알고리즘설계와분석 HW2 보고서

20171461 이강희

*****Readme*** 본 보고서에서 알고리즘의 실행시간을 비교할 때 clock()의 end – start 를 CLOCKS_PER_SEC 로 나누어 주지 않은 실행시간임을 참고 부탁드립니다. 코드는 질의응답을 보고 초 단위로 수정해두었습니다. 감사합니다.**

Experiment environment:

*****본 보고서 작성에 사용된 hardware spec 은 아래와 같습니다.*****
CPU: 2.2 GHz Quad-Core Intel Core i7
Memory: 16 GB 1600 MHz DDR3
OS type: mac OS
OS version: macOS Monterey(ver. 12.6.9)

Experiment setup:

1. Random array

각 알고리즘을 비교하기 위한 첫번째 metric 으로는 우선랜덤한 배열을 정렬하는 것을 이용했다. Input 을 크기 별로 20 부터 200,000 크기까지의 범위로 testcasegen 에서 각 input 에 대해서 다른 seed 값으로 각각 3 번씩 평균을 내어 비교했다. 이때 정렬 속도는 clock()함수를 이용했다. 그리고 average case 로 O(nlogn)을 갖는 quicksort 와 worst case 로 O(nlogn)을 갖는 merge sort 와 최적화를 시킨 4 번 알고리즘 정렬에 해당하는 introsort 세 개의 경우 input size 1,000,000 까지 배열의 크기가 커질 수록 어떤 알고리즘이 더 잘 동작하는지를 그래프를 통해 비교했다.

2. Non-increasing ordered array

두번째 metric 으로는 내림차순으로 정렬된 배열을 이용했다. 내림차순에서는 우리가 원하는 정렬방향이 오름차순이기 때문에 반대로 정렬되어 있다. 따라서 insertion sort 의 worst case 에 해당하여 O(n^2)의 time complexity 를 갖는다. 그리고 Quicksort 가 일반적으로 가장 빠른 정렬 알고리즘으로 사용되지만, Non-increasing order 나 non-decreasing order 처럼 정렬된 배열의 경우에는 partition 이 worst case 로 quadratic 의 time complexity 를 갖는 것을 예상할 수 있다.

3. **Non-decreasing ordered array**

위의 non-increasing order 는 내림차순 정렬이므로 우리가 구현한 정렬 알고리즘과는 반대로 정렬을 한 경우이다. 반면, 오름차순 정렬에서는 insertion sort 가 best case 인 O(n)을 갖는다. Quicksort 는 두 정렬된 경우 모두 quadratic time complexity 를 나타내지만 insertion sort 는 내림차순 정렬에서는 time complexity 가 O(n^2) 것과 비교할 수 있으므로 내림차순 뿐만 아니라 오름차순으로 정렬된 두 경우 모두 비교하였다.

Algorithm 4 Design:

위에서 언급한 것처럼 Quicksort 가 일반적으로 average case 에서는 대체로 가장 빠른 정렬 알고리즘이기는 하다. 하지만 배열의 끝 부분을 pivot 으로 고르는 알고리즘의 경우, 정렬된 배열에서 input size 를 n 이라고 한다면 O(n^2)의 시간복잡도를 갖는다. 정렬된 배열에서 배열의 끝부분을 pivot 으로 고르면 partition 이 비효율적으로 일어나기 때문이다. 따라서 quicksort 의 장점을 살리면서 worst case 경우를 방지하고, 그럼에도 worst case 에 해당하는 배열일 경우에는 이를 판별하여 다른 sorting 방식으로 전환하여 개선시켰다. 자세한 설계내용은 아래와 같다.

