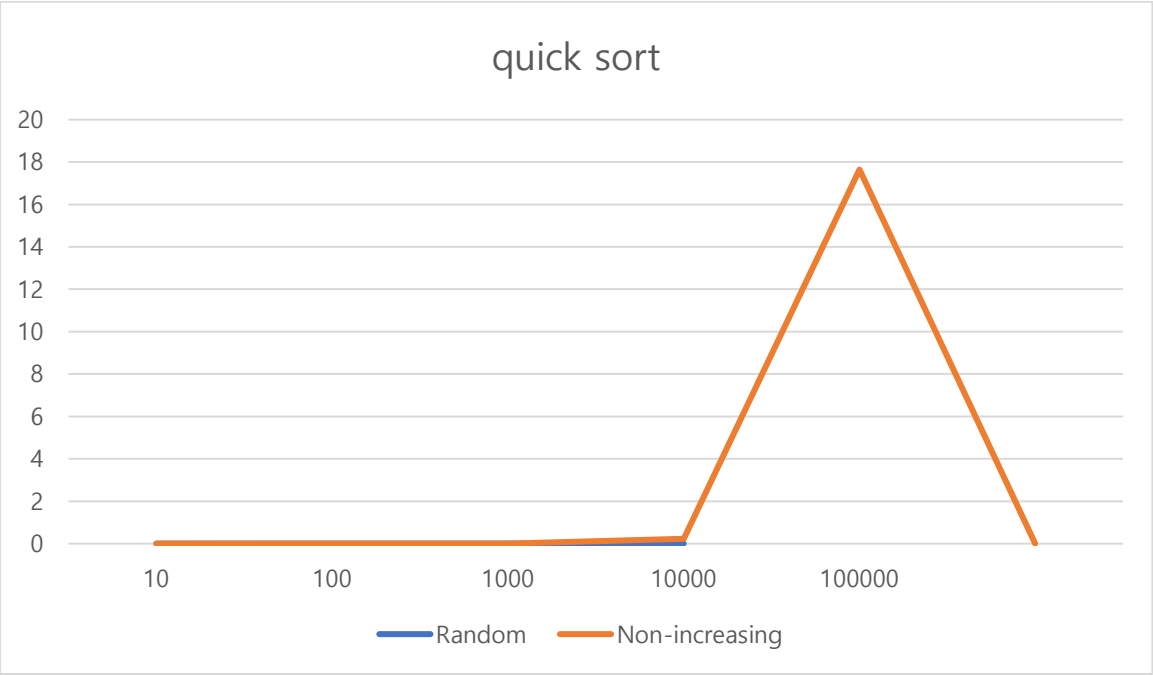
첫번째로, testcasegen으로 random한 배열 10, 100, 1000, 10000, 100000, 1000000까지 test 해보았다.

n	10	100	1000	10000	100000	1000000
Random	0.000004	0.000021	0.000234	0.003063	0.020914	0.179209

두번째로, for문을 사용하여 특정 n부터 1까지 출력하는 non-increasing testcase를 만들고, 이를

test 해보았다.

n	10	100	1000	10000	100000	1000000
non-increasing	0.000004	0.000053	0.004482	0.226825	17.644689	x



My comments

quick sort는 하나의 pivot을 기준으로 divide – conquer solving을 하는 sorting 프로그램이다. 구현한 quick sort는 맨 오른쪽 값이 pivot이 되는 quick-sort 프로그램이다. quick-sort의 경우, O(nlogn)의 시간 복잡도를 가지지만, 최악의 경우는 맨 오른쪽의 pivot이 항상 큰 값일 때이어야 한다. Divide and conquer solving을 할 때면, 다시 pivot의 왼쪽 부분도 quick-sort 해야 할 때 그 왼쪽 부분에서의 맨 오른쪽 값도 계속 제일 큰 값이 될 때가 최악의 경우이다. 이러한 경우는, 쉽게 말하자면, 거꾸로 sorting되어 있거나 제대로 sorting 된 경우이다. 시간 복잡도는 O(N^2)에 해당되고, 따라서, 문제의 입력 값으로 준 non-increasing file이 Random inputfile보다 더 오래 걸리고, 1000000의 경우, non-increasing file의 연산이 일상생활에서 확인이 어려울 만큼 오래 걸렸다.

