Sorting 보고서

2017-15108 박지상

1. Bubble Sort

버블 정렬은 배열 안의 인접한 두 수를 비교하여 가장 큰 수를 맨 뒤로 보내는 방식으로 이루어지는 정렬 알고리즘이다. k번째 원소와 k+1번째 원소를 비교하였을 때, k+1번째 수가 k번째 수보다 작다면, 두 수를 교환(swap)하고 작지 않다면 교환하지 않은 채 그 다음 k+1번째 원소와 k+2번째 원소를 비교한다. 비교 범위의 가장 마지막 두 개의 원소의 비교가 끝난 경우 가장 마지막 원소를 정렬이 완료된 것으로 보고 비교 범위에서 제외하며 처음부터 다시 반복한다. 이렇게 반복하여 모든 원소가 비교 범위에서 제외되면 정렬을 완료한다.

값의 개수	최솟값	최댓값	평균	표준편차
10000	226	356	261.7	38.53
30000	2102	2674	2302	194.15
70000	13438	16615	15079.5	1221.47
100000	23966	25623	24670.9	532.53

각 10회 난수 범위: -1000000000 이상 1000000000 이하

2. Insertion Sort

삽입 정렬은 왼쪽에서부터 k개의 원소가 정렬된 상태에서 k+1번째 원소를 대상으로 정렬이 이루어지는 식으로 진행된다. k+1번째 원소는 1번째 원소부터 k번째 원소까지 비교하면서 오름차순에 맞게 자신이 삽입될 위치를 발견하였을 때, 해당 위치의 원소부터 k번째 원소까지 한 칸씩 오른쪽으로 한 칸씩 옮기고 자신을 삽입하여 결과적으로 k+1개의 원소가 오름차순으로 정렬된 상태로 된다. 똑같은 과정을 k+2번째 원소에 대해서도 진행하면서 결과적으로는 n개의 원소 모두 정렬된 상태가 될 때까지 진행한다.

개수	최솟값	최댓값	평균	표준편차
10000	53	229	121.6	68.99
30000	386	583	488.8	61.74
70000	3148	4368	3512	385.14
100000	4610	5439	5132.4	214.96

각 10회 난수 범위: -1000000000 이상 1000000000 이하

3. Heap Sort

힙 정렬은 입력된 n개의 원소에 대해서 먼저 maxheap 구조 (완전 이진 트리에서 부모노드의 값이 항상 자식노드들의 값보다 큰 형태)를 형성한다. 이렇게 형성된 maxheap구조에서 root에 위치한 값과 마지막 노드의 key를 교환하여 최댓값을 담게 된 last node는 정렬 대상에서 제외한 뒤, n-1개의 노드를 갖는 이진트리를 또 다시 maxheap 구조로 변환한다. 이런 일련의 과정을 거쳐서 모든 노드가 정렬 대상에서 제외되면 정렬이 완료된다.

이때 maxheap 구조를 만들기 위해서 percolateDown이라는 방식을 사용하는데, 이는 input으로 받은 노드에서부터 시작하여 자신의 자식 노드들과 값을 비교한 뒤, 부모보다 값이 큰 자식이 있는 경우 최댓값을 갖고 있는 자식과 교환한다. 이렇게 교환이 더 이상 발생하지 않을 때까지 계속해서 반복한다.

개수	최솟값	최댓값	평균	표준편차
10000	2	19	8.4	4.48
30000	6	25	12.4	8.03
70000	14	28	19.5	3.98
100000	24	116	65.7	37.58

각 10회 난수 범위: -1000000000 이상 1000000000 이하

4. Merge Sort

합병 정렬은 n개의 원소에 대해서, 절반의 크기로 극한까지 분할하다가 극한(1개의 원소)에 다다르면 오름차순의 순서에 맞게 합병하면서 최종적으로 n개의 원소에 대해서 정렬된 원소로 만든다.

