4 번 과제 보고서

2013-11392 김지현

A. 동작 방식

1. Bubble Sort

2 중 반복문을 사용하여, 인접한 두 숫자들을 swap 해가며 정렬하는 알고리즘이다.

- 첫번째 루프는 $i \in [0, n)$범위에서 움직인다.
- 두번째 루프는 $j \in [0, n-1-i)$범위에서 움직인다. j와 $j+1$번째 원소를 비교하며, j번째 값이 더 클경우 둘을 바꿔친다.

반복횟수는 $\sum_{i=0}^{n-1} n-1-i \in O(n^2)$이다. 불필요한 swap 횟수가 너무 많아서, 다른 정렬 알고리즘들에 비해 느리다.

2. Insertion Sort

버블소트와 마찬가지로 2 중 반복문을 사용하여 인접한 두 숫자들을 swap 해가며 정렬하는 알고리즘이나, 배열의 앞부분을 항상 정렬된 상태로 유지시키며, 새 원소들을 정렬된 앞부분의 배열에 삽입해가며 정렬한다는점이 제일 큰 차이점이다.

- 첫번째 루프는 $i \in [1, n)$범위에서 움직인다.
- 두번째 루프는 $j \in (0, i]$범위에서 움직인다. j와 $j-1$번째 원소를 비교하며, $j-1$번째 값이 더 클경우 둘을 바꿔친다.

반복횟수는 $\sum_{i=1}^{n-1} i \in O(n^2)$이다. 일반적인 경우 다른 $O(n^2)$ 정렬 알고리즘들에 비해 빠르다.

3. Heap Sort

Priority Queue 인 Heap 자료구조를 사용하여 정렬하는 알고리즘이다. 본 과제에선 Binary MaxHeap 을 사용하였다. 힙소트는 아래와 같은 방식으로 작

동한다.

A. 정렬되지 않은 배열이 주어지면, 빈 Heap 에 배열의 원소들을 하나하나 추가한다. 이때 원소 하나 추가에 $O(\log n)$의 시간이 필요하다.
B. 배열의 원소가 남지 않을때까지 반복하여, 완전한 Heap 을 만든다. 배열의 원소가 총 n개이므로, 힙의 완성에 $O(n \log n)$ 이 소요된다.
C. Heap 에서 최댓값을 하나씩 빼낸다. Heap 자료구조는 $O(1)$ 시간만에 최댓값을 찾을 수 있게 해주므로, 이 과정은 $O(n)$ 만에 끝난다.

결과적으로 시간복잡도는 $O(n \log n)$ 이다. 힙소트는 Inplace 소트여서 필요한 추가메모리가 아주 적다는 장점은 있으나, 불필요한 swap 횟수가 너무 많아서 같은 시간복잡도의 다른 알고리즘들에 비해 비효율적이다.

4. Merge sort

재귀 알고리즘이다. 정렬을 요청받은 배열을 반으로 나눠, 왼쪽 반과 오른쪽 반을 재귀적으로 merge sort 한 후, 정렬된 양쪽 두 배열을 하나로 합친다. 양쪽 반이 정렬되어있음이 보장되어서 합치는 과정은 효율적으로 $O(n)$만에 이뤄진다.

이때 이진 분할정복 알고리즘의 특성상 Recursion 의 depth 는 항상 $\log n$ 이므로, 정렬 전체의 시간복잡도는 $O(n \log n)$ 이다.

5. Quick sort

재귀 알고리즘이다. 정렬을 요청받은 배열 안에서 임의로 원소(=pivot)를 하나 고른 뒤, 요청받은 배열을 pivot 보다 큰 원소들과 pivot 보다 작은 원소들로 나눈다. Pivot 보다 작은 원소들은 모두 pivot 의 왼쪽으로 몰고, pivot 보다 큰 원소들은 모두 pivot 의 오른쪽으로 몬 뒤, pivot 의 왼쪽과 오른쪽을 각각 재귀적으로 quick sort 하는것이다.

원소들을 pivot 보다 큰 원소와 작은 원소로 나누는 과정은 $O(n)$만에 이뤄진다. 하지만 머지소트와는 달리 Recursion 의 depth 가 $\log n$만에 끝난다는 보장이 없어, 시간복잡도는 $O(n^2)$ 이다. Pivot 을 고를때 우연히 그 배열의 최솟값이나 최댓값을 골랐다면, 다음 재귀가 둘로 갈라져서 실행되지 못하기

때문이다.

