DS HW4

2018-11550 김민주

1. Implementation Environment

Operating System: Linux Mint 20.1

Kernel: Linux 5.8.0-63-generic

Processor: 11th Gen Intel© Core™ i7-1165G7 @ 2.80GHz × 4

2. Method

2-1 Bubble Sort

Bubble Sort는 첫 번째 원소부터 시작해서, i번째 원소와 i+1번째 원소를 비교한 뒤, i번째 원소가 i+1번째 원소보다 크면 자리를 바꾼다. 이렇게 하면 가장 큰 원소는 배열의 가장 끝에 오게 될 것이다. 이렇게 정렬된 가장 큰 원소를 관심 대상에서 제외한 뒤, 똑같은 작업을 반복해 두 번째로 큰 원소를 뒤에서 두 번째 자리에 놓는다. 관심 대상이 되는 원소가 한 개가 될 때까지 위의 작업을 반복하면 정렬이 완료된다. 이 알고리즘은 $O(n^2)$ 이 걸린다.

2-2 Insertion Sort

Insertion Sort는 첫 번째 원소부터 시작해서, index가 하나씩 늘어날 때 마다 A[0...i]의 적당한 자리에 A[i]를 삽입하는 방식으로, 이렇게 하면 A[i]까지의 원소는 정렬된 상태이다. 이러한 작업을 배열의 끝까지 하면 정렬이 완료된다. 위의 Bubble Sort는 관심 대상이 되는 원소를 하나씩 줄여 나가는 방식이라면, Insertion Sort는 정렬된 배열의 크기가 하나씩 늘어나는 알고리즘이다. 이 알고리즘은 $O(n^2)$이 걸린다.

2-3 Heap Sort

Heap이란 complete binary tree이며, 각 node의 element 값이 children node의 element 값보다 크거나 같다는 특징을 가지고 있는 자료구조이다. Heap Sort는 heap을 이용한 알고리즘이다. 먼저 배열을 heap으로 만들고, root node, 즉 element의 값이 가장 큰 node와 제일 마지막 node의 자리를 바꾼 뒤, 제일 마지막 node를 관심 대상에서 제외하고 다시 heap을 만든다. 관심 대상이 되는 원소가 한 개가 될 때까지 이 작업을 반복하면 정렬이 완료된다. 이 알고리즘에서, heap을 다시 만드는 percolate down이 최대 트리의 높이, 즉 O $O(\log n)$의 시간이 걸리므로, 이 알고리즘은 평균 $O(n \log n)$이 걸린다.

2-4 Merge Sort

Merge Sort는 배열의 원소가 하나씩 남을 때까지 이등분한 뒤, 정렬하면서 병합하는 알고리즘으로, 대표적인 재귀 알고리즘이다. 예를 들면, [4, 2, 3, 1]의 배열이 있으면, [4, 2], [3, 1] 두 개의 배열로 나누고, 또 각각 [4] [2], [3] [1] 로 나눈 뒤, [2, 4]와 [1,3]으로 정렬하며 병합한 후, [1, 2, 3, 4]로 정렬하며 병합하는 방식이다. 배열을 나누는 것은 곧, 크기가 자신의 절반인 배열 두 개를 불러오는 것과 같고, 정렬을 하는 데에는 n에 비례하는 시간이 드므로, T(n) = 2*T(n/2) + O(n)으로 수행시간을 계산할 수 있으며, 수행시간은 $O(n \log n)$이다.

2-5 Quick Sort

Quick Sort는 배열의 원소들 중 하나를 pivot으로 잡은 뒤, pivot보다 작으면 pivot의 왼쪽, 크면 오른쪽에 다가 놓아준다. 그러면 pivot은 자기 자리를 찾은 것이다. pivot의 왼쪽 배열과 오른쪽 배열을 각각 재귀적으로 위와 같은 작업을 반복해주면 정렬이 완료된다. 본 과제에서는 pivot을 배열의 마지막 원소로 지

정하였다. 이 알고리즘의 수행 시간은 위의 Merge Sort와 같은 방식으로 계산할 수 있다. 이 알고리즘은 평균 $O(n \log n)$이 걸리지만, 이미 정렬 되어있는 배열이나 모두 같은 element로 이루어진 배열의 경우, pivot이 중간에 위치하지 않고 한 쪽 끝에 위치하게 되어, worst case $O(n^2)$이 걸릴 수 있다.