개수	최솟값	최댓값	평균	표준편차
10000	3	21	12	6.51
30000	8	55	28.3	17.17
70000	27	64	38	13.24
100000	36	142	91.2	40.28

각 10회 난수 범위: -1000000000 이상 1000000000 이하

5. Quick Sort

퀵 정렬은 기준이 되는 원소(pivot)를 정해놓고 기준 원소와의 대소 관계 비교를 통해서 더 큰 원소와 작은 원소를 두 파티션으로 구분한다. 그렇게 구분된 파티션 각각에 대해서도 또다시 기준 원소를 잡고 파티션을 진행하며, 극한(1개의 원소)에 다다르면 그것을 작은 파티션과 기준 원소와 큰 파티션을 순서대로 합병하면서 최종적으로 n개의 원소에 대해서 정렬된 원소로 만든다. 이 때 기준이 되는 원소를 정하는 방법은 매우 다양하지만, 여기서는 배열의 첫 번째 원소로 설정하였다.

개수	최솟값	최댓값	평균	표준편차
10000	3	23	11	5.37
30000	9	63	28	17.08
70000	17	99	25.3	17.24
100000	32	132	80.9	33.94

각 10회 난수 범위: -1000000000 이상 1000000000 이하

6. Radix Sort

기수 정렬은 낮은 자릿수부터 비교하여 가장 높은 자릿수를 갖고 있는 숫자의 자릿수까지 비교하여 진행한다. 자릿수를 기준으로 비교할 때, 각 자리의 값 0부터 9에 해당하는 Queue를 10개 만들어 모든 원소가 enqueue될 때 까지 진행한다. 이는 원소들 간의 상대적 순서를 보존하여 안전한 정렬(stable sort)이 이루어지게 하기 위함이다. 이후 자리의 값이 낮은 Queue부터 empty가 될 때까지 dequeue를 하면서 배열한다. 이를 반복하면서 최종적으로는 모두 정렬이 되게 된다.

이 때, 음수의 정렬을 위해서 초반에 음이 아닌 정수 와 음의 정수를 구분하여 배열을 만들고, 음의 정수에 대해서는 음의 정수 중 최솟값의 절댓값을 모든 원소에 더한 뒤, 음이 아닌

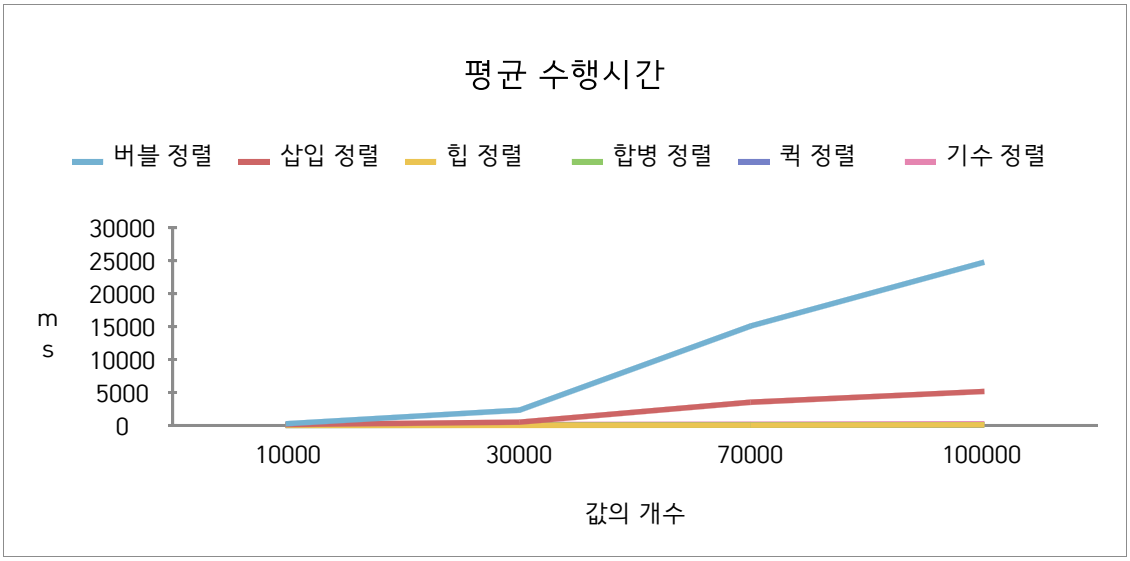
정수와 동일한 방식으로 배열하고 마지막에 모든 원소에서 더해주었던 절댓값을 다시 빼주는 형식으로 진행한다. 마지막에는 음의 정수의 집합과 음이 아닌 정수의 집합을 더해 리턴한다.

