CSE3081: Design and Analysis of Algorithms (Fall 2022)

MP2: Master of Sorting

20180071 오서영

1. Experienment environment

CPU speed: Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz 2.71 GHz

memory size: 8.00GB RAM

OS type and version: Windows 64비트 운영 체제, x64 기반 프로세서

리눅스 기반 실습 환경을 조성하기 위해 동일한 기기의 개인 코랩 상에서 실습을 진행했다.

2. Experiment setup

(1) 변수

-int* list: 정렬 전 배열과 정렬 후 배열을 모두 저장하는 데 사용되는 정수형 배열 변수이다. 동적 할당을 통해 입력 크기 만큼의 공간을 사용하였다.

-n: 정렬할 배열의 크기를 저장한 정수형 변수이다.

-start, end: 시간 측정을 위해 사용한 double형 변수이다.

(2) 함수

-(**algorithm1**)insertion_sort: insertion sort를 구현하였다.

-(**algorithm2**)quick_sort, partition: quick sort를 구현하였다.

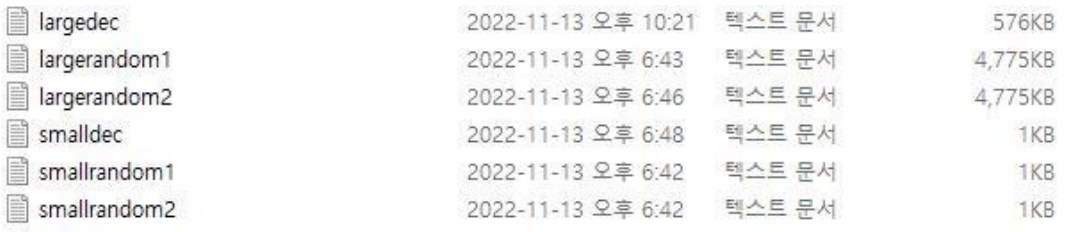
-(**algorithm3**)merge_sort, merge: merge sort를 구현하였다.

-(**algorithm4**)threesort, bet_quick_sort: quick sort with median of 3를 구현하였다.

(3) 단위

-시간 측정: 알고리즘 시간 측정을 위해 시작 시간과 끝 시간을 clock() 함수와 CLOCKS_PER_SEC으로 구한 다음, double로 형변환하여 둘의 차를 seconds 단위로 구하였다.

(4) 입력 사이즈의 범위



-large random input: 1~9999 범위의 랜덤한 수로 구성되어 있으며, 배열의 개수는 1000000개이다. 중복값을 허용하며 양수만 있다. 랜덤한 케이스이므로 두 입력 파일에 대한 알고리즘 수행시간을 각각 구한 뒤, 둘의 평균을 구하였다.

-small random input: 1~9999 범위의 랜덤한 수로 구성되어 있으며, 배열의 개수는 10개이다. 중복값을 허용하며 양수만 있다. 랜덤한 케이스이므로 두 입력 파일에 대한 알고리즘 수행시간을 각각 구한 뒤, 둘의 평균을 구하였다.

- large non-increasing input: 1~100000 범위의 수가 decreasing한 순서로 정렬되어 있으며, 배열의 개수는 100000개이다. 중복값은 없으며 양수만 있다.

- small non-increasing input: 1~10 범위의 수가 decreasing한 순서로 정렬되어 있으며, 배열의 개수는 10개이다. 중복값은 없으며 양수만 있다.