저런 특수한 경우를 제외하고, 평균적인 경우를 가정하였을경우 매 Recursion 마다 배열이 둘로 갈라져서 실행된다면 시간복잡도는 $O(n \log n)$ 이다.

이 과제에선 partitioning algorithm 으로 포인터를 양쪽 끝에서 이동시키는 방식인 LR pointer 방식을 사용하였다. Pivot 은 배열의 정중앙에 있는 값으로 골랐다.

개선점 제안

퀵소트는 알고리즘 특성상 개선시킬부분이 많다.

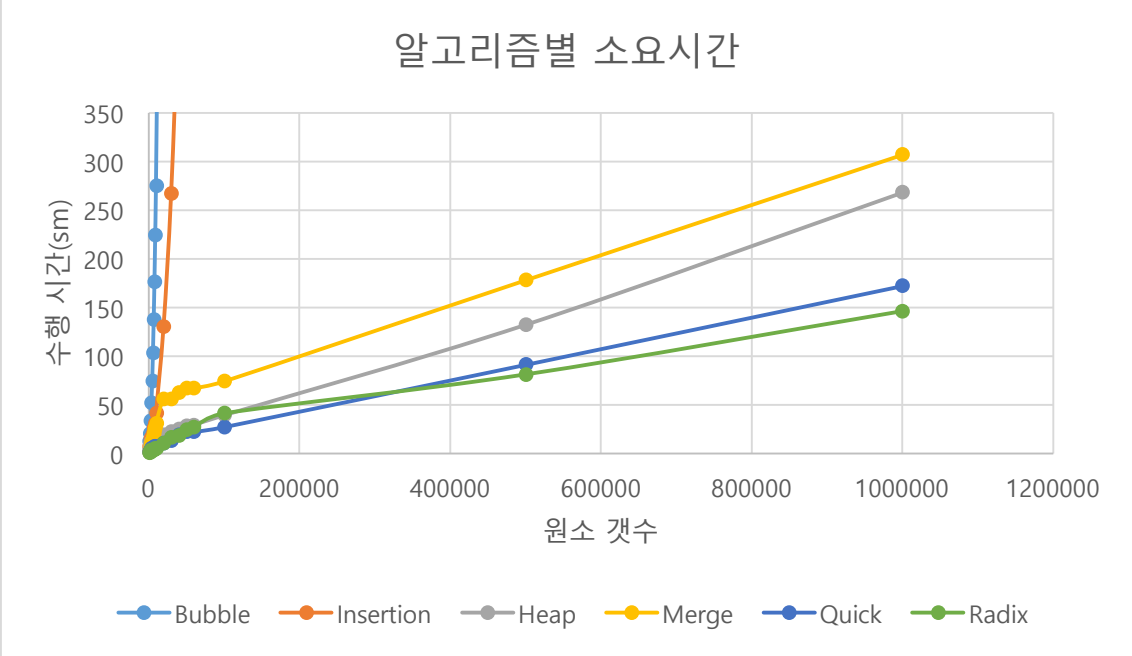
- 배열 길이가 아주 짧을경우 퀵소트 대신 Insertion sort 와 같은 정렬 알고리즘을 대신 사용하면 성능을 올릴 수 있다.
- Pivot 을 고르고 배열을 pivot 의 위 아래로 나누고 나면 pivot 의 위치는 고정되어 움직이지 않는다. 배열이 둘로 갈라졌을경우, 양쪽 배열은 서로에게 영향을 주지 않기때문에, 병렬프로그래밍을 할 수 있다.
- 본 과제에선 그렇게 하지 않았으나, pivot 선정을 첫번째값, 맨 끝값, 배열의 중앙에 위치한 값 중 중앙값으로 고르는 median-of-3 알고리즘을 쓸경우 성능이 개선된다.