개수	최소값	최대값	평균	표준편차
10000	19	125	48.9	32.26
30000	39	178	111.8	42.94
70000	116	217	155.8	35.27
100000	175	328	249.2	53.59

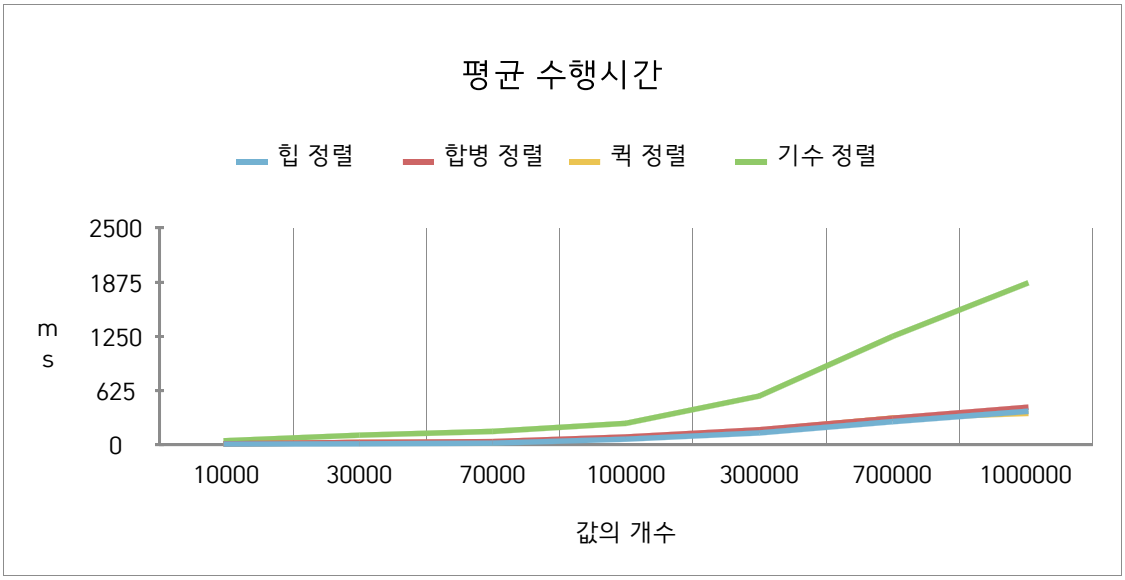
각 10회 난수 범위: -1000000000 이상 1000000000 이하