3. Experiment

(1)실습 목적

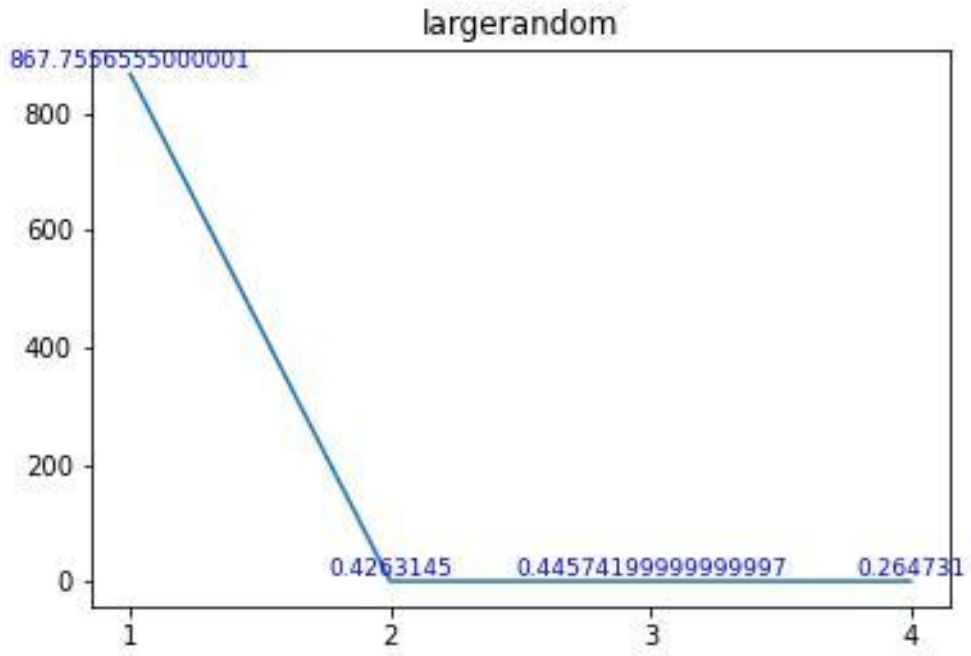
정수로만 구성된 배열을 정렬하는 문제에 대해 네가지 알고리즘을 적용하여 어떤 케이스에 어떤 알고리즘이 가장 빠른 속도를 보이는지를 측정한다. 앞의 세가지는 insertion sort, quick sort, merge sort로 구현하였으며 네번째 알고리즘은 자신이 설계한 효율적인 알고리즘으로 구현한다.

(2)실습 결론

random, decreasing으로 나뉠 수 있는 네가지 input에 대하여 실습을 진행하였다. 이중 random한 input의 경우 input file에 따라 수행시간이 달라질 수 있는 문제가 있기 때문에 두 종류의 다른 input file로 실습을 진행하고 수행시간의 평균을 구하였다.