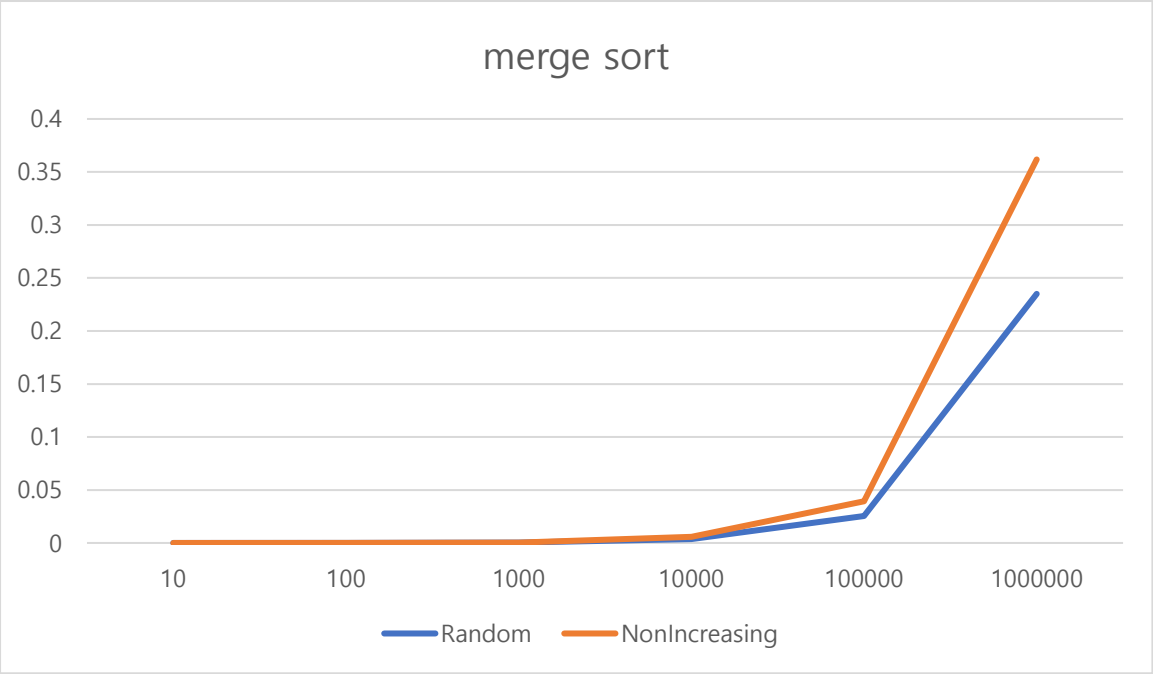
- **merge sort**

첫번째로, testcasegen으로 random한 배열 10, 100, 1000, 10000, 100000, 1000000까지 test 해보았다.

n	10	100	1000	10000	100000	1000000
Random	0.000005	0.000028	0.000299	0.003780	0.025330	0.235000

- 두번째로, for문을 사용하여 특정 n부터 1까지 출력하는 non-increasing testcase를 만들고, 이를 test 해보았다.

n	10	100	1000	10000	100000	1000000
non-increasing	0.000005	0.000017	0.000169	0.002156	0.014085	0.126809



My comments

merge-sort는 임시 배열(temp) 배열이 필요해, 메모리적 관점으로 보았을 때 손해라고 볼 수 있다. Merge sort는 quick-sort처럼 특수한 경우 worst case의 시간 복잡도가 달라지는 것이 아니라, 순환 호출의 깊이만큼 합병 후 연산, 즉 특수한 case에서 worst case가 달라지는 것이 아니다. 따라서, quick sort에서 worst case였던 거꾸로 sorting된 결과도 큰 차이는 없음을 알 수 있다.

- **Fastest sort**

앞서 살펴보았을 때, input size가 작았을 때, quick sort를 사용하는 것은 비효율적이며, insertion 삽입 정렬을 통해서 보다 더 효율적인 sorting 방법을 사용할 수 있다. 또한, quick sort의 pivot을 두 개 사용하여 quick sort를 사용하였다.

