2018-15579

조성민

1. 알고리즘 구현 방식

먼저, 버블 정렬의 경우 매번 루프를 돌 때마다 남은 배열에서 가장 큰 수가 맨 뒤로 가도록 구현하였다. 구체적으로는 어떤 원소가 그 다음에 오는 원소보다 클 때, 다음 원소와 그 원소의 위치를 교환하는 방식을 택하였다. 이와 같은 방법으로 가장 큰 수를 맨 뒤로 보낸 후 루프를 돌 때 살펴볼 마지막 인덱스 last을 하나 줄여 다시 루프를 돈다. 이런 식으로 last가 1이 될 때, 즉 정렬되지 않은 원소가 2개 남을 때까지 이와 같은 작업을 반복한다. 이때 루프를 돌면서 한 번도 교환이 일어나지 않았다면 이미 그 배열은 정렬된 것이므로 last가 1이 될 때까지 루프를 순환하지 않고 동작을 멈춘다. 즉, 교환이 일어났는지 여부를 확인하는 flag을 사용하여 버블 정렬의 시간을 단축하였다.

다음으로 삽입 정렬은 버블 정렬과 반대로 배열의 앞에서부터 정렬된 부분을 늘려 나간다. 앞에 i-1개의 원소가 정렬되어 있다고 할 때, 앞에 i-1개의 원소를 훑어가며 i번째 원소가 들어갈 위치를 찾고 그 위치에 있는 숫자와 i번째 원소의 위치를 교환한다. 이런 식으로 마지막 index까지 작업을 수행하면 배열이 정렬된다.

힙 정렬은 먼저 주어진 배열을 힙 구조로 변환한 후 가장 상단에 위치한 최대값을 가장 밑에 있는 값과 교환하는 방식으로 구현할 수 있다. 임의의 배열을 max heap으로 변환하기 위해서는 밑에 있는 잎이 아닌 노드부터 차례로 위로 올라가며 percolate down 작업을 수행하면 된다. Percolate down을 통해 그 노드를 뿌리로 하는 subtree은 힙 구조가 된다. Max heap이 완성되면 가장 위에 있는 노드를 맨 아래에 있는 노드와 교환하고 max heap의 크기가 하나 줄어들었다고 보고 최상단 노드를 percolate down하여 다시 max heap 구조를 복원한다. 노드가 1개 남을 때까지 이 작업을 반복하면 정렬이 완료된다.

병합 정렬은 주 배열과 보조 배열을 번갈아 사용하는 방식으로 시간을 단축하였다. 맨 처음에 보조 배열이 주 배열과 동일한 값을 갖도록 초기화한다. 그 다음에 보조 배열을 반으로 자른 후 각각을 정렬한 것을 병합할 때 주 배열에 값을 넣는다. 마찬가지로 반으로 자른 보조 배열을 정렬할 때는 이전에 주 배열이었던 것이 보조 배열이 되고, 보조 배열은 이전에 주 배열이 했던 역할을 수행하게 된다.

퀵 정렬의 경우 배열의 마지막 원소를 기준으로 하여 그보다 작거나 같은 원소와 큰 원소로 배열을 나누어 재귀적으로 각각의 배열에 대해 다시 퀵 정렬을 수행하는 방식으로 구현하였다. 이렇게 두 배열을 나눌 때, 기준 원소보다 큰 원소들 중 가장 앞에 있는 것의 index을 i라고 할 때, 기준 원소보다 작은 원소를 만나면 i에 있는 원소와 그 원소의 자리를 교환하고 i를 1 증가하는 식으로 partition을 구현하였다.

기수 정렬은 음수로 구성된 배열과 양수로 구성된 배열을 따로 기수 정렬한 후 이 둘을 합치는 방식으로 구현하였다. 구체적으로는 각각의 배열을 절댓값을 기준으로 기수 정렬한 후 음수 배열은 뒤에서부터 거꾸로, 양수 배열은 앞에서부터 결과 배열에 값을 옮겨 배열을 정렬하였다. 이때 기수 정렬은 각각의 자리에 대해 가장 밑에 있는 자리부터 계수 정렬을 하는 방식으로 구현하였다.

2. 정렬 시간 분석

모든 정렬에 대하여 실행횟수는 5회.

2. 1 최대 자릿수에 따른 시간 분석

Test size가 각각 1000, 10000, 50000일 때 최대 자릿수가 1~9인 경우를 모두 검증해 보았으며, test size가 1000일 때를 비롯하여 모든 경우에서 merge sort 또는 quick sort가 가장 빠른 것으로 나타났다.