2-6 Radix Sort

Radix Sort는 모든 element가 k이하의 자릿수를 가지고 있는 자연수인 경우에만 사용할 수 있다. 가장 낮은 자릿수를 첫 번째 자릿수라 할 때, 첫 번째 자리부터 k번째 자리까지 i번째 자릿수에 대해서 안정성을 유지하면서 정렬하면 된다. 즉, 원소의 값이 같을 때, 정렬 후에도 원래의 순서를 유지시킨다. 대표적인 Stable sort로는 Counting Sort가 있다. 예를 들면, [59, 52, 61, 5]라는 배열이 있다면, 먼저 첫 번째 자리를 정렬하면 [61, 5, 52, 59]가 되고, 두 번째 자리에 대해 정렬하면 [5, 52, 59, 61]이 될 것이다. 자릿수는 상수이고, Counting Sort는 O(n)이 걸리므로, 이 알고리즘의 수행시간은 O(n)이다.

3. Analysis

적당히 큰 n에 대해 평균 수행시간을 구했다. 각 sorting 알고리즘에 대해서, n = 10000, 25000, 50000, 75000, 100000 이고, $-10^6 \leq n \leq 10^6$일 경우, 각각 100번 수행해, 평균 수행 시간을 구했다. 결과는 아래 표와 같고, 그래프로도 표현해보았다. (시간 단위: ms)

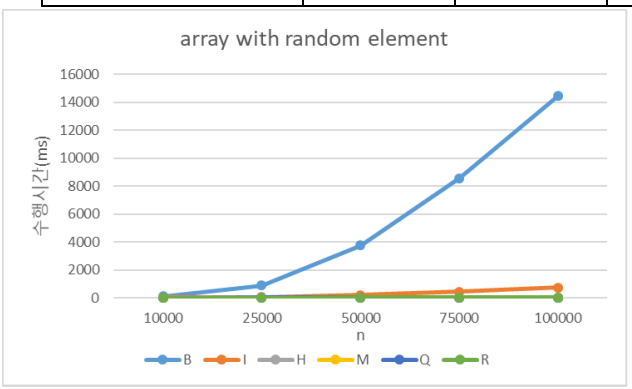
n=10000/100 회	B	I	H	M	Q	R
최댓값	126	16	2	2	1	5
최소값	76	7	0	1	0	0
평균	107.94	7.77	0.92	1.25	0.73	0.62
표준편차	9.184574	1.05693	0.337046	0.433013	0.443959	0.745386

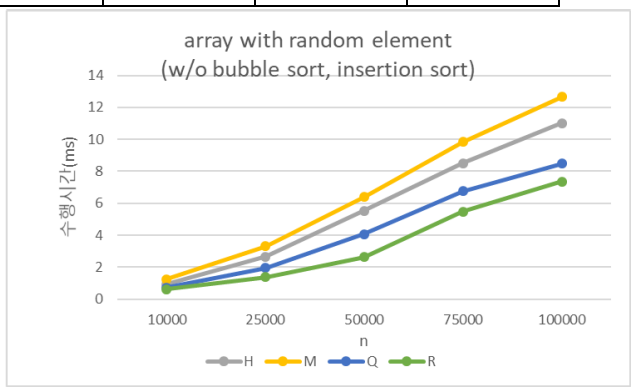
n=25000/100 회	B	I	H	M	Q	R
최댓값	903	57	6	17	4	5
최소값	858	46	2	2	1	1
평균	875.23	47.64	2.65	3.32	1.95	1.38
표준편차	8.104141	1.628005	0.606218	1.509172	0.357071	0.914112

n=50000/100 회	B	I	H	M	Q	R
최댓값	4074	206	10	15	8	9
최소값	3690	186	5	5	3	2
평균	3737.43	188.9	5.52	6.4	4.06	2.64
표준편차	52.3802	3.907685	0.699714	1.240967	0.50636	1.29244

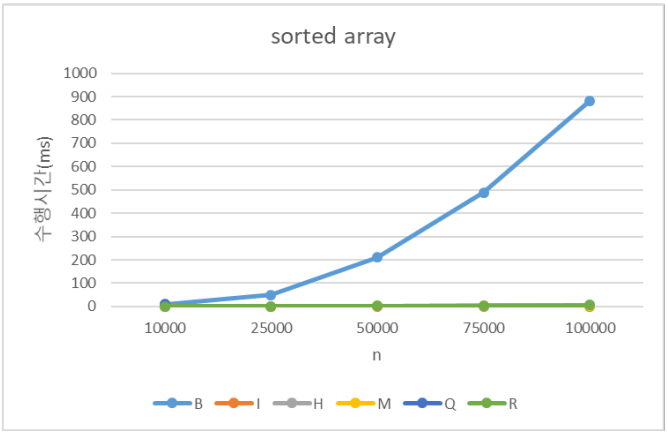
n=75000/100 회	B	I	H	M	Q	R
최댓값	9768	469	12	14	42	14
최소값	8466	419	8	9	6	3
평균	8561.77	429.33	8.52	9.85	6.76	5.47
표준편차	129.3543	13.37988	0.624179	1.071214	3.583629	3.278582

