**고급프로그래밍방법론 정렬 알고리즘 구현**

전기정보공학부 2016-18829 이성민

1. 정렬 알고리즘 수행 시간

Sorting algorithm의 수행 시간은 아래 그래프와 같다.

Insertion sort, merge sort, quick sort(마지막 pivot을 사용), quick sort with median(linear quick sort), heap sort, stooge sort로 총 6가지의 sorting algorithm을 사용하였다. 가로 축은 좀 더 쉽게 알아보기 위해 log scale로 나타내었다. 5~6분 이상이 걸리는 경우에는 데이터 수를 더 이상 늘리지 않고 측정을 그만두었다.

Stooge sort의 수행 시간이 가장 빠르게 증가하였고, insertion sort가 그 뒤를 따랐다. 이는 계산한 time complexity의 추세와 일치한다. Stooge sort는 약 의 time complexity를, insertion sort는 의 time complexity를, merge, heap sort는 의 time complexity를 가지며 quick sort 역시 평균적으로는 의 time complexity를 갖는 것으로 이론적으로 증명 가능하다. Stooge sort와 insertion sort를 제외한 알고리즘은 data의 수가 10,000,000개일 때부터 유의미한 값을 보였다. Heap sort – Linear Quick – Merge sort - Random quick 순서대로 점점 소요되는 시간이 짧아졌다. Linear quick 은 linear한 시간 내에 median을 찾는 알고리즘을 포함하는 quick sort인데, median value를 찾는 시간이 overhead로 작용해 이와 같은 결과가 나온 것으로 보인다.

2. Quick Sort의 Worst case와 random case의 수행 시간

Quick Sort이 worst case는 기존의 target array가 완벽하게 반대로 정렬 혹은 원래 방향대로 정렬되어 있을 경우이다. 두 경우 모두 마지막 항을 pivot으로 놓으면 해당 pivot을 기준으로 0개오 n-1개로 나뉘어지기 때문에 worst case이다. 이들 두 경우와 함께 기존 방식대로 random array의 마지막 node를 pivot으로 선택하였을 때의 수행 시간을 측정 및 비교하였다. 해당 결과는 아래 그래프와 같다.

Quick sort의 이론적인 time complexity는 O(n^2)로 평균적인 time complexity보다 큰 값을 갖는다. 위 그래프에서도 그와 같은 추세를 확인할 수 있었다. 정방향으로 정렬된 배열을 넣어준 initialize가 더 오랜 시간이 걸리는 까닭은 구현에 있어 pivot보다 값이 작으면 swap을 해주기 때문으로 보인다.

3. Stooge sort의 수행 시간의 recurrence equation과 time complexity

Stooge sort는 가장 왼쪽 node와 가장 오른쪽 node의 값을 비교하여, 왼쪽 node의 값이 더 큰 경우 두 값을 swap한다. 만일 l이 r 이상일 경우에는 다루고 있는 항이 하나 밖에 없는 경우이므로 그 항을 return한다. 이 과정에서는 O(1), 즉 constant time이 소모된다. 그 후, 기존 array size 중 앞쪽 2/3에 대해 stooge sort를 recursive 진행한다. 그 후, 뒤쪽 2/3와 앞쪽 2/3에 대해서도 차례대로 recursive하게 진행한다. 기존 array의 데이터 수를 n개라고 하면 각 2/3 크기에 대해 stooge sort를 하는 과정은 T(2/3\*n)만큼의 시간이 소모되며, 3차례 진행하므로 3T(2n/3)이라 표기할 수 있다. 이를 정리하면 stooge sort의 recurrence equation은

로 쓸 수 있다. O(1)은 1로 표기하였다. 이를 풀면 stooge sort의 time complexity 역시 구할 수 있다.

4. Stooge sort의 correctness proof

Stooge sort의 correctness proof는 다음과 같다.

1) Base case

Data size n=1인 경우, 항이 하나 뿐이므로 항상 sorting 되어 있다고 생각 가능하다.

Data size n=2인 경우, 가장 왼쪽 항과 오른쪽 항을 비교하고 왼쪽 항이 큰 경우 swap을 하므로 결과로 얻은 array는 sorting되어 있을 것이다.

2) Induction

Data size 인 경우 성립함을 보이자.

에 대해, n=1, 2, …, k-1일 때 stooge sort과 제대로 기능한다고 가정하고, n=k일 때에도 제대로sorting함을 보이겠다.

크기 k의 array에 대해 stooge sort를 적용시키면, 우선 가장 왼쪽 항과 가장 오른쪽 항을 비교해 더 작은 항이 왼쪽에 오도록 한다. 그 후, 왼쪽 2/3, 즉 개의 node에 대해 stooge sort를 적용시킨다. 가정에 의해 해당 node들은 제대로 sorting이 될 것이다. 이 때, sorting된 결과물의 첫 번째 절반, 즉 개의 수는 모두 적어도 자신보다 큰 수가 개는 존재하는 수들이다. 즉, 전체 node들 중 작은 의 portion에 속하는 항 만이 있는 것이다. 다르게 말하면 전체 node들 중 큰 의 portion에 속하는 항은 앞쪽 1/3에는 존재하지 않음을 알 수 있다. 따라서, 전체 node 중 큰 의 부분에 속하는 원소들은 모두 array의 뒤 2/3 부분에 속해 있을 것이다.

Stooge sort algorithm을 따라가면 뒤 2/3 부분에 대해 stooge sort를 다시 돌리게 된다. 이 때, 2/3k개로 구성된 배열에 대해서는 stooge sort가 올바르게 동작함이 보장되어 있으므로, 전체 원소들 중 큰 1/3은 모두 맞는 자리에 들어있을 것이다.

그 후, 다시 앞 2/3 부분에 대해 stooge sort를 적용시키면, 작은 2/3에 속하는 원소들에 대해 stooge sort를 돌리는 것이고 가정에 의해 맞게 정렬된다.

따라서, n=1,…,k-1일 때 stooge sort가 제대로 기능한다면, n=k일 때 역시 효과적으로 작동함이 증명되었다.

n=1, 2일 경우에는 제대로 동작함을 알고 있으므로 n=3, 4, 5, …일 때 stooge sort가 제대로 배열을 정렬한다는 것을 증명하였다.