Design and Analysis of Algorithms HW2 Report

20180492 김범준

0. 개발 환경 및 테스트 환경

과제 2는 VScode의 WSL ubuntu 원격 서버에서 개발되었고, 개발이 완료된 이후에는 교내 cspro 리눅스 서버 환경에서 테스트케이스들이 테스트되었다. 테스트케이스들에서 입력되는 배열은 random order와 non-increasing order로 정렬되어 있는 경우가 존재하였다. random order는 최대 10만 개의 입력을 받았고, non-increasing order는 최대 100만 개의 입력을 받았다.

1. 알고리즘 1, 2, 3, 4의 성능 비교

1) selection sort

과제 2의 첫 번째 알고리즘은 selection sort를 선택하여 구현하였다. 첫 번째 루프가 배열의 크기 -1 만큼 실행되고, 중첩된 두 번째 루프가 첫 번째 루프에서 선택된 인덱스 + 1부터 배열의 마지막 인덱스까지 실행된다. 따라서, 해당 정렬 알고리즘의 시간복잡도는 O(n^2)이다. 각 테스트 케이스별로 소요된 시간은 다음과 같다.

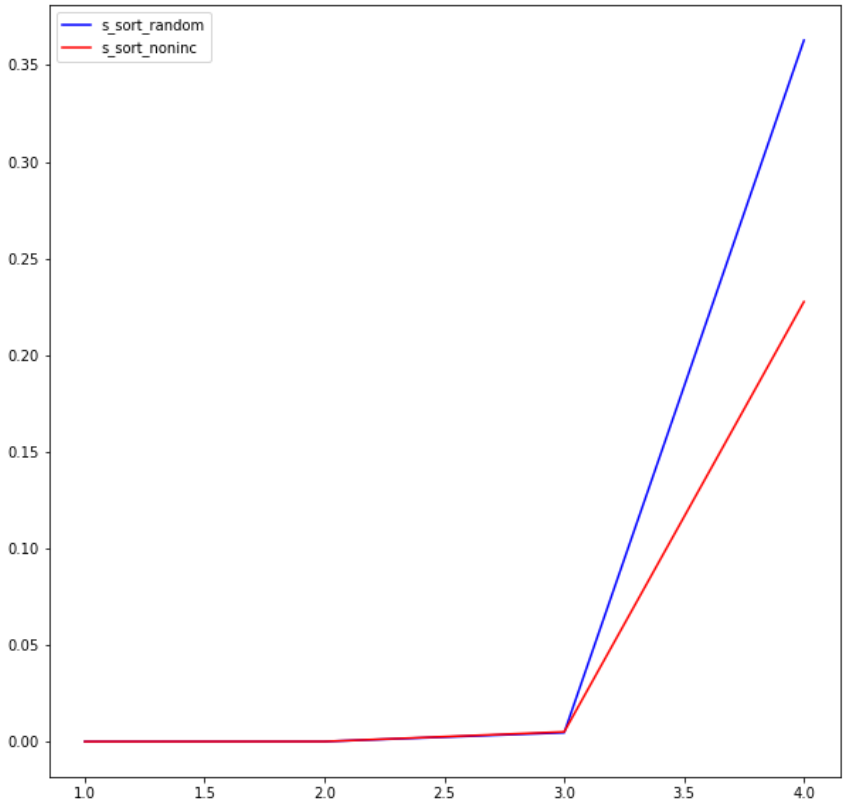
-random order

size 10	size 100	size 1000	size 10000	size 100000
0.000002	0.000053	0.004634	0.362901	35.259525

-non-increasing order

size 10	size 100	size 1000	size 10000	size 100000
0.000003	0.000054	0.005048	0.227603	22.890886

소요된 시간을 그래프로 시각화한 것은 다음과 같다.



위 그래프의 x축에서 1, 2, 3, 4는 각각 input1, input2, input3, input4로 입력 개수는 10, 100, 1000, 10000개이다. y축은 시간복잡도를 나타낸다. 100000개의 입력부터는 실행시간이 20초 이상이 걸려 제외하였다. 파란색 그래프는 random order, 빨간색 그래프는 non-increasing order의 입력으로 주어진 경우의 시간복잡도이다. 10000개의 입력이 들어온 경우에는 non-increasing order의 시간복잡도가 조금 더 낮았지만, 나머지 경우에서 유의미한 차이를 보이지는 않았다.

