자료구조 HW4 Report

2021-13194 고재준

1. 정렬 알고리즘의 동작 방식

1) Bubble Sort

Bubble sort 는 인접한 한 쌍의 숫자들을 비교하면서 정렬하는 방법이다. 배열의 처음부터 정렬되지 않은 곳까지 iteration 을 반복하면서, 모든 데이터가 정렬될 때까지 반복한다. n 번째의 iteration 이후, 뒤에서 n 번째까지 정렬되었음이 보장된다. 배열의 원소가 n 개일 때, 항상 비교는 총 n(n-1)/2 회 일어난다. 즉, 시간 복잡도는 $\Theta(n^2)$이다.

2) Insertion Sort

Insertion sort 는 배열을 iterate 하며 n 번째 원소를 첫 번째부터 (n-1) 번째 원소까지 중 적합한 위치에 삽입하는 방식으로 정렬한다. n 번째의 interation 이후, 앞에서 n 번째까지 정렬되었음이 보장된다. 이미 배열이 정렬된 상태에서는 비교가 총 n 번 일어나므로 최적의 경우 시간 복잡도는 $\Theta(n)$이며, 평균적인 경우와 최악의 경우 $\Theta(n^2)$이다.

3) Heap Sort

Heap sort 는 max heap 을 만들고 deleteMax 를 반복하는 방식으로 정렬이 이루어진다. 이때, heap 을 만드는 과정이 $\Theta(n)$이며, 최대값을 추출한 뒤 heap 을 수선하는 과정이 $O(nlogn)$으로, 총 시간 복잡도는 $O(nlogn)$이 된다. 이때 최선의 경우는 모든 값이 동일할 때로, 이 경우의 시간 복잡도는 $\Theta(n)$이 된다.

4) Merge Sort

Merge sort 는 배열을 작은 크기를 나누어 가며 재귀적으로 정렬하는 방법이다. 조각난 배열을 합치는 과정에서 정렬하며, 시간 복잡도는 $\Theta(nlogn)$이 된다.

5) Quick Sort

Quick sort 는 배열 안의 하나의 원소를 pivot 으로 하여, 이를 기준으로 partition 을 재귀적으로 만들며 정렬한다. 대부분의 자료에 대하여 빠른 속도를 자랑하지만, 중복된 값이 많거나 이미 정렬, 혹은 반대로 정렬된 경우 $\Theta(n^2)$의 시간 복잡도를 가지게 된다. 그 외의 평균적인 경우나 최선의 경우, $\Theta(nlogn)$의 시간 복잡도를 갖는다.

6) Radix Sort

Radix sort 는 주어진 수의 자릿수가 충분히 작을 때 사용하기에 적합하며, 자릿수 k 이하의 자료에 대하여 $O(kn)$의 시간 복잡도를 가진다. 자릿수가 낮은 수를 기준으로 stable sort 를 진행함으로써, 정렬이 정상적으로 일어날 수 있도록 한다. 즉, $O(n)$의 시간 복잡도를 가지기 위해서는 이 stable sort 를 $O(n)$안에 해결할 수 있는 counting sort 등을 활용해야 한다.

2. 동작 시간 분석

Bubble sort 의 경우, 정렬 속도가 다른 정렬에 비해 매우 느리므로, 분석에 포함하지 않았다.

1) 데이터 수 분석

데이터 수가 많은 경우, 평균적으로 Quick sort 가 가장 빠르다는 것이 알려져 있으므로, 데이터 수가 적은 경우를 중심적으로 분석하였다. 데이터 수를 10 개부터 5 개씩 늘려가며 테스트 하였으며, 100 개의 자료에 대해 평균을 계산하였다. 데이터의 수가 적을 때에는 Insertion sort 가 충분히 빠르다가, 50 개를 넘어가면서부터 Quick sort 보다 성능이 안 좋아지는 것을 확인할 수 있었다.

	10 개	15 개	20 개	25 개		45 개	50 개	55 개	60 개
I	0.00	0.01	0.00	0.00		0.01	0.03	0.01	0.04
H	0.00	0.01	0.02	0.01		0.02	0.04	0.06	0.03
M	0.01	0.00	0.00	0.02	…	0.08	0.02	0.06	0.05
Q	0.02	0.05	0.03	0.01		0.01	0.01	0.02	0.01
R	0.11	0.02	0.04	0.05		0.10	0.14	0.11	0.09

2) 자릿수 분석

자릿수 분석을 위하여 해당 자릿수 이하의 수를 랜덤으로 50000 개씩 추출하여 100 개의 자료에 대해 테스트 하였다. 자릿수가 작은 경우, 중복 데이터가 많아져 Quick sort 의 성능이 매우 떨어지는 것을 확인 할 수 있었으며, 7 자리를 넘어서면서 Quick sort 의 성능이 Radix sort 를 앞지르는 것을 확인할 수 있었다.

