HW #4 Report

2018-10727 전민혁

1. Introduction

   이번 과제에서는 다양한 정렬 알고리즘을 구현하고, 이들에 대한 비교를 수행하며 각 알고리즘의 특성을 알아보았다.

   구현한 알고리즘은 Bubble Sort, Insertion Sort, Heap Sort, Merge Sort, Quick Sort, Radix Sort 가 있다.

   알고리즘 간의 비교를 위한 실험은 i7-8565U Windows 10 x86-64 환경에서 OpenJDK 11 을 이용해 진행하였다.

2. Background

   A. Bubble Sort

      Bubble Sort는 O(n^2) 시간이 걸리는 정렬 알고리즘으로, 원소들을 옮기는 과정이 마치 거품이 액체 위로 떠오르는 것 같다 하여 이름이 붙여졌다.

      한 iteration마다 배열의 아래부터 위로 순회하며 두 원소를 비교하며 더 큰 값을 위쪽에 오도록 옮긴다. 이렇게 하면 마지막에는 배열의 원소들 중 가장 큰 원소가 가장 위에 오게 된다. 이렇게 옮겨진 원소는 자기 값을 찾은 것이므로, 맨 위를 제외한 배열에 대해 똑같은 작업을 반복하면 정렬이 완료된다. 배열을 순회하는데 n만큼 시간이 걸리고, 이 작업을 원소를 하나씩 제외해가면서 n번 수행하므로 n^2이다.

   B. Insertion Sort

      Insertion Sort는 O(n^2) 시간이 걸리는 정렬 알고리즘으로, 원소들을 배열에 삽입하는 방식으로 작동하여 이름이 붙여졌다.

      배열의 맨 처음부터 시작하여, 아직 정렬되지 않은 부분의 가장 앞 원소를 골라 정렬된 부분의 알맞은 위치에 삽입한다. 이 과정에서 배열의 순회와 shifting이 일어나며, 이 과정에 n만큼 시간이 걸리고, 이를 n번 반복하므로 n^2이다.

   C. Heap Sort

      Heap Sort는 O(nlogn)시간이 걸리는 정렬 알고리즘으로, 최대힙을 사용해서 최댓값을 찾아나가는 알고리즘이다.

      먼저 주어진 배열을 Heapify하는 작업을 수행한다. 코드에서는 PercolateDown 알고리즘을 배열의 끝부터 차례대로 처음까지 반복하는 방식으로 구현하였다. 아래쪽부터 반복하므로 PercolateDown은 상수 시간이 걸리고(Heap 구조에 위배되지 않는 경우 재귀를 멈춘다.) 반복에 의해 n만큼 반복하므로 Heapify 파트의 시간복잡도는 O(n)이다.

      이후 만들어진 Heap에서 차례대로 최댓값을 뽑아내어 정렬되지 않은 배열의 끝에 삽입하는 과정을 수행한다. 이는 Swap의 방식으로 구현되며, 이로 인해 Heap의 root에 leaf 원소가 들어오면서 Heap 구조가 깨진다. 이 문제는 PercolateDown 알고리즘의 수행으로

해소되며, logn 시간이 소모된다. 이 과정을 원소의 개수만큼 반복하므로, 이 부분의 시간 복잡도는 O(nlogn)이다. 따라서 Heap Sort 알고리즘의 전체 시간 복잡도는 O(nlogn)이다.

D. Merge Sort

Merge Sort는 O(nlogn)시간이 걸리는 알고리즘으로, 배열을 나눠서 차례로 정렬된 형태로 합쳐가는 모습에서 이름이 유래되었다.

먼저 배열을 차례로 2개씩 나눈다.(이는 재귀를 이용하면 쉽게 구현 가능하다.) 이후, 가장 낮은 단계의 배열부터 차례로 배열의 정렬 상태를 유지하면서 합친다. 예를 들어, [1,3]과 [2,4]가 있으면 [1, 2, 3, 4]로 합쳐가는 방식이다. 이는 인덱스 변수를 두어서 수행할 수 있으며, 이렇게 하면 원소의 개수만큼 시간이 드므로 O(n)이다.