2) quick sort

과제 2의 두 번째 알고리즘은 quick sort를 사용하여 구현하였다. 강의 시간에 배운 것처럼, quick sort의 average-case 시간복잡도는 O(nlogn), worst-case 시간복잡도는 O(n^2)이다. 여기서, random order의 입력이 들어온 경우가 average-case 시간복잡도를 측정한 경우, non-increasing order의 입력이 들어온 경우가 worst-case 시간복잡도를 측정한 경우라고 할 수 있다. non-increasing order의 경우 파티션을 하면 한 쪽으로만 수들이 몰리기 때문이다. 각 테스트케이스별로 소요된 시간은 다음과 같다.

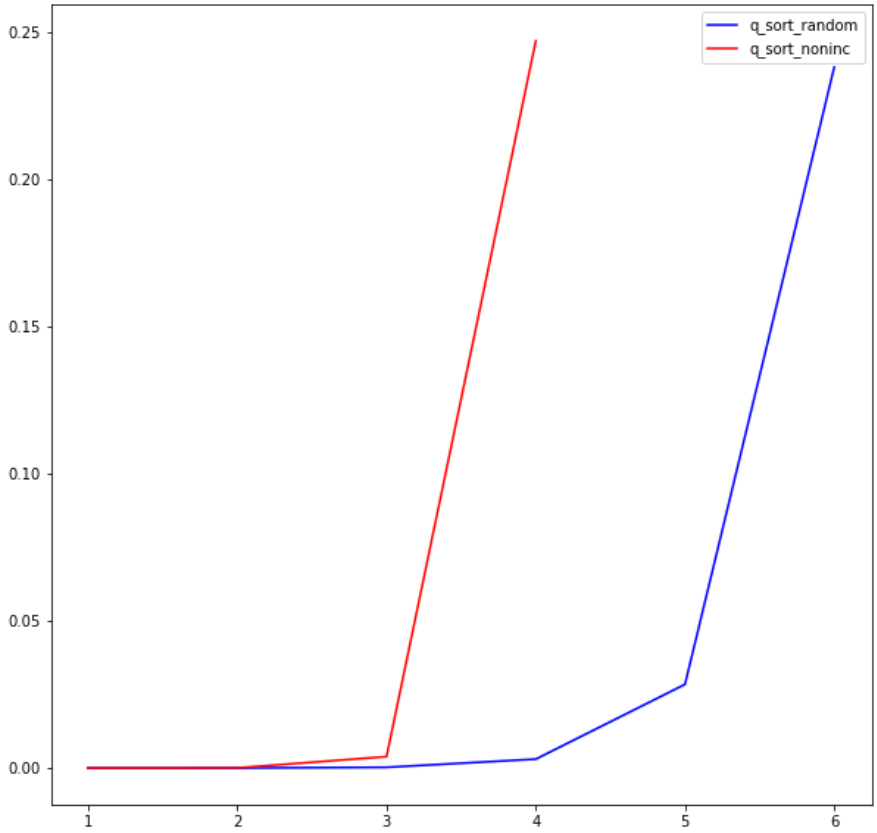
-random order

size 10	size 100	size 1000	size 10000	size 100000	size 1000000
0.000005	0.000019	0.000250	0.003028	0.028459	0.238349

-non-increasing order

size 10	size 100	size 1000	size 10000	size 100000
0.000003	0.000053	0.003873	0.247282	18.346040

소요된 시간을 그래프로 시각화한 것은 다음과 같다.



위의 그래프에서 non-increasing order 입력의 quick sort의 시간 복잡도는 15초 이상이 걸려 제외하였다. 추가적으로, x축의 1, 2, 3, 4, 5, 6은 각각 input1, input2, input3, input4, input5, input6으로 입력 개수는 10개, 100개, 1000개, 10000개, 100000개, 1000000개이다. 그래프에서 알 수 있듯이, quick sort의 worst case 시간복잡도의 추이는 O(n^2)의 시간복잡도를 가지는 selection sort와 비슷하다는 것을 알 수 있다.

3) merge sort

과제 2의 세 번째 알고리즘은 merge sort를 선택하여 구현하였다. 강의 시간에 배운 것처럼,

merge sort의 시간 복잡도는 average case와 worst case 모두 O(nlogn)이다. 하지만, average case의 경우에 일반적으로 상수항이 quick sort보다 크므로, 대부분의 경우에 quick sort보다 소요되는 시간이 조금 더 걸리게 된다. 각 테스트케이스별로 소요된 시간은 다음과 같다.