7. 정렬 방법 간 비교

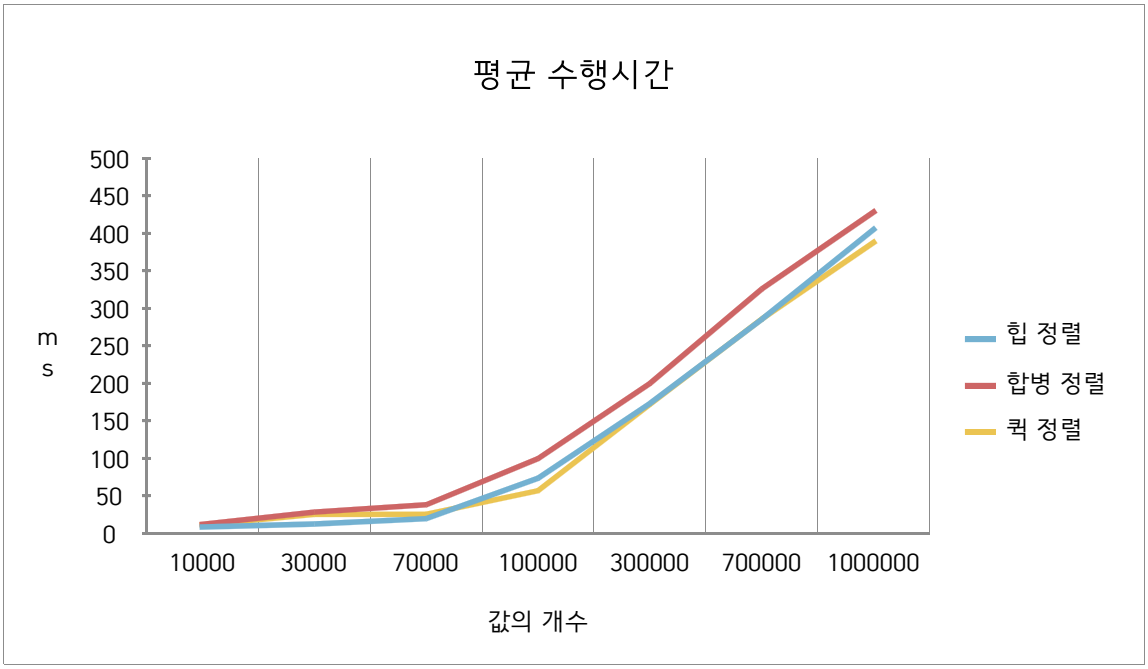
Data는 모두 r (값의 개수) -1000000000 1000000000를 바탕으로 총 10회 수집하였다.



수행시간이 버블 정렬 > 삽입 정렬 > 그 외의 정렬 순인 것은 분명하지만, 나머지 4개의 정렬(힙, 합병, 퀵, 기수)는 유의미한 차이가 보이지 않았기에 이 4가지 정렬을 바탕으로 다시 분석하였다.



이에서 기수 정렬이 나머지 3개의 정렬 (힙, 합병, 퀵)보다는 훨씬 수행시간이 긴 것은 분명하지만, 나머지 3개 정렬에서 유의미한 차이가 보이지 않았다. 따라서 3개의 정렬을 더 엄밀하게 비교하기 위해서 값의 개수가 100000개 이상의 경우에 대해서 표본을 각각 100회로하여 자료를 추가 수집하여 비교하였다.



개수	힙 정렬	합병 정렬	퀵 정렬
10000	8.4 (4.48)	12 (6.51)	11 (5.37)
30000	12.4 (8.03)	28.3 (17.17)	28 (17.08)
70000	19.5 (3.98)	38 (13.24)	25.3 (17.24)
100000	73.39 (32.91)	99.58 (36.09)	56.79 (37.22)
300000	173.13 (60.72)	199.74 (84.54)	172.05 (69.87)
700000	285.26 (61.09)	335.91 (71.14)	257.96 (45.89)
1000000	405.27 (67.03)	428.42 (69.18)	387.85 (65.43)

각 10회 난수 범위: -1000000000 이상 1000000000 이하

*괄호 안의 숫자는 표준편차

분석 결과, 합병 정렬은 수행 시간에서 힙 정렬 및 퀵 정렬과 유의미한 차이를 보였고, 힙 정렬과 퀵 정렬은 큰 차이는 없지만 대체적으로 퀵 정렬이 수행시간이 적게 나오는 것으로 관찰된다.

결론을 내리자면, 총 6가지의 정렬 방식의 수행시간을 비교할 때, 버블 정렬 > 삽입 정렬 > 기수 정렬 > 합병 정렬 > 힙 정렬 > 퀵 정렬인 것을 알 수 있다.