MP2_Report

20191048 김도솔

1. Experiment environment

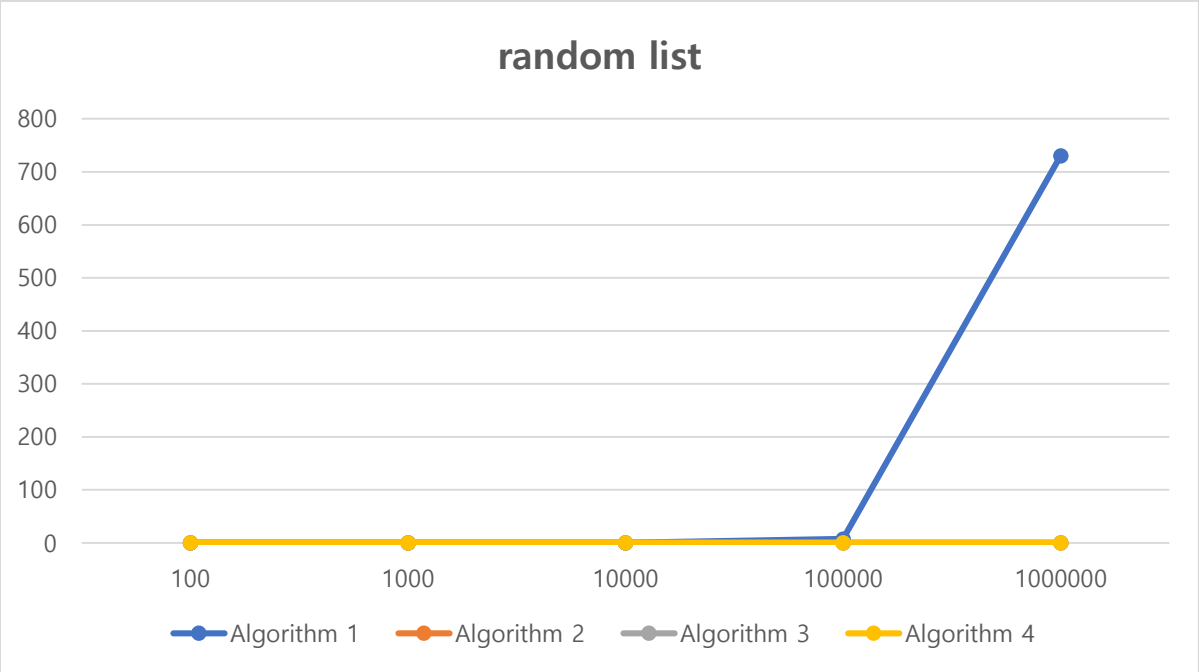
- Windows 10 Pro
- 프로세서: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
- RAM: 8.00GB(7.80GB 사용 가능)
- 시스템 종류: 64비트 운영 체제, x64 기반 프로세서

2. Experiment setup

- 측정 항목: 정렬 시간(sorting algorithm 시작부터 종료까지)
- 정수 범위: -10,000부터 9,999까지의 정수
- 입력 크기 범위: 100부터 1000000개의 정수 배열
- 반복 횟수: 각 경우마다 5개의 서로 다른 input 수행 시간 측정 후 평균을 기록

3. 알고리즘 성능 비교

1-1) random list (정수 범위: -10000 ~ 9999)



	n = 100	n = 1000	n = 10000	n = 100000	n = 1000000
Algorithm 1	0.000023	0.001916	0.107142	7.346041	730.431557
Algorithm 2	0.000019	0.000229	0.003059	0.019702	0.206176
Algorithm 3	0.000036	0.000390	0.004694	0.030357	0.244280
Algorithm 4	0.000022	0.000287	0.002435	0.014487	0.107633

● n 값이 작을 때는 알고리즘 2의 수행 속도가 가장 빠르나, 알고리즘 간의 수행 속도 차이가 미미하다. n 값을 점점 증가시키면 알고리즘 4의 수행 속도가 다른 알고리즘에 비해 확연하게 빠른 것을 관찰할 수 있다.

● 알고리즘 4는 n이 digit 수보다 큰 경우에는 radix sort를, 작은 경우에는 intro sort를 사용하도록 구현되었기에, n이 digit 수보다 큰 경우에는 O(n), 작은 경우에는 O(nlogn)의 시간복잡도를 가진다. 따라서 n이 digit 수보다 작은 경우 quick sort보다 intro sort가 좀더 복잡하게 구현되어 quick sort의 수행 속도가 더 빠르고, n이 digit 수보다 큰 경우 radix sort의 수행 속도가 quick sort보다 빨라 이러한 결과가 나왔다고 볼 수 있다.