이 작업을 반으로 나눈 배열들에 대해서 반복하므로, T(n) = O(n) + 2 * T(n/2) 의 점화식을 통해 계산할 수 있다. 이를 풀면 T(n) = O(n) + 2O(n/2) + 4O(n/4) ... 의 형태로 진행되므로 O(n)이 log_2 n번 반복되는 것으로 해석할 수 있다. (n을 2의 거듭제곱의 형태로 생각하면 이해하기 쉽다.) 따라서 Merge Sort의 총 시간 복잡도는 O(nlogn)임을 알 수 있다.

E. Quick Sort

Quick Sort는 평균적으로 O(nlogn) 의 시간복잡도를 가지는 정렬 알고리즘이다.

먼저 무작위로 배열의 원소들 중 하나를 지정하고, 이를 pivot으로 삼는다. 그 다음, 배열의 모든 원소들에 대해 pivot보다 작은 원소들은 배열의 처음 쪽으로, pivot보다 큰 원소들은 배열의 끝 쪽으로 옮기는 작업을 수행한다. 이 다음 pivot을 두 그룹 사이에 배치하면, pivot은 자기 자리를 찾고 나머지 원소들로 이루어진 2개의 하위 배열이 생겨난다. 이들에 대해서 Quick Sort 알고리즘을 재귀적으로 적용하면 Quick Sort가 완성된다.

시간복잡도의 계산은 Merge Sort와 같은 방식으로 계산할 수 있다. pivot을 이용한 분류가 O(n) 이 걸리고 이를 2개로 나뉜 배열에 대해 반복하므로 O(nlogn)의 시간 복잡도가 나온다. 그러나 최악의 경우 pivot이 배열의 최댓값이나 최솟값으로 반복하여 선택되는 경우가 있을 수 있어 O(n^2)의 시간복잡도가 나올 수 있다.

F. Radix Sort

Radix Sort는 입력 배열의 원소들이 모두 정수 타입일 때 사용할 수 있는 특수한 정렬 알고리즘이며, O(n)의 시간복잡도를 가지고 있다.

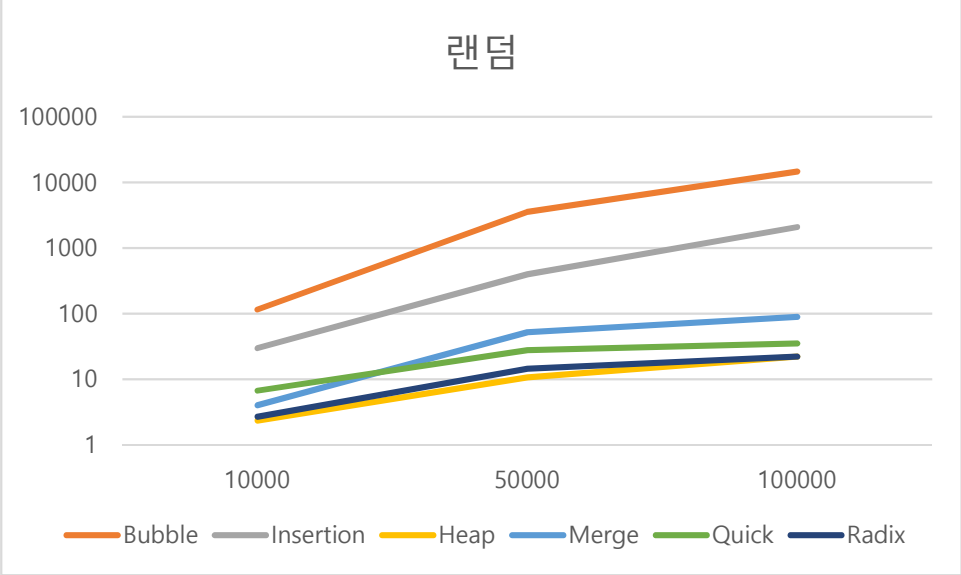
알고리즘의 내용은 배열들을 낮은 자리수의 숫자들에 대해서 정렬을 한 후, 점점 높은 자리수의 숫자들로 옮겨가며 정렬을 하는 것이다. 각 자리수의 숫자들에 대해서 정렬을 하는 것은 숫자의 값에 따라 자동적으로 수행할 수 있으므로(counting sort) O(n)이다. 이를 자리수의 개수에 따라 반복하는데, 자리수는 상수이므로 O(1) * O(n) = O(n)이다.

