[Sorting Algorithm]

**Selection Sort (~N^2/2)**

Method: Scan the sequence N times, each time we place the i’th smallest item in place (swap the due item with the i’th place.

Characteristics: Algorithm does not depend on the initial state of the sequence. It just blindly scans the scan for the ith’s smallest item and swap it with the item in the ith slot. The complexity is always ~N\*N/2

In practice, we almost never use Selection Sort.

**Insertion Sort (~N^2/4 on average)**

Scan the sequence N times, for each time we ensure the left i items are sorted. When we take in the (i+1)th, we swap adjacent items so that we gradually move that item forward to its due place.

**Merge Sort (NlogN)**

We always use (yes, NEED TO USE) an auxiliary array (copied from the original sequence) for extra space. For each merge operation on a range of items (from lo to hi), we copy that range of elements to the auxiliary array and merge them back to the original array.

The bottom-up merge sort uses iteration instead of recursion. It chooses step size of 2, 4, 8 and so on to sort and merge every that many of items.

Merge-Sort is stable.

**Quick-Sort**

Quick-Sort is also an N\*log(N) algorithm but it does not use auxiliary array as merge sort does. So it is an in-place sorting algorithm. On average it does 1.39\*N\*log(N) comparisons.

The worse-case is N\*N, it may be hit for two cases: 1. even if AFTER the random shuffling, the array is sorted somehow; 2. lots of duplicated itmes (for the original Tony Hoare’s algorithm).

The key of Quick-sort being N\*log(N) is the random shuffling step. It is very necessary to do this to have a probabilistic guarantee.

Improvements: median of 3 (random samples, instead of the first one) will get us a pivot element that is more likely to be close to the middle.

Use insertion sort for small array or small partitions within that array.

**Quick-Select**

It’s using the same partition method as in Quick-Sort, but can be used to find the Kth largest element from an unordered array.

**Performance of Quick-Select**: The average performance of Quick-Select is linear. Theoretically, a paper in the 70s published a linear comparison-based method to solve selection problem.

**How to fix Quick-Sort for duplicate keys?**

Use Dijkstra’s 3-way partitioning to place equal keys (to the pivot) in the middle. Practically the sorting becomes linear when there are large amount of duplicate keys.

**Heap-Sort**

With a binary heap (max-heap), we can do heap-sort for ascending order with the following two steps:

1. Make Heap: for K=N/2 to 1, since Kth element. Now the heap is built.
2. Swap the max element with the last element and keep it there, Now since the top element to ensure (N-1) size heap. Continue will the array is fully sorted.

Q: How to keep track of the maximum 5 elements from an incoming data stream?

A: We use an array of size 6 and build a binary heap (min-heap) of that size. We always keep 5 elements in the array and ensure it’s a min-heap. For every new element, we insert the new element into the binary heap, do swim operation to make sure the heap holds, then remove the minimum element. Now the 5 element in the heap are always the top 5 largest. Note that since we keep a fixed-size array, it takes constant time to maintain it.

Radix-Sort