	1 자리	2 자리	3 자리	4 자리	5 자리	6 자리	7 자리	8 자리	9 자리
I	217.0	22.56	229.8	226.7	227.1	229.4	227.3	231.6	229.9
H	8.04	9.08	9.60	9.64	9.62	9.95	9.52	9.70	9.65
M	7.75	8.52	9.17	9.61	9.65	9.97	9.56	9.63	9.58
Q	63.15	27.06	8.71	7.88	8.14	8.67	8.05	8.03	8.04
R	5.18	5.86	6.14	6.63	7.06	7.54	7.63	8.15	8.55

3) 중복 데이터 분석

중복된 데이터가 5%씩 늘어나도록 50000 개의 데이터를 랜덤으로 생성한 100 가지의 데이터에 대해 분석을 진행하였다. 이때 중복 데이터를 만들기 위해 데이터의 범위를 제한하였으므로, Radix sort 의 경우는 분석에서 제외하였다. 5%를 넘어가는 순간, Merge sort 의 속도가 Quick sort 를 앞지르며 가장 빨라졌다. 세부 분석한 결과로는 2%에서 이미 Merge sort 이 Quick sort 를 앞지른 것을 확인 할 수 있었다. 반면 45%를 넘게 되면 Heap sort 가 Merge sort 보다 효율적인 것으로 드러났다. 100%에서는 Insertion sort 가 가장 효율적이었다. Quick sort 는 중복율이 올라감에 따라 느려지더니 55%에 이르러서는 stack overflow 가 일어났다.

	1%	2%	3%		40%	45%	50%		100%
I	144.1	144.0	143.6		128.2	127.2	120.2		1.12
H	6.17	6.14	6.09		5.68	5.57	5.51		2.05
M	6.06	6.07	6.05	…	5.63	5.74	5.65	…	3.47
Q	5.33	6.23	7.45		76.76	92.38	111.5		OF
R	6.28	6.15	6.23		6.65	6.84	6.95		5.62

4) 정렬 데이터 분석

($a_i > a_{i+1}$ 인 i 의 수)/(데이터 수 - 1)을 정렬 비율로 하여 정렬 비율을 5%씩 늘려가며 테스트를 진행하였다. 각 경우에 대하여 50000 개의 수를 포함한 100 가지의 자료를 만들어 테스트 하였으며, 완전히 정렬된 경우 Insertion sort 가, 정렬 비율이 20% 이하 혹은 75% 이상인 경우 Merge sort 가 제일 빠른 것을 확인할 수 있었다.

	10%	15%	20%	25%		70%	75%	80%		100%
I	356.7	329.3	305.7	286.1		183.0	168.9	153.0		1.91
H	9.14	9.43	9.78	9.72		9.67	9.43	9.68		7.71
M	7.49	8.22	8.65	8.96	…	8.86	8.60	8.69	…	5.41
Q	9.37	9.01	8.15	8.13		8.21	8.47	8.92		OF
R	9.62	9.62	9.73	9.62		9.59	9.49	9.58		9.41

3. Search 동작 방식

위에서 분석한 바를 토대로, Search 는 자릿수, 데이터 수, 정렬 비율, 중복 데이터 순으로 최적의 정렬 방법을 제시한다. 이때 자릿수는 상용로그 값을 이용하였고, 중복 데이터의 경우 데이터의 최대 개수 50000 이상의 최소 소수, 50021 개의 entry 를 가진 hash table 의 충돌 횟수 중 가장 충돌이 많은 경우를 이용하였다. 정렬 비율은 인접한 두 원소가 정렬된 횟수로 근사하여 이용하였다. 결론적으로 자릿수가 8 미만인 경우, 즉 상용로그의 값이 7 미만인 경우 'R'을, 데이터 수가 50 개 이하인 경우 'I'를 출력하며, 완전히 정렬된 경우에는 'I'를, 그 외에 정렬 비율이 20%이하 75% 이상인 경우 'M'을 출력하도록 했다. 또, 모든 원소가 같을 때에는 'I'를, 충돌 비율이 0.45 이상인 경우 'H'를, 그 외에 충돌 비율이 0.02 이상일 때에는 'M'을 출력하도록 했다. 나머지 경우에는 'Q'를 출력하도록 했다.

4. Search 동작 시간 분석

Search 가 정상적으로 작동하는지 확인하기 위해서는 특정 성질을 가진 데이터가 필요하므로, 위에서 진행했던 테스트 중 하나를 골라 분석을 진행하였다. 그 결과는 다음과 같다.

테스트 종류	Search 결과	All sorts	All sort w/o B	Search + Result
5 자릿수	R	3158 ms	156 ms	3 ms
정수 33 개	I	1 ms	1 ms	0 ms

15% 정렬	M	2673 ms	259 ms	5 ms
85% 정렬	M	3041 ms	136 ms	7 ms
100% 정렬	I	637 ms	29 ms	3 ms
5% 중복	M	4856 ms	264 ms	12 ms
50% 중복	H	5825 ms	378 ms	10 ms

이후, 무작위 데이터에 대해 수행한 결과 평균적으로 가장 빠른 Quick sort 를 search 결과를 출력했으며, 실제로도 모든 sort 중에 Quick sort 가 가장 빨랐다. 즉, 모든 sort 를 이용하여 최적의 sort 를 찾는 것보다 빠른 O(n)의 시간으로 최적의 sort 를 효율적으로 확인할 수 있었다.