● 알고리즘 1(insertion sort)은 다른 알고리즘들 보다 n 값의 증가에 따른 수행 시간 증가의 폭이 크다. 특히 n = 1000000인 경우 알고리즘 1의 수행 시간이 나머지 세 알고리즘에 비해 매우 오래 걸려 성능이 가장 좋지 않음을 확인할 수 있다. 이는 알고리즘 1이 O(n^2)의 시간 복잡도를 가지기 때문이다.

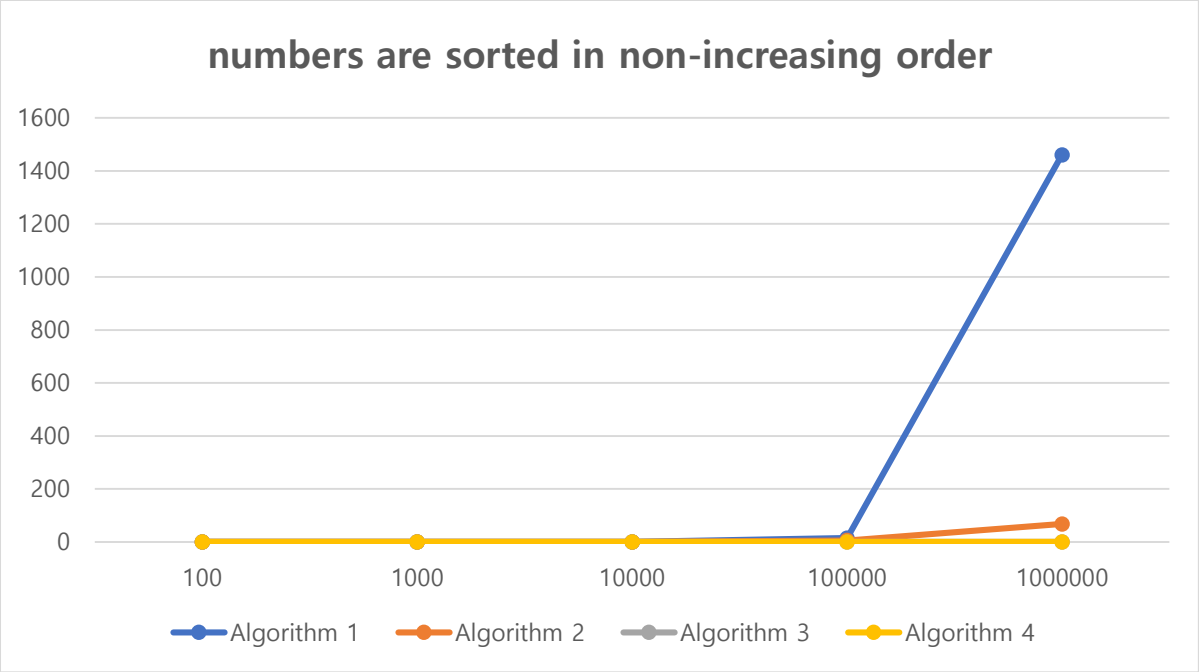
● 알고리즘 2(quick sort)는 n 값이 자릿수보다 작을 때 가장 빠른 수행 속도를 보인다. 이는 알고리즘 2가 O(nlogn)의 시간복잡도를 가지는 알고리즘 중에서도 빠른 편에 속하기 때문이다.

● 알고리즘 3(merge sort)는 알고리즘 2와 비슷한 수행 속도를 보인다. 이 또한 O(nlogn)의 시간복잡도를 가지기 때문이다.

● 알고리즘 4(radix sort + intro sort)는 input size와 digit 수의 관계에 따라 O(n) 또는 O(nlogn)의 시간복잡도를 가진다. 따라서 input size가 digit 수보다 작을 경우에는 알고리즘 2, 3과 비슷한 수행 시간을 보이고, input size가 digit 수보다 커지면 다른 알고리즘들 보다 적은 수행 시간을 보인다.

● 정리하자면 알고리즘 1(insertion sort)은 O(n^2) 알고리즘 2(quick sort), 3(merge sort)는 O(nlogn), 알고리즘 4(radix sort + intro sort)는 경우에 따라 O(n) 또는 O(nlogn)의 시간복잡도를 가진다. 위 실험을 통해 input size가 커짐에 따라 O(n^2), O(nlogn), O(n)의 차이가 점점 커지는 것을 실험적으로 관찰할 수 있었다.