먼저 partition 이 효율적으로 일어나려면 pivot selection 이 중요한데, 이때 pivot 을 median of three 함수를 이용하여 pivot 을 골라 worst case 의 partition 인 n -1 번이 일어나지 않도록 보장하여 quicksort 에서 O(nlogn)의 time complexity 를 보장했다. 또한 이론과 더불어 실제 실험 결과 크기가 작은 배열의 경우 insertion sort 가 빠르고 효과적으로 정렬했으므로 partition 으로 인해 배열이 일정 크기 이하로 작아지면 insertion sort 로 전환하도록 cutoff 를 이용했다. 이때 cutoff 는 insertion sort 가 다른 정렬보다 성능이 더 좋은 범위로 알려진 16 으로 설정했다. 실제로 insertion sort 는 상수계수가 작기 때문에 작은 배열에서 유리하다는 점을 이용했다. 뿐만 아니라 recursion depth 가 일정 범위 이상 증가하면 quicksort 가 아닌 heap sort 로 전환하여 worst case time complexity 인 O(n^2)를 방지하고 O(nlogn)의 time complexity 를 유지할 수 있도록 했다. 이때 depth limit 은 2*logn 으로 정했는데, 이는 introsort 에서 가장 효율적으로 heap sort 로 전환하는 깊이로 알려져 있다. 위 알고리즘 설계방식을 간단히 말하자면, introsort 가 실행되며 partition 이 일어나는데, 이때 recursion

depth 가 depth limit 인 2*logn 을 넘으면 heap sort 로 전환하여 정렬이 일어나고, 배열 내 원소의 개수가 cutoff 인 16 이하일 경우 insertion sort 로 정렬하게 되는 것이다.

즉, introsort 는 각 알고리즘의 장점을 혼합하여 구현한 hybrid 알고리즘이다. Quicksort 의 빠른 정렬속도, heap sort 의 안정성, 작은 배열에서 효과적인 insertion sort 를 이용하여 quicksort 의 worst case 에 해당하는 배열에서도 O(nlogn)의 시간복잡도를 유지하도록 디자인된 알고리즘이다.

Discussion:

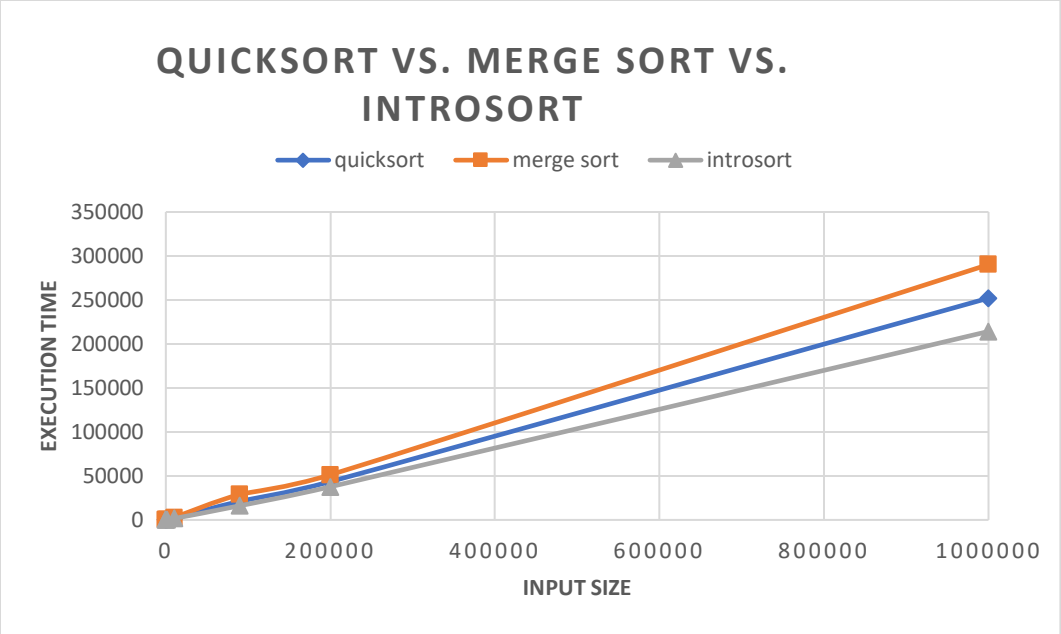
각 experiment setup 에 대한 실험 결과는 아래와 같다.

