Sorting 보고서

2013-11422 이은하

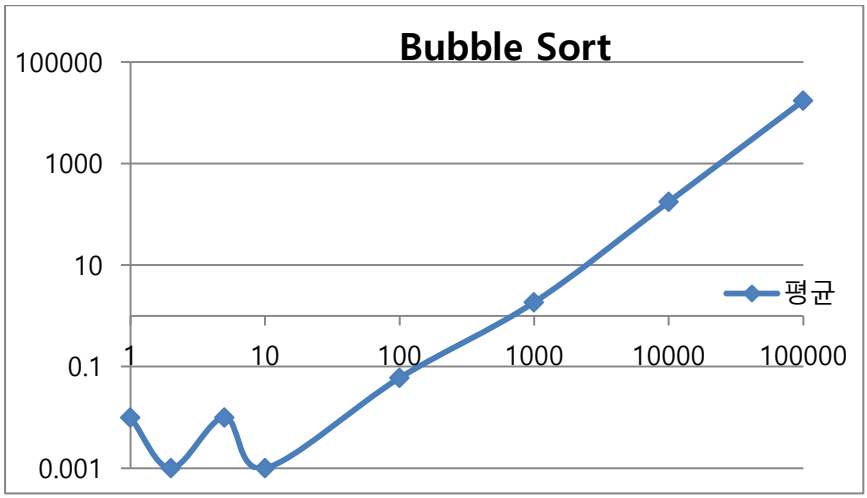
1. Bubble Sort: 바로 옆에 있는 데이터와 크기를 비교하는 과정을 배열의 처음부터 끝까지 수행한 후, 그럼으로써 정렬된 마지막 값을 빼고 같은 과정을 반복하는 방법으로 정렬한다. 이중 for 문을 돌려 정렬했다.

- 이론상 running time: $(n-1)+(n-2)+\cdots+2+1 = O(n^2)$

- 개수 별 실험 결과(범위는 -100000000 ~ 100000000, 10000개 까지는 100번, 100000개에서는 시간 관계상 10번만 수행)

개수	1	2	5	10	100	1000	10000	100000
평균	0.01	0.0	0.01	0.0	0.06	1.85	176.46	17311.0
표준편차	0.1	0.0	0.1	0.0	0.24	0.92	2.73	831.02

0.0을 0.001이라 치환하고 그래프를 그려보면 다음과 같다.



→ bubble sort는 $O(n^2)$의 시간 복잡도를 지니므로 위 그래프의 기울기가 2가 되어야 한다. 불안정한 값을 가지는 앞 부분의 값을 제외하고 계산해 보면 100과 1000 사이가 약 1.48, 1000과 10000 사이가 약 1.99, 10000과 100000 사이가 약 1.99로 비슷한 수치를 보인다.

2. Insertion Sort: 두 번째 원소에서 시작해서 앞 쪽 정렬에서 원소가 들어갈 자리를 찾아 삽입하는 방법으로 정렬한다. 두 번째 원소부터 for문을 돌려서 앞 쪽 정렬에 그 원소보다 큰 원소의 개수를 세어 그만큼 배열을 뒤쪽으로 밀어내고 빈 자리에 그 원소를 넣었다.

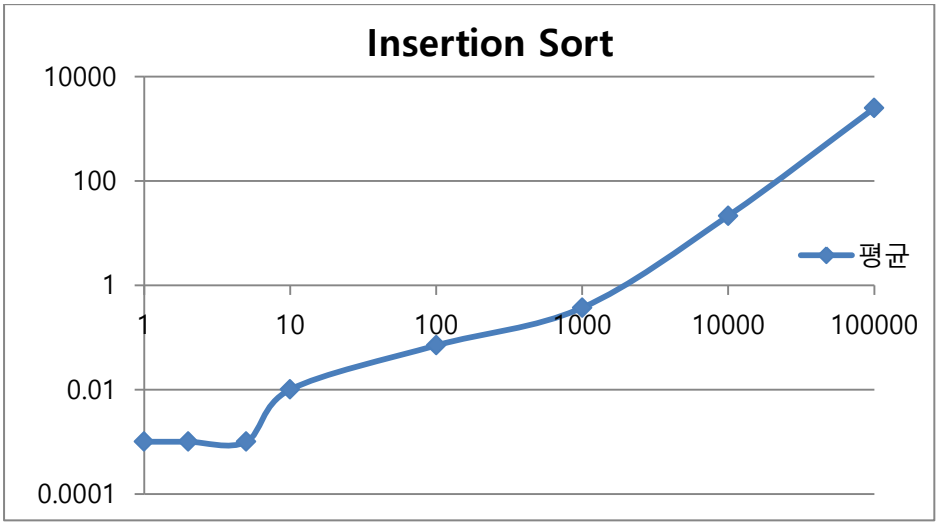
- 이론상 running time(worst case): $1+2+\cdots+(n-2)+(n-1) = O(n^2)$

running time(average case): $\frac{1}{2}(1+2+\cdots+(n-2)+(n-1) = O(n^2)$

- 개수 별 실험 결과(범위는 -100000000 ~ 100000000, 10000개 까지는 100번, 100000개에서는 시간 관계상 10번만 수행)