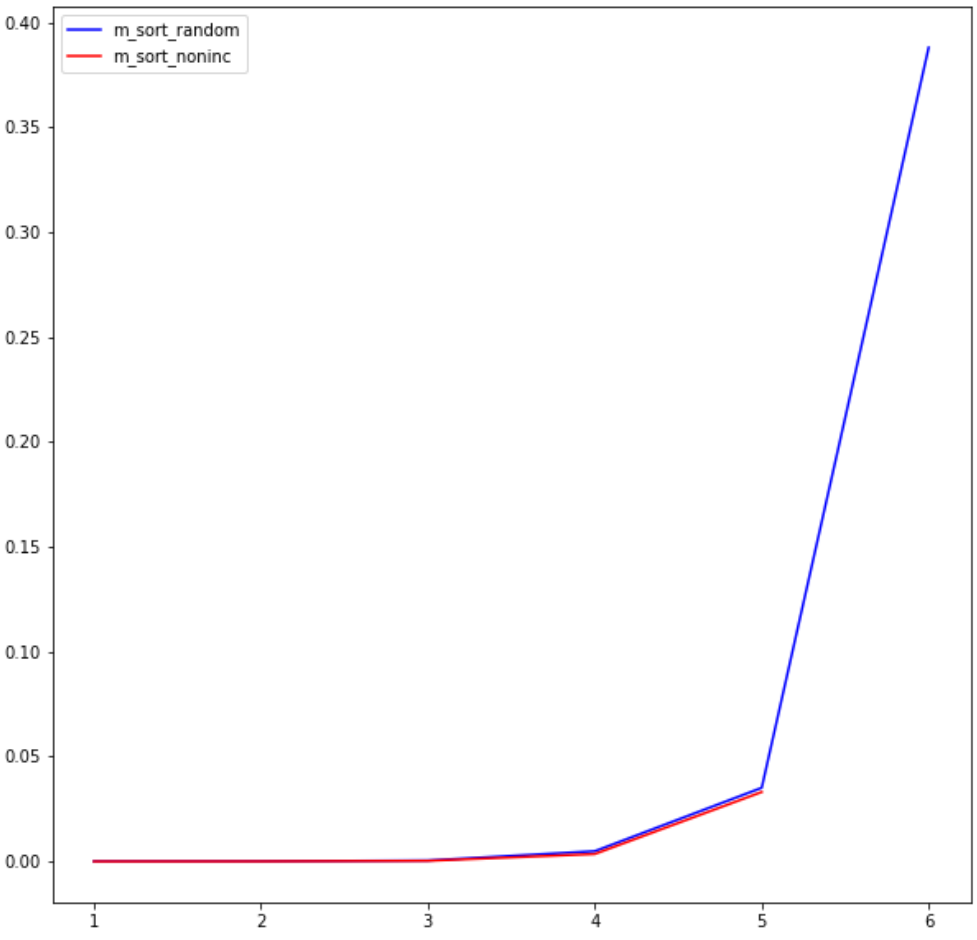
-random order

size 10	size 100	size 1000	size 10000	size 100000	size 1000000
0.000008	0.000044	0.000459	0.004901	0.035113	0.387908

-non-increasing order

size 10	size 100	size 1000	size 10000	size 100000
0.000008	0.000046	0.000313	0.003549	0.033106

소요된 시간을 그래프로 시각화한 것은 다음과 같다.



merge sort는 random order를 입력으로 받았을 때 quick sort보다 조금 느리지만, worst case에도 시간복잡도를 O(nlogn)으로 유지하는 것을 그래프를 통하여 알 수 있다.

4) fast-quick sort

과제 2의 네 번째 알고리즘으로 quick sort를 조금 변형시켜 fast-quick sort로 사용하였다. 해당 알고리즘은 quick sort의 시간복잡도에서 상수항을 최대한 줄여 조금 더 빠른 시간에 정렬을 수행한다. 각 테스트케이스별로 소요된 시간은 다음과 같다.