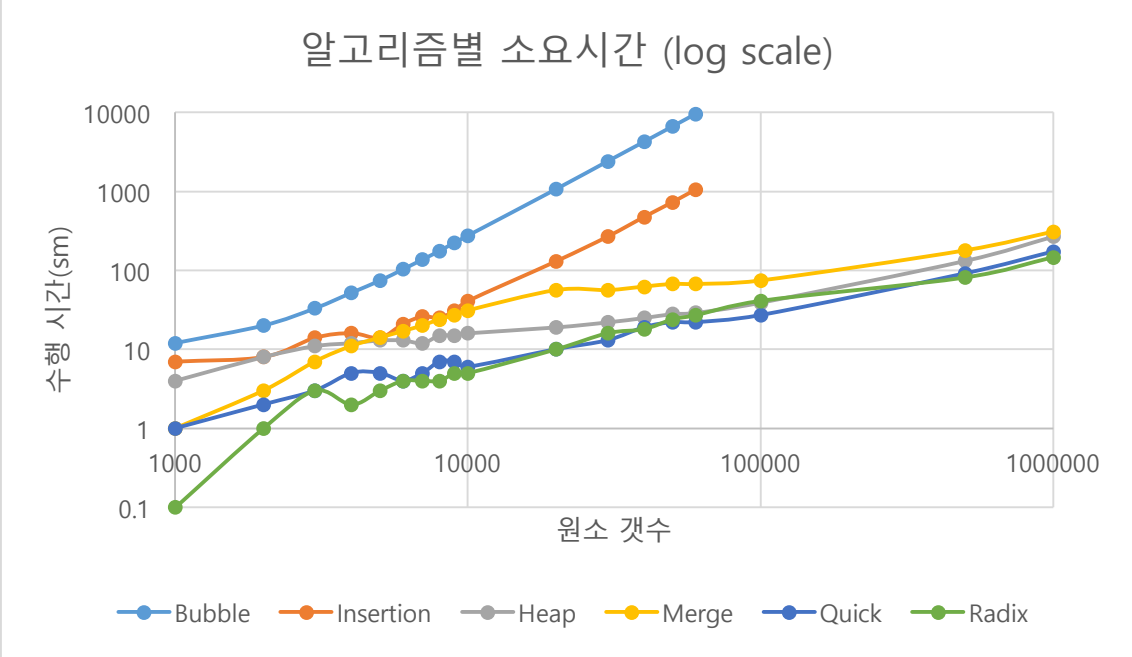
6. Radix sort

정수를 자리수별로 나눠, 자리수별로 독립적으로 stable sort 를 수행하는 알고리즘이다. 본 알고리즘에선 자리수별로 실행할 stable sort 알고리즘으로 counting sort 알고리즘을 사용하였고, 정수들은 256 진법으로 나누었다.

B. 비교분석

1. 정렬 방법에 따른 수행시간 변화





위의 두 그래프는 6 가지 정렬방법의 성능을 비교한것이다. 입력으로 주어지는 숫자들은 [-1000000, 1000000]의 범위로 중복이 생기지 않도록 하였다.