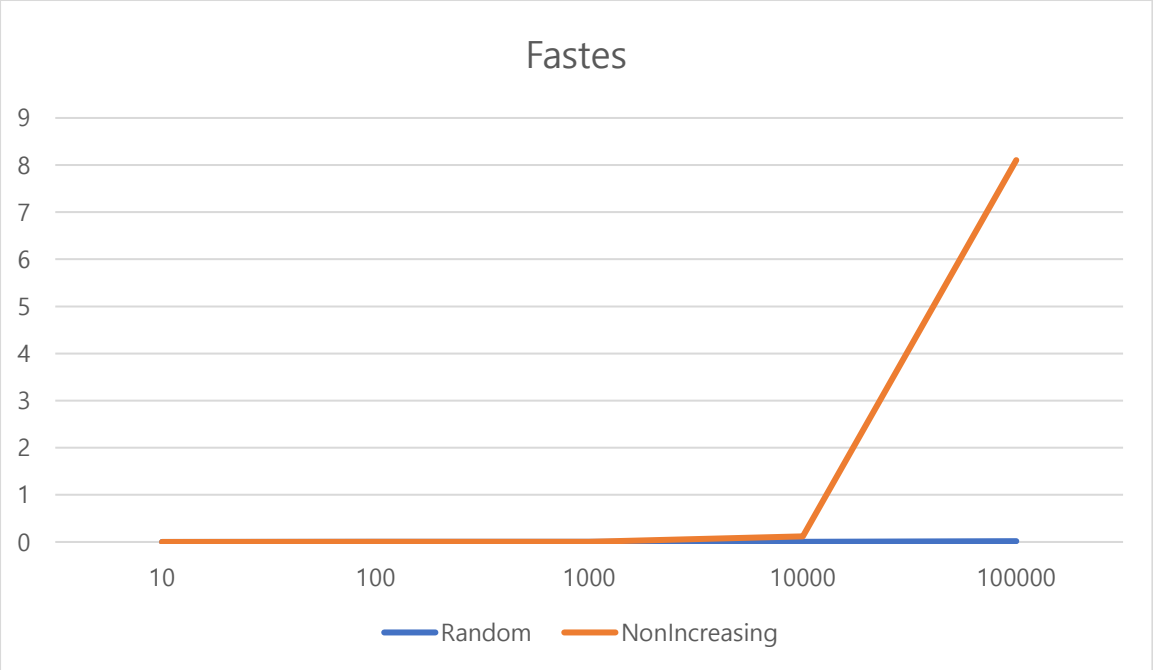
n	10	100	1000	10000	100000
Random	0.000003	0.000018	0.000235	0.002593	0.016767

- 두번째로, for문을 사용하여 특정 n부터 1까지 출력하는 non-increasing testcase를 만들고, 이를 test 해보았다.

n	10	100	1000	10000	100000
non-increasing	0.000004	0.000034	0.002130	0.116764	8.087734

quick sort를 사용한 것과 마찬가지이기 때문에 Non-increasing은 좀 오래 걸린 경향이 있다.

하지만, pivot 두 개를 사용하였다.



- **두 개의 pivot을 사용한 Algorithm 설명**

우선 주어진 배열의 부분이 크기가 작은 16이하라면, 비교적 더 효과적인 Insertion sort 함수를 호출하여 사용합니다. 그렇지 않으면 DualPivotQuickSort함수를 호출하여 두 개의 pivot을 가지는 quick sort로 문제를 해결합니다. 두 개의 pivot을 사용하여 partition한다는 것은 파티션을 세 개를 가진다는 것을 의미합니다.

배열의 가장 왼쪽 항목과 오른쪽 항목의 크기를 항상 유지한 채로 진행한다. 그 후 세 가지의 변수를 이용하여 작은 pivot의 왼쪽, 큰 pivot의 오른쪽, pivot 사이의 공간에 항목들을 삽입하고, 이를 quick-sort와 마찬가지로 Divide and Conquer를 진행한다.

하지만, 본인이 간과한 게 있었다. 이 Dual-pivot quick sort는 두 개의 pivot을 사용한 quicksort에 지나지 않는다. 즉, 최악의 경우인, 제대로 sorting 되어 있거나, 거꾸로 sorting 되어있는 경우에서는 효율적인 알고리즘이 되지는 못한다. 하지만, 많은 random한 값이 주어졌을 때, 이 random한 값들을 quick sort보다 더 빠르게 진행할 수 있다는 점에서 의미가 있다.