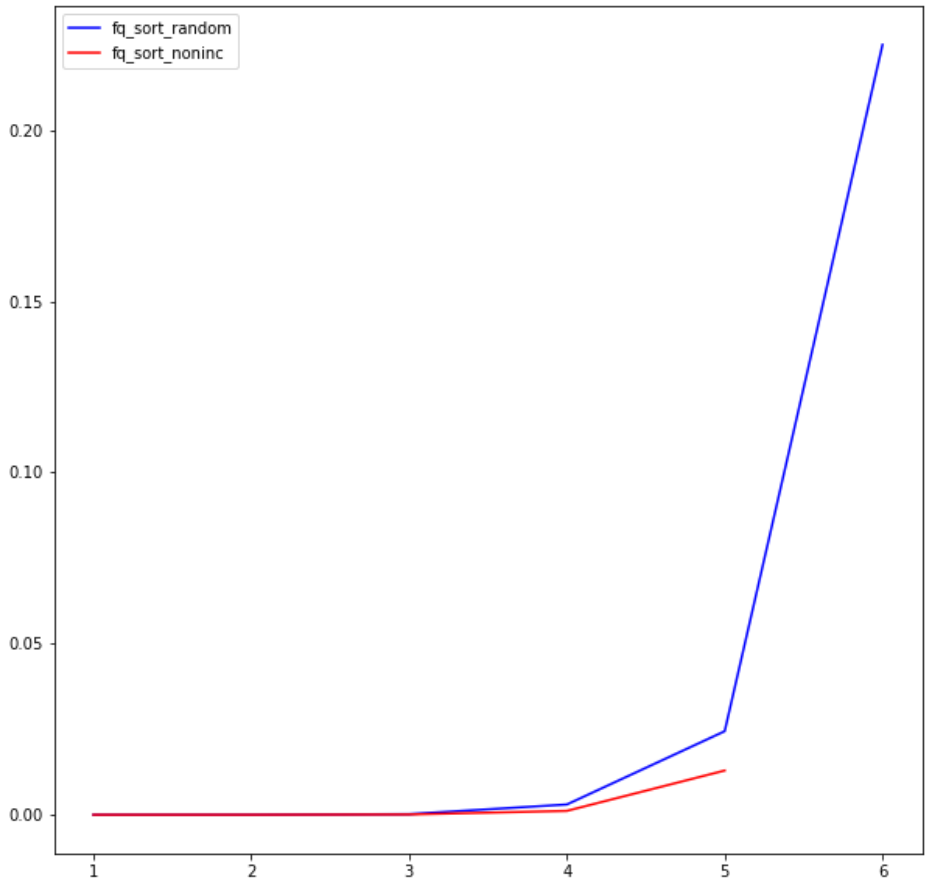
-random order

size 10	size 100	size 1000	size 10000	size 100000	size 1000000
0.000003	0.000017	0.000202	0.003012	0.024418	0.224963