실험결과는 아래와 같이 해석할 수 있었다.

- 대부분의 경우 속도 순위
  Radix > Quick > Heap > Merge > Insertion > Bubble
  그리고 10 만개 언저리의 좁은 숫자범위 내에선 Radix sort 와 Quick sort 의 순위가 역전함을 알 수 있다.

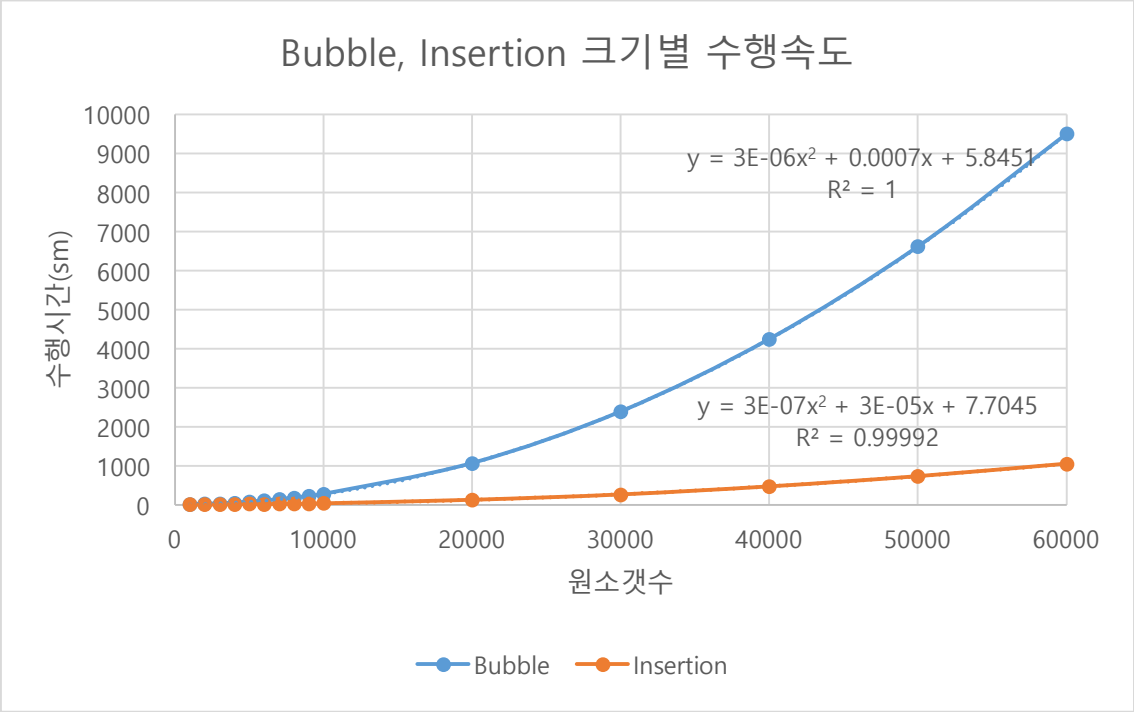
일반적인 경우와는 다르게 래딕스소트의 속도가 퀵소트보다 느린 구간이 존재하는데, 이는 256 진법 래딕스소트를 사용하였기 때문이다. 메모리를 더 사용하여 좀더 큰 밑을 사용했으면 저 범위에서도 Quick sort 보다 빠를 수 있었다.