개수	1	2	5	10	100	1000	10000	100000
평균	0.0	0.0	0.0	0.01	0.07	0.37	21.32	2467.4
표준편차	0.0	0.0	0.0	0.1	0.26	0.83	3.59	727.55

0.0을 0.001이라 치환하고 그래프를 그려보면 다음과 같다.



→ insertion sort는 $O(n^2)$의 시간 복잡도를 지니므로 위 그래프의 기울기가 2가 되어야 한다. 마찬가지로 앞 부분의 값을 제외하고 계산하면 100과 1000 사이가 약 0.72, 1000과 10000 사이가 약 1.76, 10000과 100000 사이가 2.06으로 뒷부분은 잘 맞는 것을 볼 수 있다.

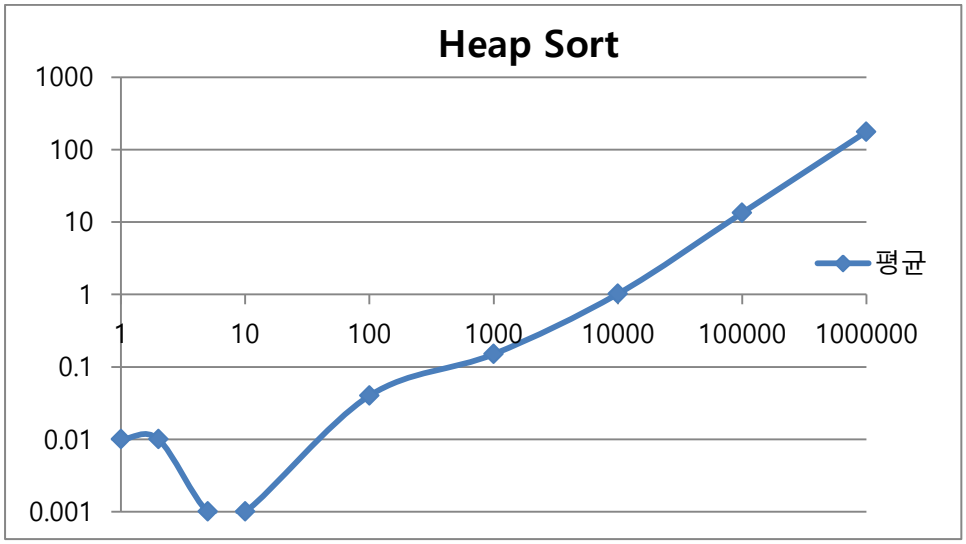
3. Heap Sort: heap을 이용하는 sort이다. heap은 parent node가 child node 이하이거나(min heap) parent node가 child node 이상인(max heap) complete binary tree로, 과제를 수행할 때는 max heap을 이용하였다. max heap의 root node를 뽑아 배열의 맨 뒤에서부터 집어넣고 나머지 배열을 다시 heap으로 만드는 것을 재귀적으로 반복하는 방법으로 정렬한다.

- 이론상 running time = (logn+log(n-1)+…+log2) = (logn+log(n-1)+…+log 2)+(logn+log(n-1)+…+log2)= O(nlogn)

- 개수 별 실험 결과(범위는 -10000000 ~ 10000000, 모두 100번씩 수행)

개수	1	2	5	10	100	1000	10000	100000	1000000
평균	0.01	0.01	0.0	0.0	0.04	0.15	1.02	13.37	176.66
표준편차	0.1	0.1	0.0	0.0	0.2	0.36	0.76	1.23	3.37

0.0을 0.001이라 치환하고 그래프를 그려보면 다음과 같다.



→ heap sort는 O(nlogn)의 시간 복잡도를 가지므로 계산해보면 0.04/200 = 0.0002, 0.15/3000 = 0.00005, 1.02/40000 = 0.0000255, 13.37/500000 = 0.00002674, 176.66/6000000 = 0.0000294 로 뒤로 갈수록 알고리즘에 맞는 값이 나온다는 것을 알 수 있다.