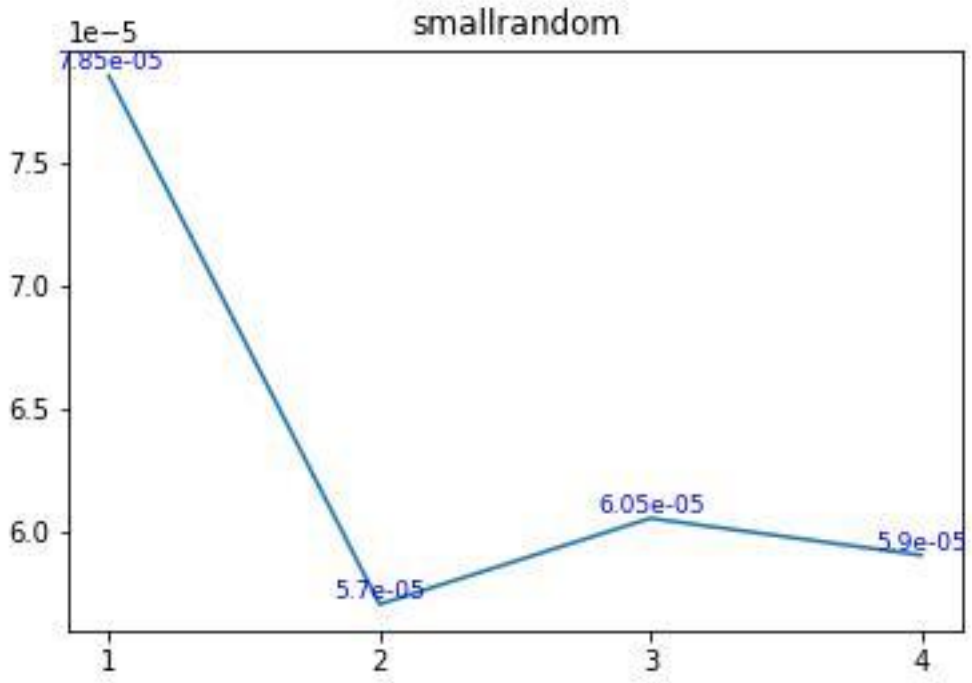
(1)large random

	algorithm1	**algorithm2**	**algorithm3**	**algorithm4**
large random 1	867.503672	0.423288	0.435109	0.267189
large random 2	868.007639	0.429341	0.456375	0.262273
mean	867.755656	0.426315	0.445742	0.264731



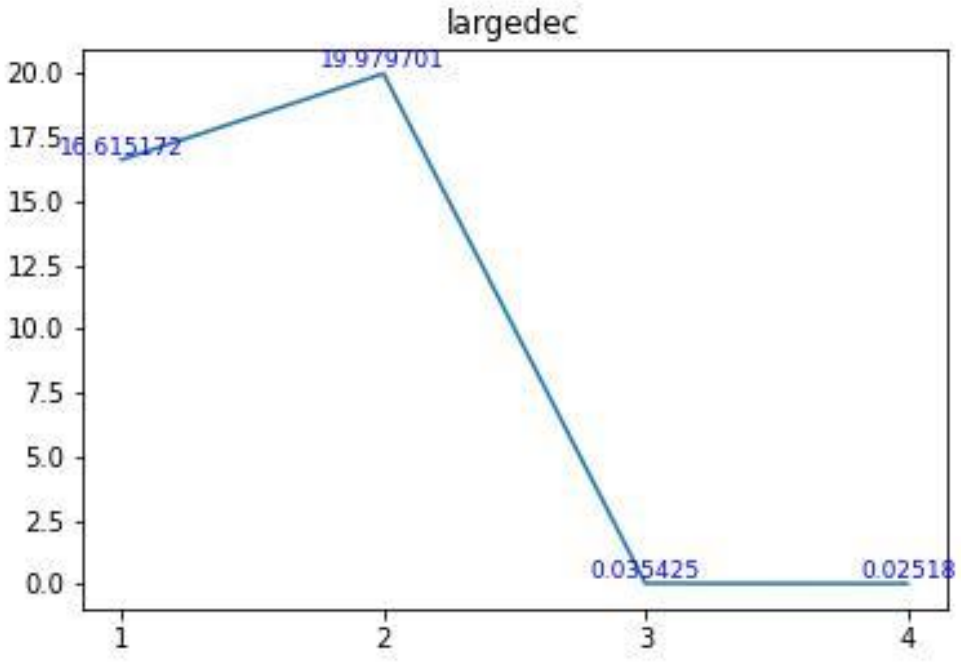
(2)small random

	algorithm1	algorithm2	algorithm3	algorithm4
large random 1	0.000065	0.000057	0.000054	0.000051
large random 2	0.000092	0.000057	0.000067	0.000067
mean	0.000079	0.000057	0.000061	0.000059



