Sort algorithm: put elements in a list with certain order.

What we care:

Time complexity

Memory complexity

Stability: if two have same rank, they will be resulted in same order as input order.

Comparison sorts:

Quick, bubble

Bubble sort:

sinking sort, iterate and swap when they are not in order. Too slow not practical. Only practical when most of them are in order.

Iterate every element to check order, go more rounds to finish sorting

Optimization: if inner loop does not cause any swap, finish (already sorted)

Then, reduce outer loop iteration by finished elements.

Selection sort:

In-comparison sort, two sub lists. one is in-order, the other is out-of-order. Find minimum in out-of-order list and put them in in-order list.

Insertion sort:

Simple, most efficient among O(n^2). Stable, in-place, online. Two sub lists. One is in-order, the other is out-of-order. Take first element from out-of-order list then put right order in in-order list.

heap sort:

improved selection sort, in-place but not stable. Create a heap to sort. Max heap: compare parent node and children nodes to swap.

Merge sort:

Efficient, general purpose, stable. Divide and conquer algorithm. Divide to many sub lists then merge them with right order.

Top down: split all then merge them

Bottom up: assume split array, then merge them

It can be implemented as parallel

Quick sort:

Efficient, divide and conquer.

Set a pivot, put elements who are less than pivot left of pivot, put others are on right.

Keep doing that until all sub lists are sorted

Issues: choice of pivot, repeated elements, optimization: set threshold then insertion sort.