서규혁, 2018-10786

I. 정렬 알고리즘 동작 방식

1. Bubble Sort

코드와 함께 설명하자면 1 번째 원소와 2 번째 원소를 비교하여 더 큰 수가 2 번째원소에 오도록 하고, 2 번째 3 번째 원소에 대해 진행, 그리고 3 번째 4 번째 원소에 대해, 계속 진행하여 (length-1)번째 (length)번째 원소에 대해 진행하면 1 회전이 종료되는 것이다. 1 회전의 결과 가장 큰 원소가 (length)번째 원소로 배치되어 있다. 만약 1 회전을 진행하면서 단 한번의 swap 과정도 일어나지 않은 경우에만, 오름차순으로 이미 정렬된 배열의 경우, 1 회전 종료 후 is_swap 이 false 이므로 이 경우는 바로 배열을 출력하기 위해 break 가 있는 것이다. is_swap 이 True 인 경우, 1 회전 종료 후 1 번째부터 (length-1)번째 원소에 대해 1 회전에서 했던 작업을 계속하며, 총 (length-1)회전까지 진행하게 되면, 뒤에서부터 원소들이 정렬되어 채워지면서, 오름차순으로 정렬된 배열이 완성된다.

2. Insertion Sort

두 번째 원소부터 시작하며, 자신보다 앞에 있는 원소들과 자신의 값을 비교하여 자신의 위치에(여기서는 오름차순에 따른 자신의 위치) 맞게 삽입되어 정렬되는 알고리즘이다. 두 번째 원소는 첫 번째 원소와 자신을 비교하여 정렬하게 된다. 세번째 원소입장에서 보자. 1~2 번째 원소끼리는 정렬되어 있기 때문에 2 번째원소부터 자신과 비교하기 시작한다. 비교 후에 자신의 위치를 찾고 알맞게 삽입된다면 1~3 번째 원소는 정렬된 상태를 유지한다. 즉 i 번째 원소를 알맞은 위치에 삽입시키고자 할때, 1~(i-1)번째 원소까지는 정렬된 상태를 유지하고 있다. i 번째 원소 값인 target 을 (i-1)번째 원소부터 1 번째 원소까지 비교하게 되며 target 값보다 작거나 같은 원소값이 나타나게 되면은 그 즉시 비교를 멈추고 해당 원소의 앞에 i 번째 원소를 삽입하게 된다. 이렇게 i 를 2 부터 (length)까지에 대해 진행해주면 value 배열은 정렬이 된다.

3. Heap Sort

Heap Sort 는 두개의 추가적인 메서드를 이용하여 구현하였다. DobuildHeap 이라는 메서드는 배열의 원소들을 완전이진트리로 Max Heap 을 만드는 것이다. DopercolateDown 의 경우는 value[i]의 원소를 root 로 가지는 완전이진트리의 subtree 를 max heap 형태가 지켜지도록 하는 것이다. 이러한 두 메서드를 기반으로 DoHeapSort 에 대해 설명하자면, 처음 input 인 value 라는 배열에 대해 1 번째 원소부터 (length)번째 원소까지 DobuildHeap 을 통해 완전이진트리로 Max Heap 형태가 되도록 한다. 그 후 첫번째 원소(현재 완전이진트리 Max heap 형태에서 root 에 해당하는 값이므로 가장 큰 값이다)와 마지막 원소를 swap 하고, 마지막 원소를 제외한 1 번재 원소부터 (length-1)번째 원소에 대해, 1 번째 원소를 기준으로 DopercolateDown 한다. 기존에 DobuildHeap 을 하였으므로 그 후 움직임이 있었던 1 번째 원소를 기준으로만 DopercolateDown 을 하는 것이다. 그 후, 다시 1 번째 원소에 (length-1)번째 원소를 바꾸고, 이번엔 1 번째 원소부터 (length-2)번째 원소에 대해, 1 번째 원소를 기준으로 DopercolateDown 한다. DopercolateDown 하는 배열의 끝을 점점 줄여나가면서 진행하게 되면, 배열의

뒷부분부터는 큰 원소들이 쌓일 것이며 결국 2 번째 원소에 대해서까지 진행하게 되면 value 배열이 정렬된 상태로 바뀌게 된다.