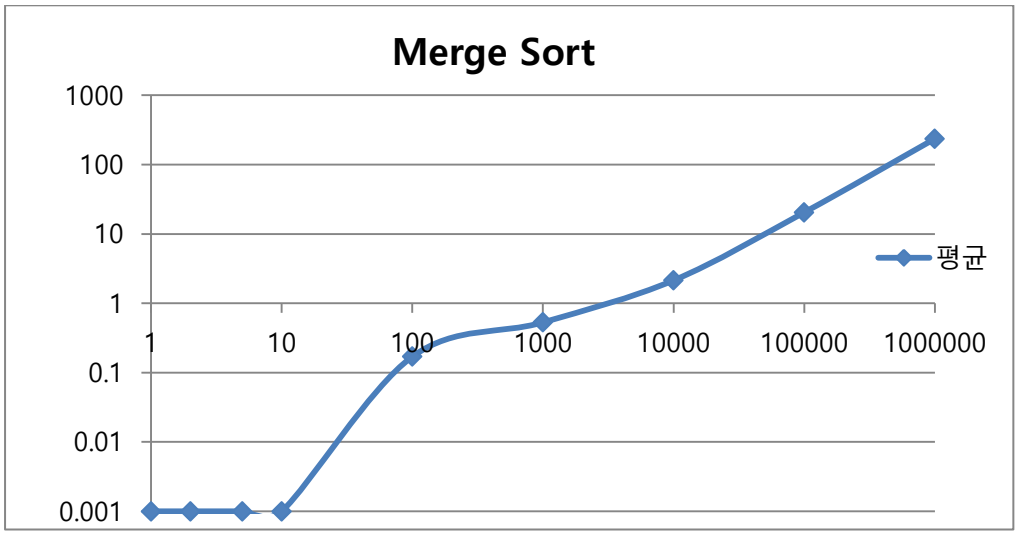
4. Merge Sort: 배열을 반으로 나누어 각각의 배열을 앞 원소끼리 비교하여 합치는데, 나뉜 배열이 충분히 작지 않으면 재귀호출을 하고 각각의 정렬된 배열을 다시 합치는 방법으로 정렬한다.

- 이론상 running time = O(nlogn)

- 개수 별 실험 결과(범위는 -10000000 ~ 10000000, 모두 100번씩 수행)

개수	1	2	5	10	100	1000	10000	100000	1000000
평균	0.0	0.0	0.0	0.0	0.17	0.53	2.14	20.44	235.61
표준편차	0.0	0.0	0.0	0.0	0.38	0.97	2.52	6.93	9.26

0.0을 0.001이라 치환하고 그래프를 그려보면 다음과 같다.



→ merge sort는 O(nlogn)의 시간 복잡도를 가지므로 뒷부분의 값을 계산해보면 100일 때 0.00085, 1000일 때 0.000177, 10000일 때 0.0000535, 100000일 때 0.0000409, 1000000일 때 0.000039로 잘 맞지 않아 보이지만 대략적인 경향은 O(nlogn)을 따르고 있다는 것을 알 수 있다.

5. Quick Sort: 무작위로 원소 하나를 골라 pivot으로 삼아(과제에서는 가장 뒤의 원소) 배열 앞부터 pivot과 비교하여 작으면 pivot의 앞으로, 크면 pivot의 뒤로 보내는 방법으로 정렬한다.

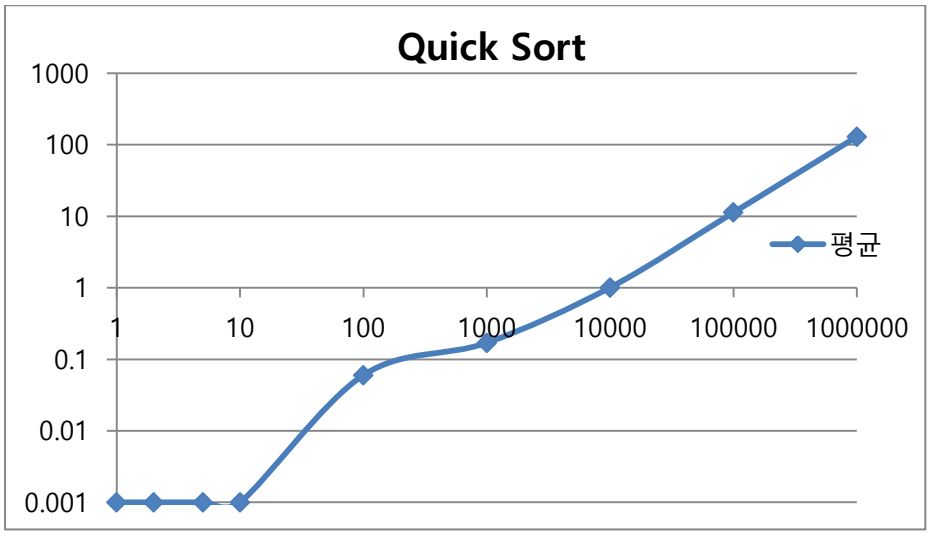
- 이론상 running time(worst case): $O(n^2)$

  running time(average case): O(nlogn)

