자료구조 HW4 Sorting 보고서

2013-11425 이창영

1. 개요

- Bubble Sort, Insertion Sort, Heap Sort, Merge Sort, Quick Sort, Radix Sort를 각각 구현해보고 각 정렬 방식에 따라 수행 시간을 비교해본다.

2. 알고리즘 동작 방식

1) Bubble Sort

n개의 인풋에 대해 처음에 인접한 것끼리 앞에서부터 n-1번 비교하고 나면 맨 뒤에 가장 큰 수가 오게 된다. 다음 맨 마지막 것을 제외한 n-1개에 대해 같은 방법을 적용한다. k번째에는 n-k+1개에 대해 같은 방법을 적용하면 뒤에서부터 k개의 데이터가 정렬된 상태로 있다.
이는 O(n^2)의 수행 시간을 가진다.

2) Insertion Sort

k개의 값이 이미 정렬되어있을 때 k+1번째 값을 적절한 위치에 끼워 넣는 방식이다. 끼워 넣을 위치를 판단한 뒤에 뒤쪽의 값들은 한 칸씩 뒤로 옮긴 후 사이에 새 값을 넣으면 된다.
이는 평균적으로 O(n^2)의 수행 시간을 가지지만, 이미 정렬되어있는 인풋에 대해서는 O(n)의 수행 시간을 가진다.

3) Heap Sort

먼저 전체 인풋을 하나씩 읽으며 새로운 minheap을 구성한다. minheap을 구성하고 나면 항상 root 자리에는 최솟값이 있다.(root를 빼내도 다시 minheap의 특성을 유지하도록 하는 자료구조) minheap이 빌 때까지 root를 빼내면서 배열의 앞부분부터 쌓아나간다.
처음에 heap을 구성할 때 데이터 하나에 대해 O(log$n$)의 시간이 걸리므로, 전체를 구성하는 데에는 O($nlogn$)의 시간이 걸린다. 또한 root를 뺄 때도 O(log$n$)의 시간이 걸리므로, 전체를 뺄 때 O($nlogn$)의 시간이 걸린다. 따라서 Heap Sort는 O($nlogn$)의 수행 시간을 가진다.

4) Merge Sort

전체를 반으로 나누어서 왼쪽을 정렬하고, 오른쪽을 정렬해서 합친다는 개념이다. 이를 재귀적으로 반복하면 데이터가 한 개가 될 때까지 쪼개지고 다시 합쳐지면서 정렬이 완료된다. 합칠 때는 왼쪽과 오른쪽의 데이터를 앞에서부터 같이 읽으며 더 작은 값을 새로운 배열에 채워 넣는 방식이다. 채워 넣다가 한쪽을 모두 읽으면 다른 쪽을 한 번에 복사하여 채워 넣는다.
이는 O($nlogn$)의 수행 시간을 가진다. 병합할 때 새로운 배열을 할당해야 하므로 in-place sorting algorithm이 아니다.

5) Quick Sort

전체 데이터 중에서 가장 왼쪽에 있는 값을 기준점으로 잡고, 이를 기준으로 작은 값들은 왼쪽에, 큰 값들은 오른쪽에 위치하도록 만든다. 왼쪽, 오른쪽 각각에 대해서도 같은 방식을 재귀적으로 적용한다.

이는 O($nlogn$)의 수행 시간을 가진다.

6) Radix Sort

가장 작은 자릿수부터 비교하여 정렬한다. 이를 가장 큰 자릿수까지 반복하면 정렬이 완료된다. 각 자릿수에 대해 정렬할 때는 Counting Sort를 사용한다. (0~9로 데이터가 정해져 있는 상황에서 아주 효율적이기 때문에)

양수와 음수에 대해 모두 처리해주어야 하므로 각 자리에 대해 정렬할 때 정렬한 결과를 임시 저장 할 때 음수인 경우와 양수인 경우 서로 다른 배열을 할당하도록 한다. radix sort는 원래 양수인 경우에만 작동하므로 음수인 값들은 -1을 곱한 상태로 거꾸로 정렬하도록 구현하였다. 거꾸로 정렬하기 위해서 Counting Sort에서 count를 할 때 9~0 순서로 count하였다.