4. Merge Sort

간단히 설명하면 리스트의 길이가 1 이하인 경우에는 정렬된 상태로 보고, 2 이상인 경우 비교적 균등하게(원소갯수가 홀수이면 1 차이나게, 짝수이면 동일하게) 두 부분 배열로 분할한다. 그리고 이 부분 배열들을 각각 정렬하고, merge 를 통해 전체가 정렬되도록 하는 것이다. 메서드 msort 는 길이가 2 이상의 배열에 대해 B 정렬을 비교적 균등하게(위에서 설명) p 부터 q, q+1 부터 r 로 나누고, 이 부분 배열을 각각 정렬한 다음, 메서드 merge 를 실행하는 것이다. 메서드 merge 는 두 부분 배열에 대해 각각 정렬되어있는 C 정렬을 D 배열에 합쳐 전체가 정렬되도록 하는 것이다. DoMergeSort 에서 msort 결과 value 에 최종 정렬된 배열이 저장되므로 return value 를 통해 merge sort 결과를 리턴한다.

5. Quick Sort

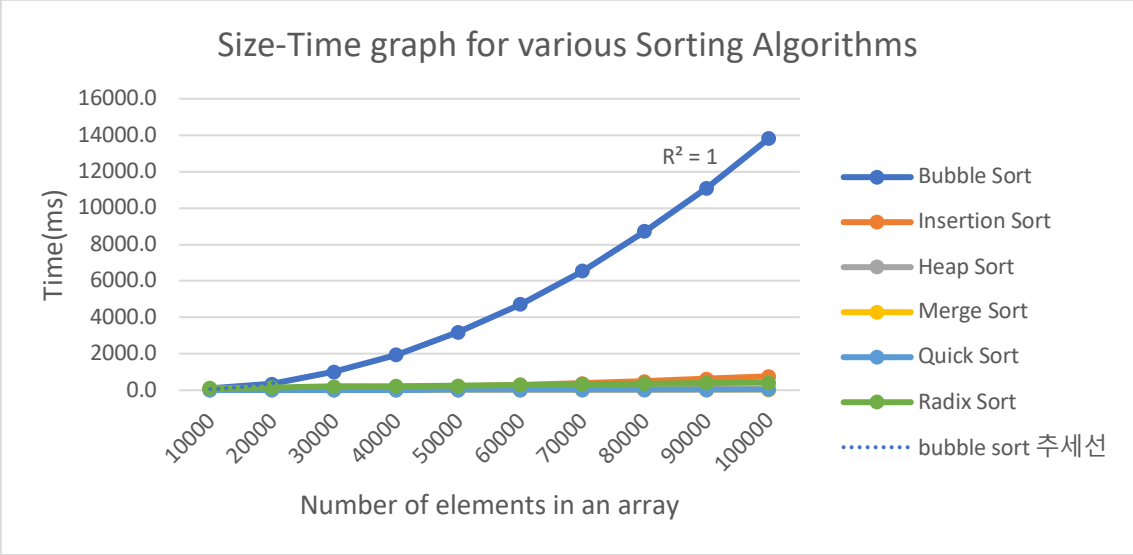
DoQuickSort 메서드는 qSort 를 호출하므로 qSort 를 살펴보자. qSort 를 살펴보기에 앞서 partition method 를 먼저 설명하겠다. Input 의 배열인 value 에 대해 마지막 원소를 pivot 으로 설정하고 pivot 이하의 원소들과 pivot 초과하는 원소들로 value 배열을 나누는 것이다. 메서드 partition 을 진행하면 value[q]에는 pivot 값이 저장되며 value[p]부터 value[q-1]에는 pivot 이하의 값들이 모이고(정렬여부는 알수없음), value[q+1]부터 value[r]에는 pivot 초과의 값들이 모인다(마찬가지로 정렬여부는 알수없음). 두 부분을 각각 정렬하게 되고 이를 합하면 전체 배열이 정렬되게 된다.