- 개수 별 실험 결과(범위는 -10000000 ~ 10000000, 모두 100번씩 수행)

개수	1	2	5	10	100	1000	10000	100000	1000000
평균	0.0	0.0	0.0	0.0	0.06	0.17	1.01	11.3	129.61
표준편차	0.0	0.0	0.0	0.0	0.24	0.38	0.77	1.79	2.23

0.0을 0.001이라 치환하고 그래프를 그려보면 다음과 같다.



→ quick sort는 average case에서 O(nlogn)의 시간 복잡도를 가지므로 100일 때 0.0003, 1000일 때 0.0000567, 10000일 때 0.0000253, 100000일 때 0.0000226, 1000000일 때 0.0000216으로 역시나 뒤로 갈수록 맞는 결과가 나온다는 것을 알 수 있다.

* quick sort에 0에서 0까지의 범위에서 100000 정도의 개수를 input으로 집어넣으면 StackOverflow error가 나는데, 그것은 pivot이 계속 한쪽 끝으로만 지정되어 나는 에러이다.

6. Radix Sort: stable sort이며 비교하는 과정이 없다. 나머지를 이용해 각 자리의 숫자만 얻어낸 뒤 일의 자리를 정렬하고 다음 자리를 정렬하고 하는 식으로 정렬한다. 과제에서는 int 범위로 제

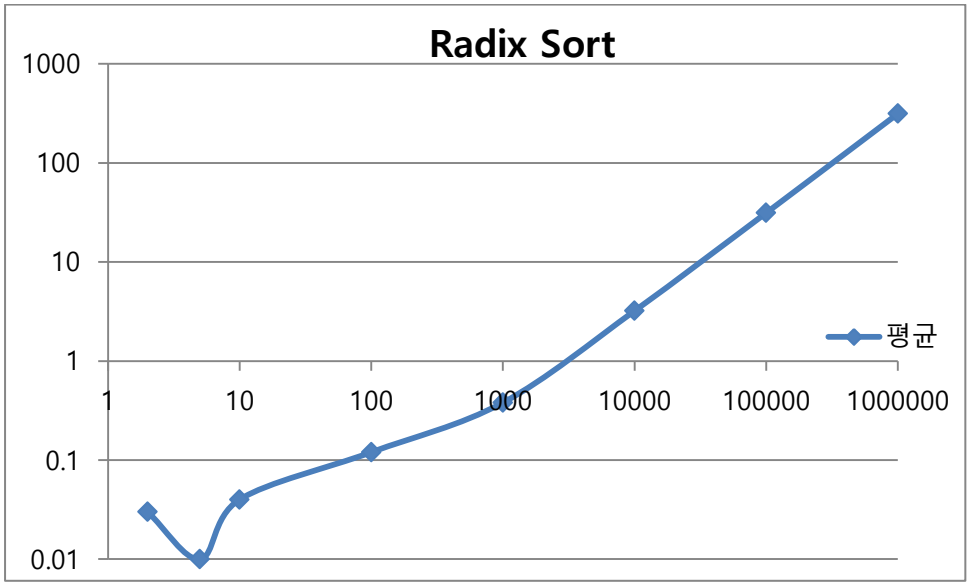
한되어 있으므로 10진법으로 열 자리까지만 정렬하도록 만들었다.

- 이론상 running time: O(n)

- 개수 별 실험 결과(범위는 -10000000 ~ 10000000, 모두 100번씩 수행)

개수	1	2	5	10	100	1000	10000	100000	1000000
평균	0.02	0.03	0.01	0.04	0.12	0.38	3.21	31.3	313.96
표준편차	0.14	0.17	0.1	0.2	0.32	0.52	1.13	2.26	3.09

0.0을 0.001이라 치환하고 그래프를 그려보면 다음과 같다.



→ radix sort는 O(n)의 시간 복잡도를 가지므로 n의 크기에 비례해서 시간이 오래 걸린다는 것을 볼 수 있다.