이 알고리즘은 각각의 자리에 대해 O(n)의 수행시간을 가지므로 최대 자릿수가 k일 때 O(kn)의 수행시간을 가진다.

3. 각 알고리즘의 수행시간

1) 10,000개 인풋(random -10,000 ~ 10,000)(단위: ms)

	1회	2회	3회	평균
Bubble	131	130	126	129
Insertion	13	13	14	13.3
Heap	2	1	1	1.3
Merge	2	2	2	2
Quick	1	1	1	1
Radix	2	3	1	2

2) 50,000개 인풋(random -50,000 ~ 50,000)(단위: ms)

	1회	2회	3회	평균
Bubble	3719	3738	3798	3751.7
Insertion	322	324	323	323
Heap	5	6	6	5.3
Merge	7	9	7	7.7
Quick	4	5	5	4.7
Radix	6	6	5	5.7

3) 100,000개 인풋(random -100,000 ~ 100,000)(단위: ms)

	1회	2회	3회	평균
Bubble	14195	14337	14355	14295.7
Insertion	1301	1400	1324	1341.7
Heap	12	13	11	12
Merge	16	16	16	16
Quick	10	11	11	10.7
Radix	15	14	15	14.3

4) 10,000,000개 인풋(random -10,000,000 ~ 10,000,000)(단위: ms)
(Bubble, Insertion 제외)

	1회	2회	3회	평균
Heap	2636	2695	2594	2641.7
Merge	2034	2060	2175	2089.7
Quick	1298	1272	1313	1294.3
Radix	1582	1577	1563	1574

4. 분석

똑같은 정렬 알고리즘을 세 번 이상 연속으로 돌렸을 때 속도가 빨라지는 경향이 있어서 최대한 여러 번 실행한 후 마지막 세 개의 결과를 기록하였다.

Bubble Sort와 Insertion Sort는 둘 다 평균 $O(n^2)$의 시간복잡도를 가진다. 그렇지만 Insertion Sort가 훨씬 빠른 속도로 작동하는 것을 볼 수 있다. 그 이유는 Insertion Sort는 어느 정도 정렬이 되어있는 인풋에 대해서는 더 빠른 속도로 작동하기 때문이다. 최선의 경우 이미 정렬되어있는 데이터에 대해서는 O(n)의 시간복잡도를 가진다.

Heap Sort, Merge Sort, Quick Sort는 $O(nlogn)$의 시간복잡도를 가진다. 이론적으로 Quick Sort는 최악의 경우에는 $O(n^2)$의 시간복잡도를 가질 수 있으므로 가장 느릴 것으로 예측할 수 있다. 하지만 실제 결과를 보면 Quick Sort가 가장 빠른 것을 볼 수 있다. 첫 번째 이유는 랜덤한 인풋에 대해서 Quick Sort를 할 때 대체로 합리적인 pivot이 잡히기 때문이고, 두 번째 이유는 Heap Sort, Merge Sort는 heap을 구성하거나 배열을 복사할 때 overhead가 크게 작용하기 때문이라고 생각한다.

Radix Sort는 인풋 중에서 최대 자릿수가 k일 경우 O(kn)의 시간복잡도를 가진다. 가장 빠른 속도로 작동할 것이라고 예상했지만 다른 $O(nlogn)$의 복잡도를 가진 알고리즘들과 큰 차이가 없었다. 이는 10만 개까지의 데이터 인풋에 대해서는 Radix Sort의 overhead로 인하여 생긴 결과인 것 같다. 그래서 Bubble, Insertion을 제외하고 1000만 개의 인풋에 대해 비교해보았더니 Heap, Merge에 비해서는 Radix가 빠르게 작동한 것을 볼 수 있었다. 따라서 더 큰 인풋에 대해서 Radix가 빠르게 작동할 것이라고 예측할 수 있다.