6. Radix Sort

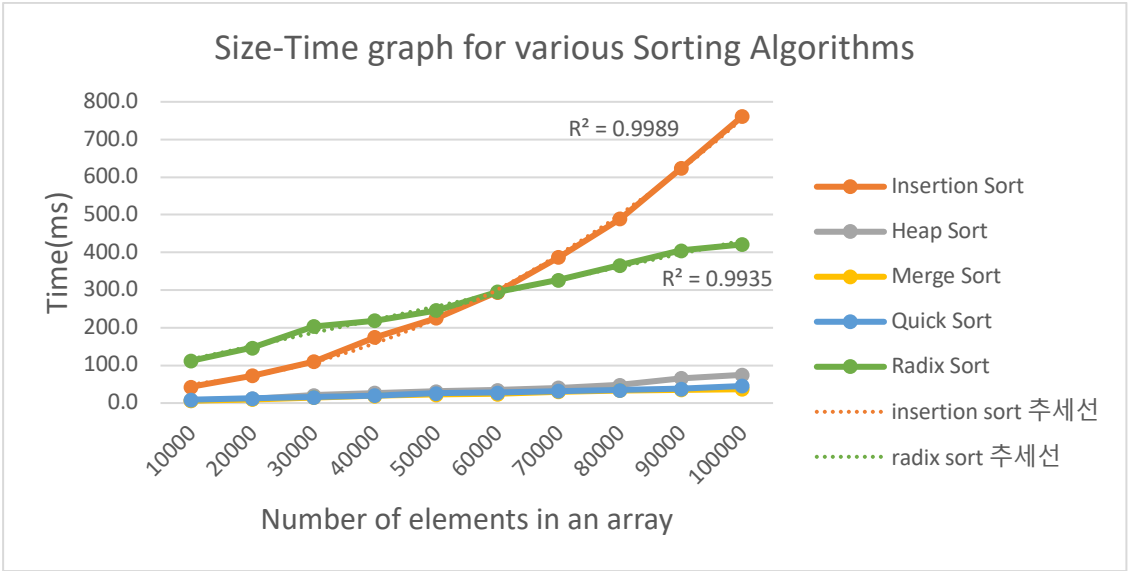
메서드 plus_minus 는 위에서 설명한 partition 과 비슷한 작업인데 0 을 pivot 으로 설정하여 index p 부터 r 까지의 value 원소들에 대해 0 보다 작은 값들과 0 이상의 값들로 나누고(마찬가지로 각각 정렬여부는 알 수 없음) i 를 리턴하는데 value[i]부터는 0 이상의 값들, index 가 i 미만인 value 의 원소들은 음수임을 알려준다. 메서드 plus_minus 를 통하여 value 배열을 음수와 0 이상의 값들로 나눠져 모이게 한다. 그리고 plus 배열과 minus 배열을 만들어 음수인 elements 들은 절대값들을 minus 배열에, value 배열의 0 이상의 elements 들은 plus 배열에 저장한다. 각각 minus, plus 배열에 대해 radix sort 에 필요한 자릿수를 구하기 위해 min, max 를 이용하여 자릿수를 구한다. 그리고 minus 배열에 대해 rSort, plus 배열에 대해 rSort 를 수행한다. 여기서 rSort 의 역할을 보면, rSort 에 들어오는 배열들은 결국 0 이상의 정수를 갖는 elements 들이고 input 으로 자릿수가 주어지며 각 자리에는 0~9 까지가 가능하다. 일의 자리부터 Counting Sort 를 진행하며 최대자리수까지 Counting Sort 를 각 자리수별로 진행하게 되면, input 으로 들어온 배열에 대해 오름차순으로 배열이 정렬된다. 배열 plus 의 경우는 rSort 를 거치면 오름차순으로 정렬되며, 배열 minus 는 절댓값이 저장된 배열이며 이들이 오름차순으로 정렬되어있기 때문에 -1 을 곱하고 역순으로 저장해야 음수의 값들이 오름차순으로 정렬된다. 이후 minus 뒤에 plus 를 합친 전체 배열을 return 하면 Radix Sort 를 통한 전체 배열이 정렬된다.