1-2) numbers are sorted in non-increasing order (정수 범위: -10000 ~ 9999)



	n = 100	n = 1000	n = 10000	n = 100000	n = 1000000
Algorithm 1	0.000043	0.003795	0.190208	14.606804	1460.395621
Algorithm 2	0.000054	0.004508	0.189912	5.742058	67.554403
Algorithm 3	0.000027	0.000259	0.002926	0.019206	0.156241
Algorithm 4	0.000022	0.000251	0.002346	0.014121	0.098195

● 1-1과 달리 알고리즘 4가 모든 경우에 가장 빠른 수행 속도를 보인다. 이는 내림차순으로 정렬된 input이 알고리즘 2(quick sort)에 불리하게 작용해 알고리즘 2의 수행 시간이 늘어났기 때문으로 보인다.

● 알고리즘 1(insertion sort)은 1-1보다 수행 시간이 확연하게 증가했다. 이를 통해 알고리즘 1의 성능이 4개의 알고리즘 중 가장 안좋음을 알 수 있고, 알고리즘 성능이 input 패턴에 영향을 받음을 알 수 있다. 이는 원하는 정렬 순서와 반대로 된 input을 만나면 while문의 수행 횟수가 늘어나기 때문이다.

● 알고리즘 2(quick sort)의 경우에는 n 값이 증가할수록 1-1보다 수행 시간이 확연하게 증가하는 것을 관찰할 수 있다. 이는 quick sort가 pivot이 어떻게 골라지는지에 따라 성능의 차이가 심해 최악의 경우에 O(n^2)과 유사한 시간복잡도를 가지기 때문이다.

● 알고리즘 3(merge sort)은 input의 패턴이 어떻든 항상 반으로 나누어 정렬하기 때문에 수행 시간이 input 패턴에 영향 받지 않는다. 따라서 1-1과 비슷한 수행 시간을 보인다.

● 알고리즘 4(radix sort+intro sort)도 3과 마찬가지로 수행 시간이 input 패턴에 영향을 받지 않는다. radix sort는 정수를 digit 별로 정렬하고, intro sort는 앞서 말한 quick sort의 단점을 보완하여 일정 재귀 호출 깊이가 넘으면 heap sort를 이용하기 때문이다. 따라서 1-1과 비슷한 수행 시

간을 보이며 오히려 감소하는 경향을 보인다.

3. Algorithm 4 설명

알고리즘 4는 **radix sort와 intro sort를 결합**하여 만들었다.

우선 compatison sort보다 정수 배열을 빠르게 정렬할 수 있는 non-comparison sort를 사용해 구현하려고 했다. 그 중 정수를 각 자릿수 별로 정렬하는 radix sort를 골랐고, 각 자릿수는 마찬가지로 non-comparison sort인 counting sort를 이용해 정렬하는 방식을 사용했다. radix sort는 음수와 양수를 한번에 비교하는 데에 어려움이 있어 음수와 양수를 각각 다른 배열에 나누어 따로 정렬한 후, 하나로 합쳤다. radix sort는 digit과 n의 관계에 따라 asymptotic time complexity가 달라질 수 있다. 따라서 양수와 음수 각각에 대해 최댓값을 찾아 max에 저장하고, 이 값이 배열 크기 n보다 작으면 radix sort를 사용하여 정렬하고, 그렇지 않으면 intro sort를 사용하여 정렬하도록 구현하고자 했다.

intro sort를 사용한 이유는 다음과 같다. quick sort는 O(nlogn)의 시간 복잡도를 가지는 다른 정렬 알고리즘보다 빠르지만, worst case의 경우 수행 속도가 O(n^2)이 되는 단점을 가진다. 이러한 점을 보완하고자 quick sort의 빠른 속도를 유지하고, heap sort의 안정성을 얻을 수 있는 intro sort를 구현했다. 이는 기본적으로 quick sort를 사용하며, 재귀 호출이 특정 깊이에 도달하면 heap sort로 전환된다. 입력 부분 배열이 일정 크기 이하로 축소되면 insertion sort를 사용하여 최종적으로 정렬을 완료하도록 구현했다.

따라서 알고리즘 4는 digit이 n보다 작은 경우 O(n)의 시간복잡도를 가지고, 그렇지 않은 경우에도 안정적으로 O(nlogn)의 시간복잡도를 가져 알고리즘 1, 2, 3에 비해 우수한 성능을 보인다.