위 실험결과를 통해 소팅 알고리즘들을 아래와 같이 비교할 수 있었다.

1. Bubble 소트는 모든경우에 항상 느린 제일 안좋은 알고리즘임을 알 수 있었다.
2. Insertion 소트는 O(n^2) 알고리즘 사이에선 성능이 우수하나, 원소 수가 많아지면 O(nlogn) 알고리즘에 비해 너무 성능이 나쁨을 알 수 있다. 그러나 수백개 단위의 매우 작은 배열의 정렬에선 오히려 다른 O(nlogn) 정렬보다 빠른 경우도 있었다.
3. O(nlogn) 알고리즘들 사이에선 항상 Quick 소트가 제일 빠르고, 그 다음은 Heap 소트, 그 다음은 Merge 소트의 순위를 갖는다.
4. Radix 소트는 대부분의 경우 제일 우수한 성능을 가졌다.

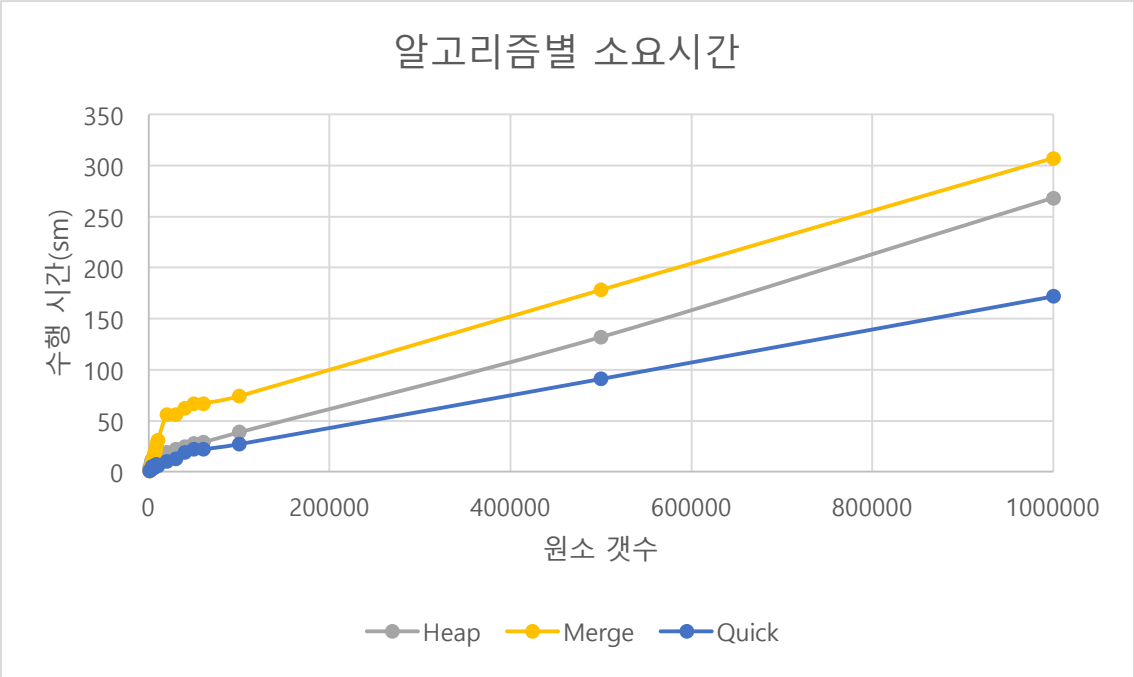
2. 데이터 갯수에 따른 수행시간 변화

a) O(n^2) : Bubble sort, Insertion sort



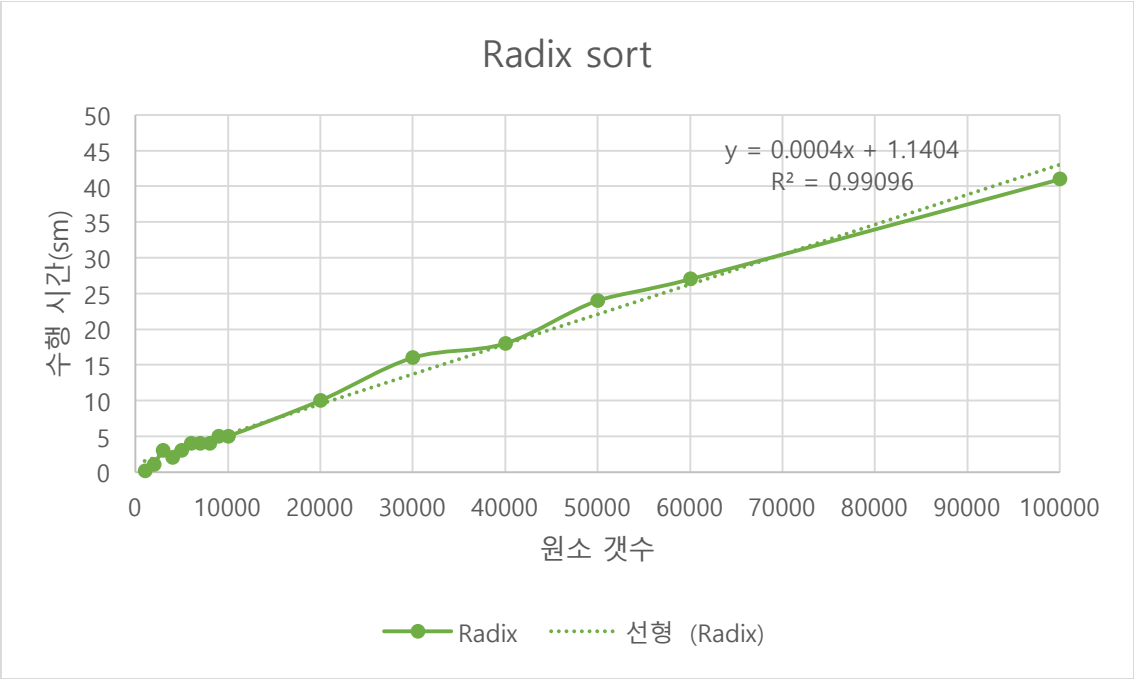
Bubble sort 와 Insertion sort 의 경우, 수행시간이 거의 완벽하게 $O(n^2)$꼴을 따름을 알 수 있다.