Test size: 1000, digit: 1

maxNumDigits(자릿수 최대값): 2, duplicateRatio(중복 비율): 0.977, sortedRatio(정렬된 비율): 0.54 (5회에 걸쳐 거의 유사하므로 첫 번째 실행에 대한 값만 표기)

평균 표준편차 (ms)

bubble: 8.6 / 2.72 insertion: 2.0 / 2.10 heap: 0.4 / 0.49 merge: 0.2 0.40
quick: 1.0 0.6324555320336759 radix: 1.2 0.40

Test size: 1000, digit: 5

maxNumDigits: 5, duplicateRatio: 0.004, sortedRatio: 0.505

bubble: 3.2 0.39 insertion: 0.2 0.40 heap: 0.2 0.40 merge: 0.2 0.40

quick: 0.4 0.48 radix: 1.4 0.48

Test size: 1000, digit: 9

maxNumDigits: 9, duplicateRatio: 0.0, sortedRatio: 0.501

bubble: 2.6 0.48 Insertion: 0.4 0.48 heap: 0.0 0.0

merge: 0.2 0.40 quick: 0.0 0.0 radix: 3.2 0.97

2.2 중복에 따른 시간 분석

Test size가 각각 1000, 10000, 50000일 때 중복도가 0.1, 0.2, ..., 0,9, 1인 경우를 살펴보았다. 중복도가 0.9보다 작거나 같은 경우 대체로 quick sort가 가장 빠른 것으로 나타났다. 추가로 중복도가 0.95,96, .., 0.99인 경우를 살펴본 결과 0.98을 기준으로 quick sort보다

merge sort가 빨라지는 경향이 나타났다. 또한, 모든 수가 동일한 경우에는 insertion sort와 bubble sort가 가장 빠른 수행시간을 보였다.

Test size: 1000, ratio: 0.98

maxNumDigits: 10, duplicateRatio: 0.98, sortedRatio: 0.499

bubble: 4.6 4.58 insertion: 2.2 2.03 heap: 0.6 0.48

merge: 0.6 0.48 quick: 0.4 0.80 radix: 3.6 1.35

Test size: 10000, ratio: 0.98

maxNumDigits: 10, duplicateRatio: 0.98, sortedRatio: 0.4999

bubble: 75.0 21.00 insertion: 11.6 0.48 heap: 1.6 0.48

merge: 1.0 0.0 quick: 1.0 0.0 radix: 12.0 0.63

Test size: 50000, ratio: 0.98

maxNumDigits: 10, duplicateRatio: 0.98, sortedRatio: 0.5

bubble: 2459.8 54.67 insertion: 237.6 30.88 heap: 4.6 0.79

merge: 3.6 0.48 quick: 4.0 0.0 radix: 48.6 4.92

2.3 정렬 비율에 따른 시간 분석

모든 (i, i+1) 쌍에 대하여 arr[i] <= arr[i+1]인 비율을 정렬 비율로 정의하였으며, 정렬 비율이 0.5에서 멀어질수록 quick sort의 효율이 떨어지는 경향이 나타났다. 이러한 경향은 배열의 크기가 클수록 더 강하게 나타났다. 한편, 단조 증가하는 수열에 대해서는 bubble sort와 insertion sort가 가장 효과적이었다.

Test size: 1000, ratio: 0.4

maxNumDigits: 7, duplicateRatio: 0.0, sortedRatio: 0.382

bubble: 2.2 0.39 insertion: 0.4 0.48 heap: 0.4 0.48

merge: 0.2 0.4 quick: 0.4 0.48 radix: 1.8 0.4

Test size: 1000, ratio: 0.5

maxNumDigits: 7, duplicateRatio: 0.0, sortedRatio: 0.485

bubble: 1.0 0.0 insertion: 0.0 0.0 heap: 0.2 0.40

merge: 0.2 0.40 quick: 0.2 0.4 radix: 1.0 0.63

Test size: 1000, ratio: 0.6

maxNumDigits: 8, duplicateRatio: 0.0, sortedRatio: 0.575

bubble: 0.0 0.0 insertion: 0.0 0.0 heap: 0.4 0.48

merge: 0.0 0.0 quick: 0.2 0.40 radix: 1.0 0.0

3. Search 동작 방식

일단 모든 수가 동일하거나(중복도가 1인 경우) 배열 전체가 단조 증가할 때 (정렬 비율이 1인 경우) 버블 정렬을 출력하도록 하였다. 또한, 퀵 소트의 경우 정렬 비율에 민감하게 반응하는 경향을 보여 0.5에서 0.01만큼 떨어지면 병합 정렬을 사용하도록 설정하였다. 그리고 중복도가 0.98보다 큰 경우에도 퀵 소트의 효율이 병합 정렬보다 떨어져 병합 정렬을 택하였다. 이외의 경우에는 퀵 소트를 택한다.

4. Search와 모든 정렬을 다 실행하는 방식 비교

Test size: 1000

others: 9.0 3.63

search: 0.8 0.40

Test size: 10000

others: 247.2 30.05

search: 1.8 0.74

Test size: 50000

others: 5674.6 18.47

search: 9.0 2.0