II. 동작 시간 분석

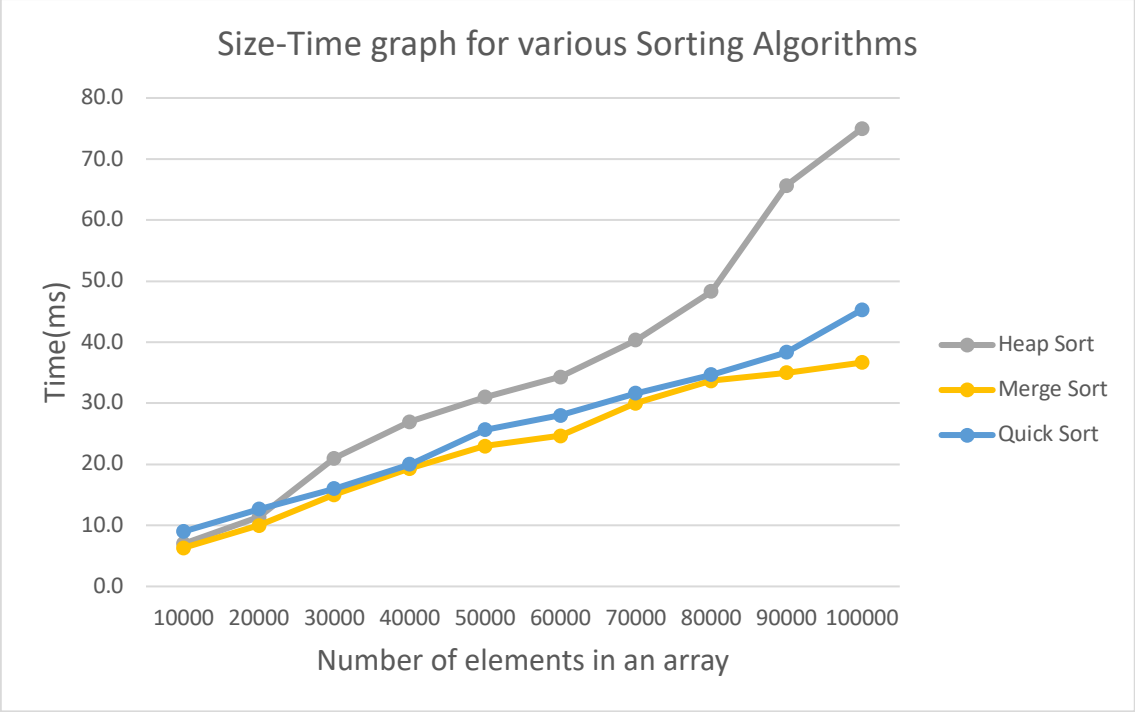
Sorting Algorithms 별로 array 의 elements 갯수에 따른 수행시간을 측정하였다. Elements 의 갯수는 10,000 개부터 100,000 개까지 10,000 개씩 늘리며 총 10 그룹으로 진행하였고 한 그룹 당 실험은 10 번씩 진행하여서 총 100 번 측정하였다.



위의 그래프를 보면 number of elements in an array 가 증가함에 따라 bubble sort 에 소요되는 시간이 증가하는 것을 볼 수 있고 이차식의 추세선의 R 제곱값이 0.9996 으로 거의 1 인 것으로 보아 bubble sort 는 평균 시간 복잡도가 n^2 에 비례하는 것을 알 수 있다.



위의 그래프를 보면 Insertion sort, Radix Sort 에 대해 각각 이차식의 추세선, 일차식의 추세선의 R 제곱값이 0.9989, 0.9935 로 0.99 를 넘는 값이므로 Insertion sort, Radix Sort 의 평균 시간 복잡도가 각각 n^2, n 에 비례하는 것을 알수있다.



이론적으로 heap sort, merge sort, quick sort 는 평균 시간 복잡도가 nlogn 에 비례한다고 알려져있다. 정렬에 걸린 시간을 측정한 결과 data 의 개수가 증가함에 따라 걸린 시간도 선형 이상에 비례하여 증가하는 것을 파악할 수 있다.

(단위 : ms)	MAX.	AVG.	MIN.	SD.
Bubble Sort	14016.0	13719.1	12764.0	426.9
Insertion Sort	766.0	761.1	757.0	3.6
Heap Sort	99.0	69.9	44.0	24.8
Merge Sort	40.0	36.1	32.0	3.0
Quick Sort	43.0	36.7	32.0	3.9
Radix Sort	437.0	428.7	410.0	9.2

위 표는 데이터의 갯수가 100,000 개일때 각 sorting algorithm 별로 10 번씩 진행하여 나온 결과이다. 여기서 주목할 점은 표준편차가 가장 작은 MergeSort 의 경우인데 표준편차가 가장 작다는 점은 길이가 같은 array 에 대해 걸리는 시간이 거의 동일하다는 뜻이며 실제로 merge sort 는 모든 경우에 대해 time complexity 가 theta(nlogn)이므로 이를 보여주는 결과라고 볼 수 있다.