b) O(nlogn) : Heap sort, Merge sort, Quick sort

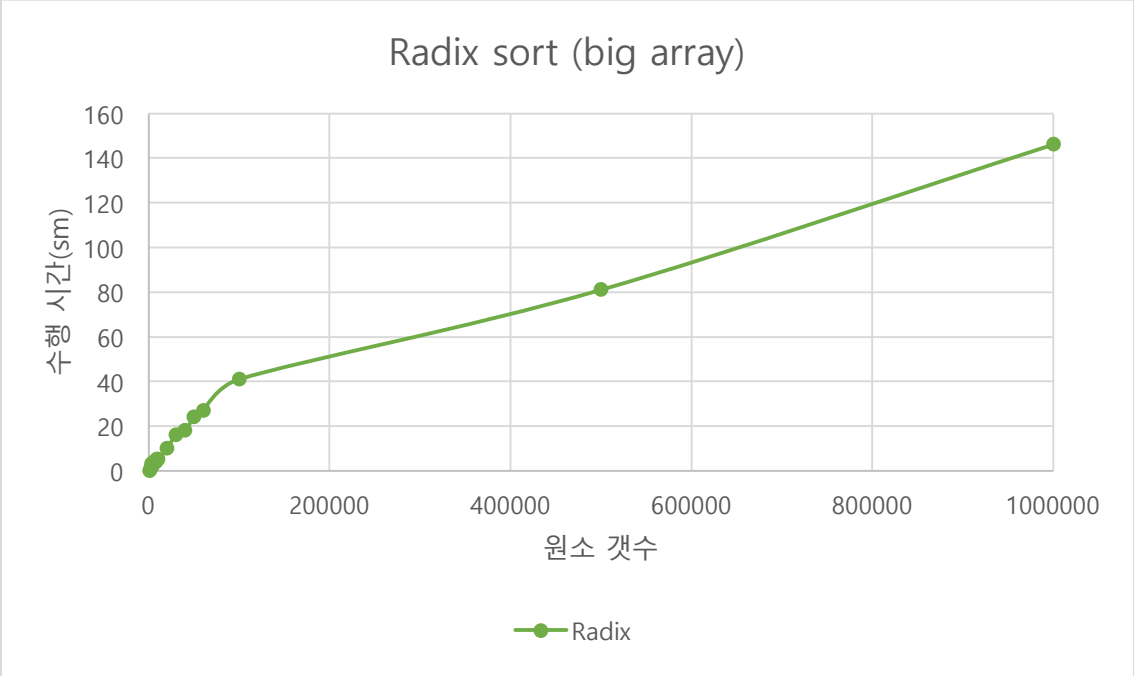


힙소트, 머지소트, 퀵소트의 경우 작은 숫자범위에선 위로 볼록한 곡선을 그리고 그 이후부터는 거의 선형에 가까운 완만한 직선을 보인다. 이러한 숫자범위에서는 O(nlogn) 그래프와 상수가 큰 O(n) 그래프의 모양을 거의 구분할 수 없기때문에 그래프로는 세 알고리즘의 시간복잡도를 판별하기 어려운것을 알 수 있다.

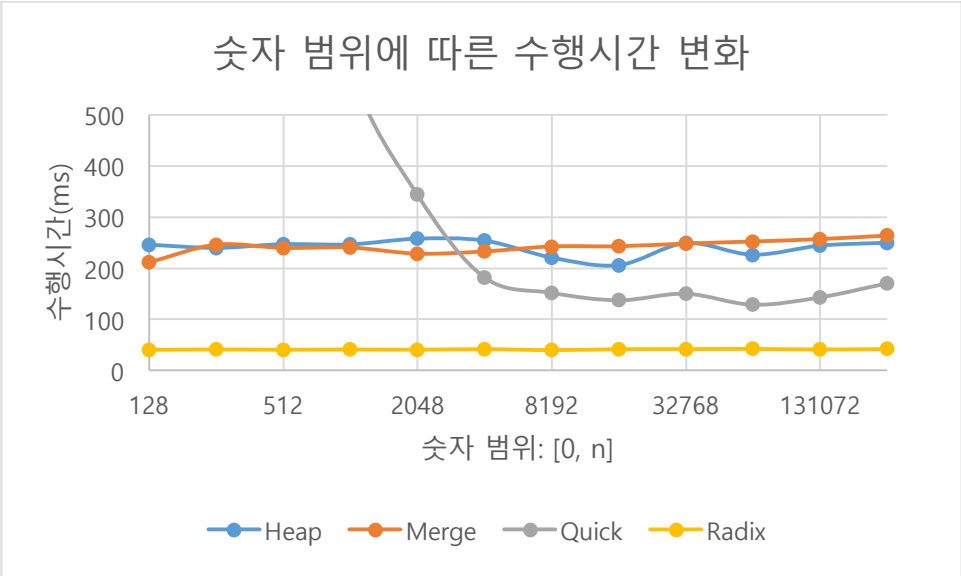
c) O(kn) : Radix sort



선형으로 완만하게 시간복잡도가 증가하는것을 알 수 있다. 단 배열 원소가 일정 수 이상으로 증가하면 아래와 같이 기울기가 한번 변화하고 또 그 이후부터는 선형으로 증가하는데, 이는 캐쉬사이즈로 인한 효과로 추측된다.



3. 숫자 범위에 따른 수행시간 변화



정렬할 원소 갯수를 100 만개로 고정시킨다음, 배열 안에 들어갈 랜덤 정수의 범위를 [0, 128]에서 [0, 32768]과 같이 다양한 범위로 실험해보자 놀라운 결과가 나왔다.

- Quick 소트를 제외한 다른 정렬알고리즘들은 숫자 범위에 따른 수행 속도 차이가 있지 않았다.
- 하지만 Quick 소트의 경우, 숫자 범위가 작을경우 Partitioning 이 나빠

져서, 수행속도가 오히려 감소하는 모습을 보였다.