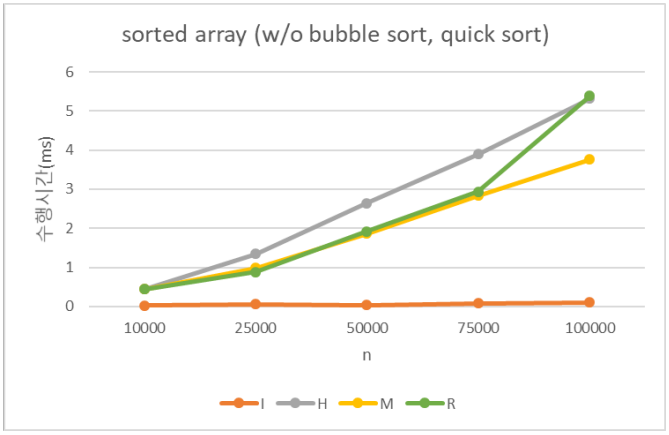
n=100000/100 회	B	I	H	M	Q	R
최댓값	17406	819	15	21	42	18
최소값	10801	600	8	10	6	3
평균	14432.29	731.13	11.01	12.65	8.49	7.35
표준편차	1586.094	59.79325	1.109009	1.971674	3.485671	4.348276

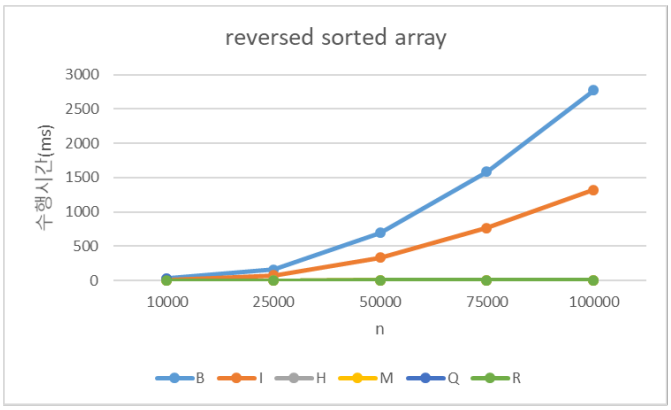


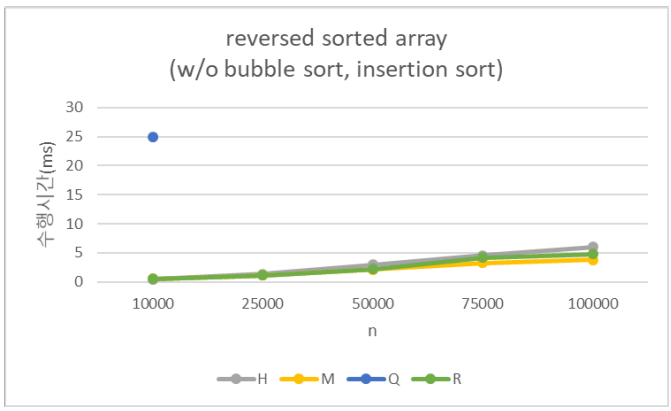


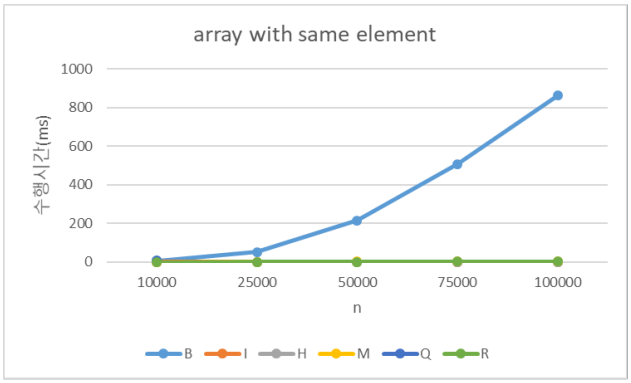
또한, average case 뿐만 아니라, best case와 worst case의 시간도 측정해보기 위해, 정렬되어 있는 배열, 역순으로 정렬되어 있는 배열, 같은 숫자로 이루어진 배열의 수행시간도 측정해보았다. 마찬가지로 n = 10000, 25000, 50000, 75000, 100000에 대해 수행하였으며, 각각 50번씩 수행하였다. 결과는 아래 그래프와 같다.

