**External Sorting**

대용량 데이터를 처리할 수 있는 알고리즘. 정렬 해야하는 데이터를 메모리에 올릴 수 없을 때 사용함. 보통 merge sort를 반복적으로 수행함 (External merge sort)

<방법>

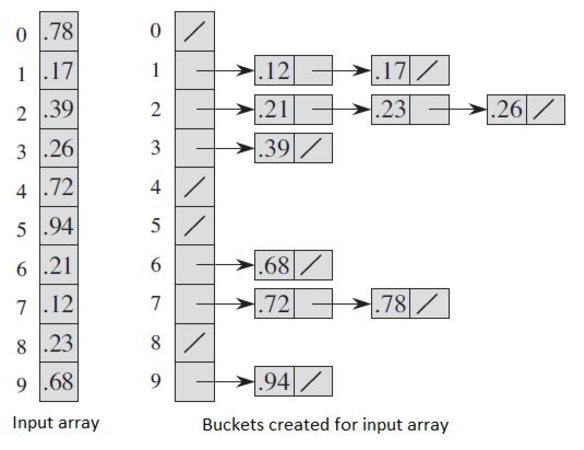
900MB를 100MB가 가능한 메모리를 이용해서 정렬하는 경우

9번의 sort를 메모리에서 하고 디스크에 저장함.

디스크에 저장된 각 9개에 대해서 10MB씩 빼서 정렬하고…

물론 bucket sort를 사용해도 External sorting을 할 수 있음.

**Bucket sort : O(n)**

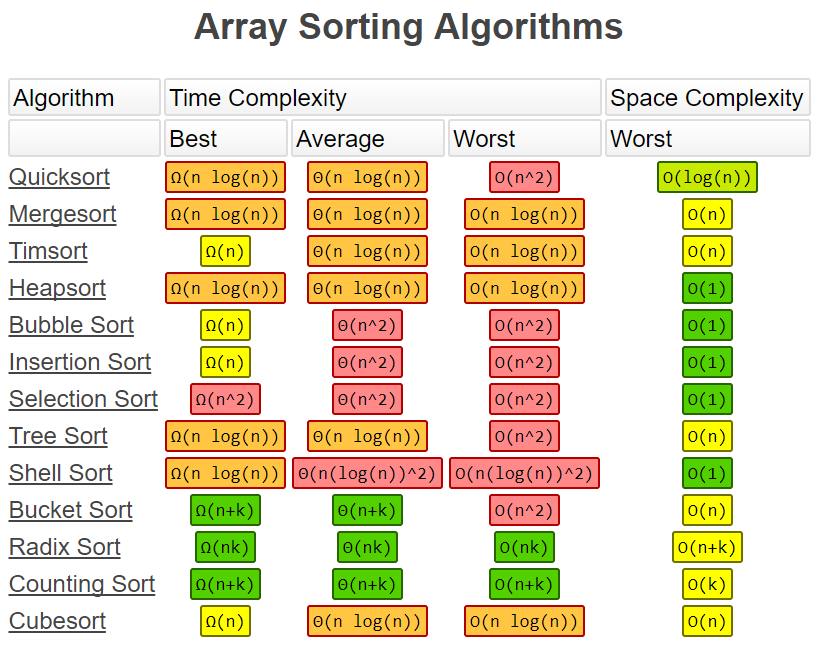


**In place sort**

정렬에서 추가적인 메모리가 필요 없는 정렬

**Stable sort**

동일한 값의 데이터가 정렬 후에도 순서를 유지하는 sorting



**Radix**

<http://blog.naver.com/PostView.nhn?blogId=markmarine&logNo=220657572364&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>

eg) 우리나라 인구 나이로 정렬할 때 씀

**Bubble sort :** Stable, In place, Best o(n)

**Selection sort :** In place, Unstable

**Insertion sort :** Best o(n), Stable, In place

Adaptive; efficient for data sets that are already substantially sorted. Time complexity is O(nk) when each element in the input is no more than k places away from its sorted position

Online; can sort a list as it receives it

**Merge sort :** Stable, Not In place

**Quick sort :** Unstable, Not In place

**Merge sort VS Quick sort**

Merge sort는 모든 경우에 n log n이고 Quick은 최악에 n^2임

하지만 보통 quick sort가 merge sort보다 좋다고 평가함.

Why? 실제 런타임에서 걸리는 시간에 영향을 미치는 요소 중 swap 횟수가 중요함.

이유는 swap을 하려면 메모리에서 데이터를 읽어야하는데 메모리에서 데이터를 읽는 행위 자체가 cpu의 성능 대신 메모리의 성능에 의존하기 때문임.

Quick sort는 little additional space만 필요하고 cache locality도 있어서 merge 보다 좋음.

Quick sort는 left랑 right를 한칸 씩 움직이니까 locality, merge는 나누니까 안되즤

또한 실제로 현대 quick sort는 최악의 케이스가 n log n임 (피봇만 잘 고르면 되니까)

**Set v List v Map**

면

**Map vs Tree**

면

메모리 측면

속도 측면

**Hash**

면

