알고리즘설계와분석 과제2

20181256 수학과 김도현

- Experiment environment:

보고서 작성을 위해 input 파일 및 output 파일 없이 코드를 단순화 하여, MAC OS 인 아이맥 m1 16gb램 에서 xcode에서 같은 환경으로 진행하였다.
하지만 실제 코드는 과제의 요구대로 linux 상에서 정상적으로 input을 받고 작동한다.

- Experiment setup:

main 함수 시작 후 알고리즘이 수행부터 해당 list 가 정렬될 때 까지, clock() 함수를 이용하여 시간을 측정하였다. 이때 second으로 계산하기 위해 clocks_per_sec으로 나눴다. 소수점 자리 6자리까지 측정하였다. 가장 작은 N=10 부터 순차적으로 늘려 N=1000000 까지 시간을 측정했다.

srand((unsigned int)time(NULL))과 rand() 함수를 사용하여 input을 계속해서 random 하게 받았고 같은 N 에 따라 다른 random input 값을 5번 바꿔서 average running time 을 계산하였다.

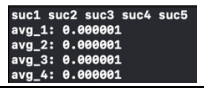
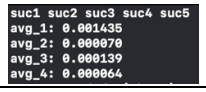
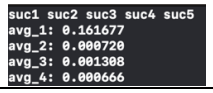
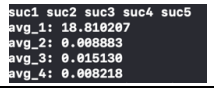
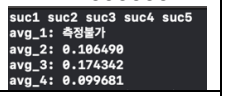
시간 측정 편의상 input 받는 법을 rand 를 통해 바로 받기 위해 단순화 하였다. 하지만 모든 알고리즘이 같은 실험 환경에서 실행했기 때문에 페널티를 받는 알고리즘은 없다.

시간 측정이 1분 이상 걸리는 경우 측정 불가로 대체하였다.

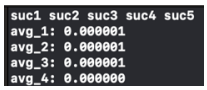
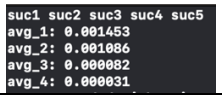
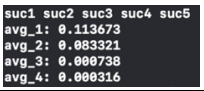
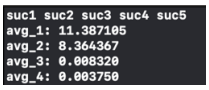
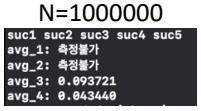
실제 코드는 실행 시 과제의 요구대로 result_input00000.txt 형태로 출력된다.

아래 사진에서의 suc(i) 의 의미는 i번째 돌았을 때 각 알고리즘에 의해 정렬된 list 들을 비교하여 모두 같으면 suc 하나라도 다르면 fail 를 출력한다. avg_i 는 i 인덱스에 해당하는 sorting 알고리즘의 5번 평균 시간이다.

1. random list
Average running time

	N=10	N=1000	N=10000	N=100000	N=1000000
					
Index. 1 (bubble sort)	0.000001	0.001435	0.161677	18.810207	측정 x
Index. 2 (quick sort)	0.000001	0.000070	0.000720	0.008833	0.106490
Index. 3 (merge sort)	0.000001	0.000139	0.001308	0.015130	0.174342
Index. 4 (improved sort)	0.000001	0.000064	0.000666	0.008218	0.099681

2. sorted in non-increasing order list(opposite way)

	N=10	N=1000	N=10000	N=100000	N=1000000
					
Index. 1 (bubble sort)	0.000001	0.001453	0.113673	11.387105	측정 x
Index. 2 (quick sort)	0.000001	0.001086	0.083321	8.364367	측정 x
Index. 3 (merge sort)	0.000001	0.000082	0.000738	0.008320	0.093721
Index. 4 (improved sort)	0.000000	0.000031	0.000316	0.003750	0.043440

- My comments on the experience:

우선 시간 복잡도가 O(n^2) 인 알고리즘을 선택할 때 insertion sort 가 아닌 bubble sort 를 사용했다. 그리고 quick sort, merge sort 는 알려진 시간 복잡도는 똑같이 O(nlogn) 로 asymptotic 하게 동일하지만, 실제 random한 input에서 N 값이 커질수록 quick sort가 더 빠르게 정렬 되었다. 이것은 merge sort는 분할 이후 다시 병합 과정이 포함되었기 때문으로 결론 내릴 수 있다. 반면 거꾸로 정렬된 input의 경우 quick sort는 시간 복잡도가 항상 O(nlogn) 인 Merge sort 보다 시간 복잡도가 O(n^2)인 bubble sort 쪽에 더 가까운 시간이 나왔음을 알 수 있다. 이것은 quick sort 알고리즘 특성상 pivot을 배열의 가장 오른쪽으로 잡는데 이때 배열의 가장 큰 값이 잡히기 때문에 recursion 할 때 마다 하나만 제거되기 때문이다.

위의 실험 결과를 이용해서 최적의 시간이 나와야 하는 index 4번 improved sort를 짰다. 결과 분석을 통해 일반적인 random 한 경우 quick sort 가 가장 빠름을 알아냈다. 그래서 quick sort 와 비슷하게 짰지만 pivot을 잘 선택하는 Median of three 를 가장 오른쪽, 가장 왼쪽, 가운데 중 선택하

는 것이 아니라 1/4 지점, 2/4 지점, 3/4 지점을 선택해서 최대한 worst 케이스인 역으로 정렬된 곳에서 좋은 pivot을 뽑아 벗어나고자 했다. 그 결과 모든 경우 pivot이 적절하게 뽑혀 향상된 quicksort가 되어 시간을 매우 단축시킬 수 있었다.

merge sort 를 결합한 intro sort 느낌으로 고려 했지만 pivot이 좋다면 결국 quicksort가 더 빠르다고 판단해서 제외 시켰다. 추가적으로 bubble sort, insertion sort 역시 O(n^2) 인 시간 복잡도를 가지지만 insertion sort는 조건에 따라 for문에서 break가 걸리기 때문에 더 빠른 정렬 시간을 보였다. 그리고 n이 충분히 작다면 insertion sort 가 상대적으로 quick sort 보다 더 빠를 가능성이 높기 때문에 n<16 이하가 되면 insertion sort 로 돌아가도록 설정하니 모든 input과 관계없이 대부분 quick sort 보다 평균 sorting 시간이 짧아 효과적인 improved sort 를 짜게 되었다.

가장 처음 시간 복잡도가 O(n) 인 radix sort 가 생각났다. 코드 파일을 보면 짜서 돌려봤는데 이상하게 Mac OS 의 xcode 에서는 improved sort 보다 radix의 더 정렬 시간이 빨랐는데 linux 환경에서는 더 느리게 나왔다. 비록 둘 다 좋은 성능을 보이지만 같은 코드로 돌렸는데 결과가 반대로 나오는 이유는 실험 환경이라 생각했다. 과제 제출 시의 실험 환경은 linux 이기 때문에 최종적인 코드는 적절한 pivot을 뽑은 quick sort, n이 작다면 insertion sort 로 이루어진 improved sort 로 제출하였다.