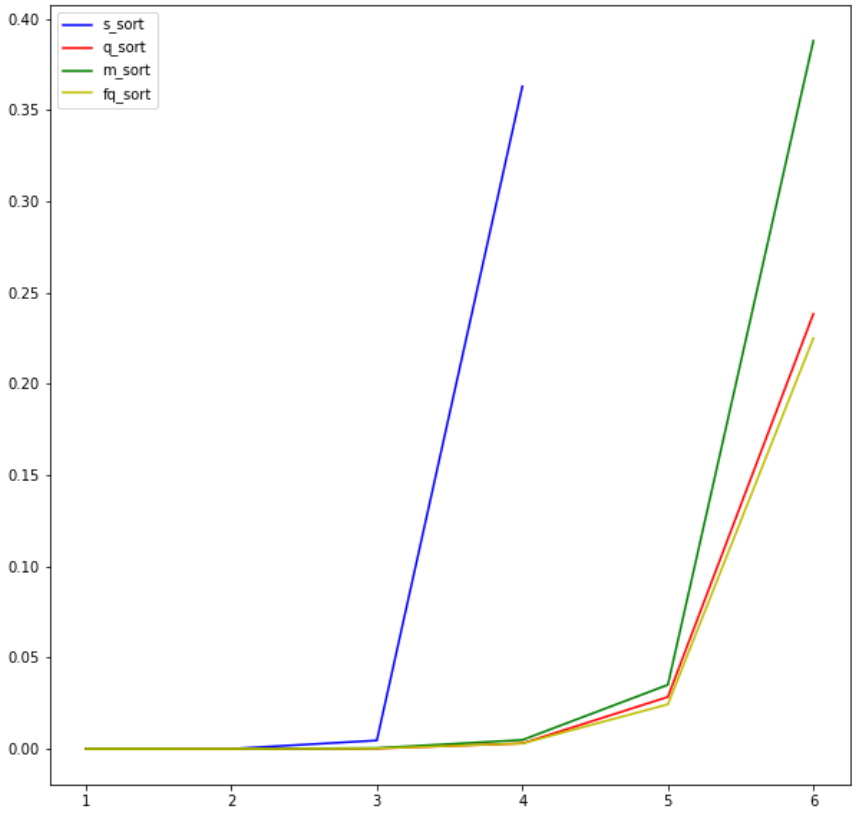
-non-increasing order

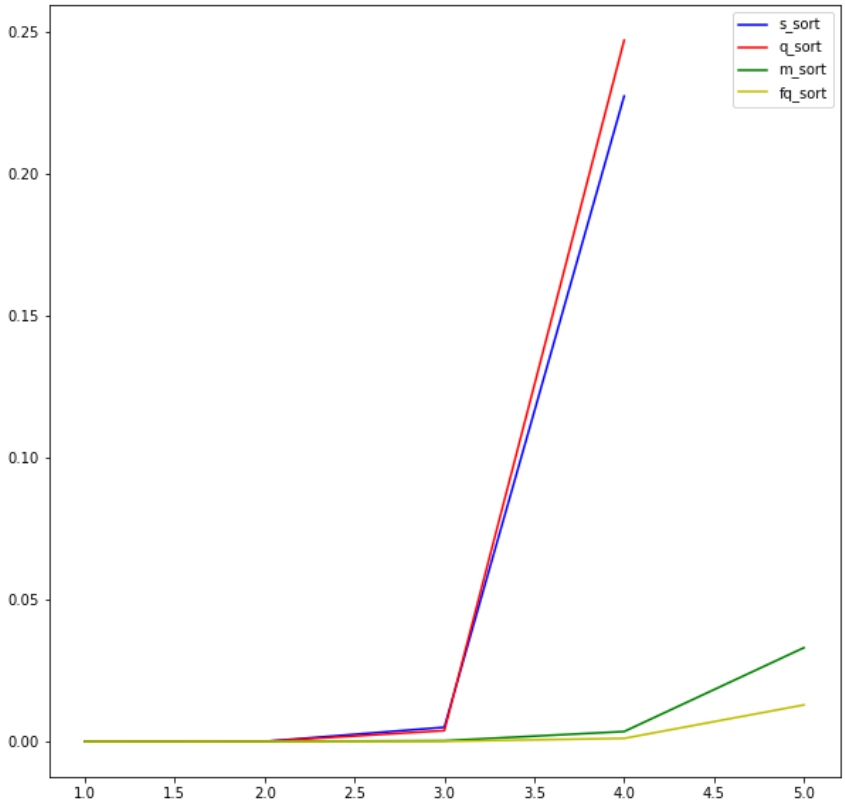
size 10	size 100	size 1000	size 10000	size 100000
0.000014	0.000022	0.000101	0.001141	0.012930

소요된 시간을 그래프로 시각화한 것은 다음과 같다.



5) random order 및 non-increasing order의 그래프 비교





위의 그래프들에서 볼 수 있듯, random order의 경우 fast-quick sort, quick sort, merge sort, selection sort의 순으로 소요된 시간이 적었고 non-increasing order의 경우 fast-quick sort, merge sort, selection sort, quick sort의 순으로 소요된 시간이 적었다.

2. 알고리즘 4의 설계와 구현

과제 2의 알고리즘 4는 초기 단계에서는 강의 시간에서 다뤘던 introspective sort로 구현하려고 하였다. 즉, quick sort에서 정렬의 범위가 15 이하, 즉 정렬시킬 원소의 개수가 15 이하인 경우에는 insertion sort, 재귀의 횟수가 log(n)을 초과할 경우에는 heap sort(merge sort), randomly-partition을 변형하려고 했다. 하지만, 테스트케이스를 생성한 후 테스트를 할 때 해당 경우를 모두 적용시켰을 때에는 알고리즘 2의 quick sort보다 소요 시간이 더 많이 걸리는 현상이 지속적으로 발생하였다.

따라서, 두 번째 단계에서는 heap sort, randomly partition을 제외하고 median-of-three partition을 적용시켜 소요되는 시간을 줄일 수 있었다. 하지만 여전히 알고리즘 2의 quick sort보다 소요 시간이 더 걸렸고, insertion sort가 적용되는 부분은 소요 시간 상 불리한 점이 없다고 판단되어 median-of-three partition을 제외하였다.

최종적으로, 기존의 quick sort에서 정렬하려는 원소의 개수가 15개 이하일 경우에 insertion sort만을 적용시키는 방법만을 택하게 되었다. 입력의 범위가 작거나 이미 거의 정렬된 상태의 입력이 들어오는 경우에 insertion sort가 가장 효율적인 정렬이라고 알려져 있으므로, insertion sort를 quick sort에 적용시켜 변형하였다. 기존에 적용하려고 했던 heap sort나 randomly partition은 처음 설계와 다르게 시간복잡도의 상수항을 과도하게 크게 만들었던 것으로 판단된다.