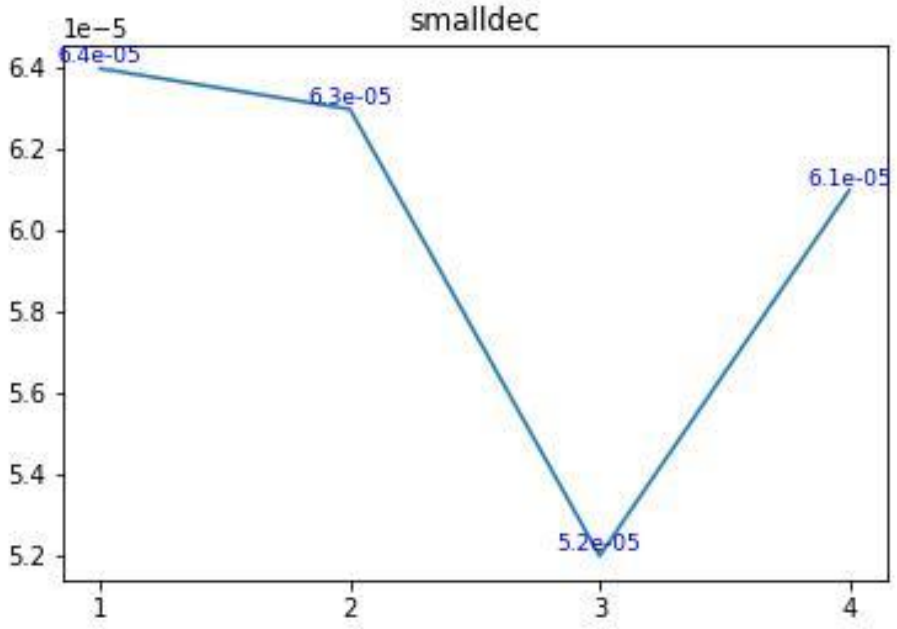
(3)large non-increasing

	algorithm1	algorithm2	algorithm3	algorithm4
large non-increasing	16.615172	19.979701	0.035425	0.025180



(4)small non-increasing

	algorithm1	algorithm2	algorithm3	algorithm4
small non-increasing	0.000064	0.000063	0.000052	0.000061



4. Your comments on the experience

(1)과 (2)에서 볼 수 있듯이 random한 input에 대해서는 알고리즘 2, 3, 4가 큰 차이를 보이지 않았으나 알고리즘 1이 훨씬 오랜 시간이 걸렸다. 또한 배열의 개수가 작을 때보다 커질 수록 이 차이는 극명해지는 것을 알 수 있었다. 즉, 입력값의 크기가 커질수록 O(n^2)의 시간복잡도를 가진 알고리즘보다는 θ(nlogn)의 시간복잡도를 가진 알고리즘이 효율적이라는 것을 알 수 있다.

(3)과 (4)에서 알 수 있듯이 정렬된 non-increasing input에 대해서는 알고리즘 3과 4가 대체적으로 빠른 속도를 보였고, 알고리즘 1과 2가 느린 속도로 수행되었다. 이때 입력값이 커지자 알고리즘 2가 오히려 알고리즘 1보다 긴 수행시간을 소요하는 모습을 관찰할 수 있었다. 그리고 마찬가지로 배열의 개수가 클수록 이들의 차이는 벌어지는 모습을 보였다. 결국, 입력값의 크기가 커질수록 시간 복잡도가 수행시간에 미치는 영향도 커질뿐만 아니라, quick sort는 정렬된 배열에 대해서 최악의 시간복잡도를 가진다는 것을 확인할 수 있었다.

5.알고리즘 4 설계

quick sort는 평균 시간 복잡도가 θ(nlogn)으로서 가장 빠른 정렬 알고리즘 중 하나이다. 그러나 quick sort에는 가장 큰 단점이 있는데, 이미 정렬되어 있는 배열에 대하여 최악의 시간 복잡도 O(n^2)가 소요된다는 점이다. 이는 맨앞 혹은 맨뒤의 값을 pivot으로 잡기 때문에 발생하는 문제이므로, 결국 pivot을 무엇으로 잡을지가 quick sort의 성능을 결정한다고 할 수 있다. 따라서 맨앞, 중간, 맨뒤 값에 대하여 먼저 정렬을 수행하고, 다시 정렬된 맨앞, 중간, 맨뒤 값들 중 중간값을 pivot으로 잡는 quick sort with median of 3를 구현하였다.

정렬되지 않은 list가 있으면 우선 맨앞, 중간, 맨뒤 값들을 정렬하고 그중 정렬된 중간값을 pivot으로 잡는다. 이제 맨앞과 맨뒤는 이미 정렬되었으므로 제외하고, 나머지 값들을 quick sort와 동일한 방식으로 정렬한다. quick sort with median of 3에서는 pivot을 최좌측 혹은 최우측으로 설정하는 것이 아니라 맨앞, 중간, 맨뒤 중 중앙값으로 설정하기 때문에, quick sort의 최악의 case(정렬된 배열)에서도 θ(nlogn)에 가깝게 작동할 수 있다.