알고리즘 설계 설계와 분석 2차 과제 보고서

20181264 노영현

1. Experiment environment

MacOS Monterey

M1 칩

메모리 사이즈 : 8GB

CPU speed : 2.89 GHz

2. Experiment setup

모든 input 숫자의 범위는 int 범위 내에서 정의하였다. input file들을 만들기 위해서 새로운 c 파일을 만들어 fopen함수와 fprintf함수를 사용하였다. random list는 random()함수를 사용해 random한 수들을 무작위로 input 크기만큼 file에 적어주었다. 2938.567691non-increasing order list 는 적당한 수에서 시작해 1부터 줄여가면서 input 크기만큼 file에 적어주었다.

3. Describe how the running time of your algorithms grow as the input size grows

input값이 커짐에 따라서 그에 따른 running time을 측정하기 위해서 input크기가 서로 다른 5개의 파일을 준비하였다. 파일 각각의 input 크기는 10, 100, 10000, 100000, 1000000이다. 마지막 input크기인 1000000은 2^20과 유사한 크기이다. 또한 본 실험에서 sorting할 list가 random list인 경우, non-increasing order list인 경우 2경우로 나누어서 실험을 진행하였다. 즉 random list인 file 5개, non-increasing order list인 file 5개 총 10개의 파일로 실험을 진행하였다.

(1) random list

random list들을 알고리즘 4개로 각각 돌려본 결과를 아래 표에 나타내었다.

	10	100	10000	100000	1000000
algorithm 1	0.000005	0.000093	0.313285	27.465422	2789.612120
algorithm 2	0.000005	0.000030	0.005011	0.038556	0.404100
algorithm 3	0.000008	0.000055	0.006234	0.045075	0.454405
algorithm 4	0.000004	0.000036	0.002876	0.036292	0.383850

algorithm 1의 경우 시간복잡도 O(n^2)을 가지는 selection sort로 구현하였다. 그 결과 input의 개수가 늘어남에 따라 running time이 굉장히 크게 늘어나는 것을 볼 수 있다. algorithm2와 algorithm 3은 각각 quick sort와 merge sort로 두 sorting algorithm 모두 O(nlongn) 시간복잡도를 가진다. 두 algorithm은 input의 개수가 크게 커지더라도 running time이 천천히 커지는 것을 볼 수 있다. algorithm 4의 경우 직접 구현하였는데 이 또한 quick sort를 최적화 한 것이기 때문에 O(nlogn)의 시간 복잡도를 가진다. 따라서 algorithm 2,3과 비슷한 증가세를 보인다.

(2) non-increasing order list

non-increasing order list들을 알고리즘 4개로 각각 돌려본 결과를 아래 표에 나타내었다.

	10	100	10000	100000	1000000
algorithm 1	0.000005	0.000050	0.310828	28.030773	2938.567691
algorithm 2	0.000006	0.000108	0.459822	41.453683	2489.221809
algorithm 3	0.000006	0.000035	0.004249	0.027744	0.291941
algorithm 4	0.000005	0.000030	0.005847	0.043217	0.408813

이를 그래프로 나타내면 아래와 같다.

algorithm1,3은 모두 (1) random-list와 크게 다르지 않은 상승세를 보이는 것을 볼 수 있다. 하지만 algorithm2인 quick sort의 경우 input이 커짐에 따라 running time이 굉장히 커지는 것을 알 수 있다. 이는 quick sort의 특성상 pivot을 비효율적으로 잡았을 경우 dividing이 계속 0개와 n-1개만큼 되어 O(n^2)의 시간복잡도를 가지기 때문이다. 이를 해결한 것이 직접 작성한 algorithm4이다. 따라서 algorithm4는 non-increasing order list임에도 O(nlogn)의 시간복잡도로 실행됨을 확인 할 수 있다.