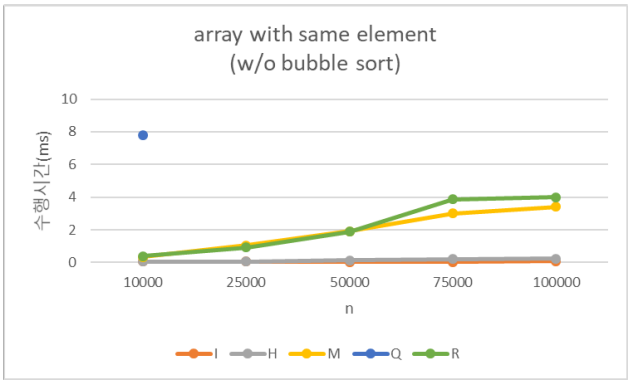












4. Discussion

4-1. Random array

Bubble Sort는 대략 $T(n) = 869.19n^2 - 1581.6n + 726.66\ (R^2 = 0.9992)$이고, Insertion Sort는 대략 $T(n) = 44.502n^2 - 84.172n + 43.946 (R^2 = 0.9995)$으로, $O(n^2)$의 수행시간이 걸리는 것을 확인할 수 있었다. 또한, 그래프를 통해 Heap Sort, Merge Sort, Quick Sort 또한 $O(n \log n)$이 걸리는 것을 확인할 수 있었다. 그 중에서도, Quick Sort가 가장 빨랐으며, 그 다음이 Heap Sort, 그리고 Merge Sort가 가장 느렸다. Radix Sort에 대해서는 다른 algorithm보다 빨랐지만 $O(n)$인지 $O(n \log n)$인지 구분하기 어려웠는데, 이는 input이 더 커지면 확인할 수 있을 것으로 보인다.

4-2. Sorted array

Insertion Sort가 압도적으로 빠른 것을 알 수 있는데, 이는 Insertion Sort의 best case로, iteration 당 한 번의 비교만 해도 돼서 수행시간이 $O(n)$이라 가능한 것으로 보인다. 또한, n이 25000 이상일 때부터 Quick Sort는 stack overflow가 났으며, n이 10000일 때는 수행 시간이 bubble sort보다도 길었다. 이는 partition의 불균형으로 인한 것으로 보인다.

4-3. Reverse sorted array

여기서의 Insertion Sort는 average case보다 약 2배 느린 $T(n) = 82.959n^2 - 166.25n + 87.832 (R^2 = 0.9995)$인것으로 확인되는데, 이는 Insertion sort의 worst case여서, 적당한 자리에 A[i]를 삽입하기 위해 비교를 끝까지 해야 하기 때문이다. 여기서는 n이 10000일 때 Quick Sort의 수행시간이 Bubble Sort와 비슷했으며, n이 25000 이상일 때 Stack Overflow가 났다.

4-4. 모든 원소가 같은 array

여기서의 Insertion Sort와 Heap Sort 모두 Best case로 $O(n)$의 시간이 걸렸다. Heap Sort의 경우, 이 때 배열이 이미 heap의 성질을 만족하고 있고, 가장 큰 node를 제거한 뒤에도 여전히 heap의 성질을 만족하고 있으므로 best case가 된 것이다. 여기 또한 n이 10000일 때 Quick Sort의 수행시간이 Bubble Sort와 비슷했으며, n이 25000 이상일 때 Stack Overflow가 났다.

4-5.

여기에서 이상했던 점은, Radix Sort의 수행 시간이 가끔은 Merge Sort보다도 느리다는 것이었다. 실험 데이터를 살펴보면 첫 번째 컴파일을 할 때 Radix Sort의 수행시간이 비정상적으로 느렸던 것과 관련이 있는 것으로 보인다. 또한, Bubble Sort는 정렬된 배열에서 랜덤 배열보다 훨씬 빠른 모습을 볼 수 있었는데, 이는 비교 결과가 항상 같아서 Brach Prediction이 잘 되어서 그런 것으로 보인다.