3. Experiment

실험은, 각 정렬 알고리즘에 대해서 각각 길이가 다른 랜덤 배열 3개, 정렬되어있는 배열 3개, 반대로 정렬되어 있는 배열 3개를 정렬하도록 하였다. 시간의 측정은 뼈대 코드 안의 내장되어 있는 시간 측정을 조금 변형하여 배열을 수동으로 입력해줄 때도 측정할 수 있도록 하였다. 편차를 보완하기 위해 같은 조건에 대해 3번씩 실험 후 평균을 내었다. 정렬된 배열은 python 스크립트로 만들어 파이프라인으로 전달하였다.

R 배열은 랜덤, S 배열은 정렬된 배열, RS 배열은 거꾸로 정렬된 배열이다.

배열	원소 수	Bubble	Insertion	Heap	Merge	Quick	Radix
R1	10000	115	29.6667	2.33333	4	6.66667	2.66667
R2	50000	3566.67	396.667	10.6667	52	27.6667	14.3333
R3	100000	14593.7	2082	22	88.6667	35	22
S1	10000	26	0.66667	2	2.33333	5	3
S2	50000	305.667	2.33333	9.33333	50.3333	32.6667	15.3333
S3	100000	1174.67	3.33333	22	80	43.3333	19.3333
RS1	10000	40	38.3333	1.66667	2.66667	5	4.33333
RS2	50000	685.333	570	10	55.3333	31.6667	11
RS3	100000	2758.67	2340.33	19	76.3333	37.3333	26



4. Result

랜덤 배열에 대해서, 알려진 시간 복잡도 대로 시간이 소모되는 모습을 보였다. 다만 실험에 사용한 배열의 크기가 극단적으로 크지 않아 Radix Sort의 시간 복잡도가 다른 nlogn 알고리즘과 다른 점이 잘 나타나지 않았다.

정렬된 배열에 대해서는, 대체로 랜덤 배열보다 더 빠른 속도를 보였으나, Heap Sort, Quick Sort, Radix Sort의 경우에는 유의미한 차이가 나타나지 않았다. 그리고, Insertion Sort는 급진

적으로 빠른 속도를 보였는데, 이는 Insertion 과정이 뒤에서부터 앞으로 찾아가는 방식이므로 항상 상수시간 만에 Insertion이 완료되어 전체 시간 복잡도가 O(n)이 됨으로써 나타나는 현상으로 보인다.

거꾸로 정렬된 배열들에 대해서는 Bubble Sort가 랜덤 배열에 비해 훨씬 더 빠른 모습을 보였고, 나머지는 비슷한 시간이 걸렸다. 다만 Heap Sort에서 아주 조금 더 빨라진 모습을 보였는데 이는 Heapify 과정에서의 시간이 다소 단축되기 때문으로 추정된다. Bubble Sort가 빨라진 이유는 항상 비교의 결과가 같아 JVM이나 Intel CPU에서 Branch Prediction을 하기 때문으로 생각된다.

5. Conclusion

유명한 정렬 알고리즘들을 구현하고, 이들의 특성을 실험적으로 알아 보았다. 다루는 배열의 특성에 따라 알고리즘을 다르게 설계하면 이득을 볼 수 있을 것이라고 생각된다.