4. How you designed algorithm4

위에서 2가지를 확인할 수 있다. 첫 번째는 앞서 언급했듯이 quick sort의 경우 O(nlogn)의 시간복잡도를 대게 가지지만 non-increasing order list에서 O(n^2)의 시간복잡도를 가져 굉장히 비효율적이라는 점이다. 두 번째는 n이 10으로 굉장히 작을 때, 오히려 재귀함수를 사용하는 merge sort와 quick sort보다 O(n^2) 정렬인 insertion sort(selection sort)가 더 빠르다는 점이다. 이 2가지를 활용해서 quick sort를 최적화 해준 뒤, insertion sort와 combination 해주었다.

우선 본인이 algorithm2에서 작성한 quick sort는 pivot을 가장 오른쪽으로 지정해준 뒤, 알고리즘을 진행하였다. 이 때문에 이미 정렬되어 있거나 거꾸로 정렬되어 있는 경우 시간복잡도가 O(n^2)이 되는 불상사가 발생한다. 이를 방지 하기 위해서 partition 함수를 실행하기 전에 Pivot을 새로 정해준다. 인덱스 p부터 r까지에서 quick_sort를 진행한다고 가정하고 pivot을 뽑는 과정을 설명하겠다. 새로운 arr를 만들고 arr에 p, (p+r)/2, r을 넣어준다. 이후 arr를 insertion sort를 사용해 정렬하고, arr[1]에 들어있는 인덱스값이 pivot의 인덱스가 된다. 이를 위해서 arr[1]에 들어있는 인덱스에 위치한 수와 r인덱스에 위치한 수를 swap해준다. 이 과정을 거치면 최대한 pivot이 가장 큰 수나 가장 작은 수를 안가지도록 할 수 있다. 이후의 과정은 algorithm2에서 작성한 quick sort와 동일하다.

앞서 확인한 부분 중 두 번째는 n이 작은 경우 quick sort, merge sort보다 insertion sort가 더 빠른 것이었다. 따라서 sorting할 리스트를 받았을 때 리스트의 길이가 50보다 작은 경우 insertion sort를 활용해 정렬하고, 50이상인 경우 위에서 직접 작성한 최적화 quick sort를 활용해 정렬하였다.

5. Comments on the experience

해당 과제를 진행하면서 우선 4개의 서로 다른 정렬 알고리즘을 작성할 수 있었다. 공부한 뒤 직접 코드를 작성해보는 것에서 정렬에 대해서 더 자세하게 배울 수 있었다. input size가 커짐에 따라서 running time 차이가 굉장히 큰 것을 알 수 있었다. input size가 백만인 경우, O(n^2)의 경우 겅의 40분 가까이가 소요되는데, O(nlogn)은 1초도 채 걸리지 않는 것을 확인하였다. 왜 알고리즘에서 시간복잡도를 정말 조금이라도 줄이려고 노력하는지 체감할 수 있었다. 하지만 input의 개수가 매우 작을 때는 시간복잡도가 O(n^2)이더라도 구현한 방법에 따라서 O(nlogn)보다 더 빠른 경우도 관찰할 수 있었다. input의 개수에 따라서 어떤 시간복잡도를 사용해야되는 지를 잘 확인할 필요가 있겠다고 느꼈다.

또한 겉보기에는 같은 시간복잡도를 가지는 merge sort, quick sort이지만 들어오는 input list가 어떤 형태를 가지느냐에 따라서 매우 다른 성능을 가지는 것을 볼 수 있었다. sorting은 다 장단점이 굉장히 뚜렷하다는 것을 느꼈다. 4번 알고리즘을 작성하기 위해서 여러가지 정렬 알고리즘들을 많이 검색하고 공부해보았다. 이 과정에서 모든 정렬 알고리즘들이 확실한 장단점이 있다고 느꼈다. 하나의 정렬 알고리즘 만을 고집하는 것이 아니라 상황에 맞춰서 그에 맞는 알고리즘을 사용하는 것이 중요할 것 같다고 느꼈다.