1. Random array

input size	20	100	1000	10000	90000	200000	1000000
insertion sort	3	14	1205	84553	7828396	38402648	
quicksort	3	19	140	1972	21193	43508	252072
merge sort	4	20	211	2774	28998	51281	290253
introsort	13	43	144	1603	16147	37666	214232

<Table 1. testcasegen 으로 각 배열의 input size 에 따른 seed 를 다르게 하여 평균을 낸 정렬속도>

100 정도까지의 input size 에서는 네 알고리즘 모두 정렬 속도가 좋았으나, 1000 부터 insertion sort 는 다른 알고리즘들에 비해서 성능이 급격하게 나빠졌다. Insertion sort 는 작은 크기의 배열에서는 효과적인 정렬 방식이나 크기가 커질 수록 성능이 악화되는 것을 관찰할 수 있었다. 그리고 배열의 크기가 커지면 커질수록 introsort > quicksort > merge sort 순으로 빠른 정렬 속도를 나타내었다. Quicksort 는 worst case 가 O(n^2)임에도 극단적인 partition 의 경우가 아닐 때에는 merge sort 와 introsort 처럼 average case 에서는 O(nlogn)의 time complexity 를 갖는 것을 확인할 수 있었다. 랜덤한 배열의 경우 Insertion sort 를 제외한 세 정렬알고리즘의 asymptotic time complexity 가 O(nlogn)임으로, 이를 그래프로 표현하여 비교하였다.



<Figure 1. Quicksort 와 merge sort, introsort 를 비교하여 나타낸 그래프>

그래프로 비교한 결과 input size 에 따른 실행속도는 introsort, quicksort, merge sort 순으로 빨랐다.

2. Non-increasing ordered array

input size	1000	10000	100000
insertion sort	2256	164994	18702226
quicksort	3178	297964	26071413
merge sort	152	1868	13768
introsort	309	4029	7078

<Table 2. Input size 에 따른 내림차순 정렬된 배열의 정렬속도>

정렬된 배열은 quicksort 의 worst case 에 해당하기 때문에 O(n^2)의 시간복잡도를 예상할 수 있고, 실험결과 실제로 가장 느린 정렬 속도를 보였다. 이때 insertion sort 도 내림차순으로 정렬된 경우 worst case 인 O(n^2)에 해당하는데 quicksort 가 이것에 비해도 느린 성능을 나타내는 것을 통해 quicksort 의 평균 정렬 속도가 가장 빠르더라도 worst case 에서는 매우 느려지기 때문에 이를 해결하는 것이 중요하다고 생각하여 introsort 의 필요성을 더욱 부각시켰다. Introsort 는 median of three 함수와 재귀깊이가 깊어질 때 heap sort 로 전환하는 것을 이용하여 가장 좋은 성능을 보이며

quicksort 의 worst case 에 전혀 영향받지 않는 것을 확인할 수 있었고, heap sort 는 배열의 정렬 여부와 관계없이 안정적인 퍼포먼스를 나타내었다.

3. Non-decreasing ordered array

input size	1000	10000	100000
insertion sort	8	62	449
quicksort	4312	351264	37220159
merge sort	160	2645	13575
introsort	373	1657	8934

<Table 3. Input size 에 따른 오림차순 정렬된 배열의 정렬속도>

오름차순 정렬된 배열은 정렬 알고리즘이 제대로 수행된 결과를 의미한다. Quicksort 의 경우 여전히 partition 이 비효율적으로 일어나는 worst case 에 해당하여 아주 나쁜 성능을 보이지만, 이번엔 insertion sort 의 best case 인 O(n)에 해당하므로 insertion sort 가 네 알고리즘 중 가장 좋은 성능을 나타내었다. 마찬가지로 heap sort 는 여전히 안정적인 성능을 나타내었고, introsort 는 그보다도 더 좋은 성능을 나타내어 quicksort 의 worst case 를 완전히 극복하여 insertion sort 의 best case 인 O(n)의 time complexity 다음으로 가장 좋은 성능을 나타내었다.