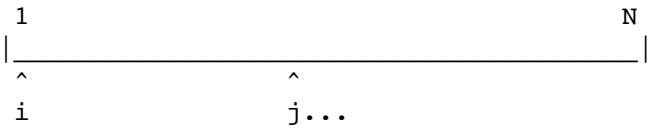
Analysis of Sorting Algorithms

Selection Sort

Find the nth smallest element and exchange it with the element in the nth position; recur.

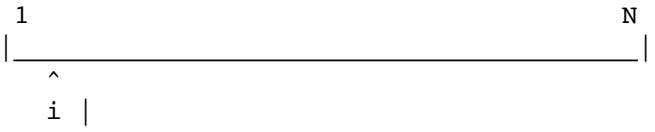


Find the smallest element and put it in place (first) by exchanging it with the first element. Then find the second smallest of those remaining and put it second, etc.

Look at every `i`. For every `i`, look at every `j`. N^2/2 comparisons and N exchanges. O(N^2). Each item is moved only once, so selection sort is good when the item records are very large relative to the sorting key.

Insertion Sort

Take the next available element and put it in its sorted place among those already sorted; recur.

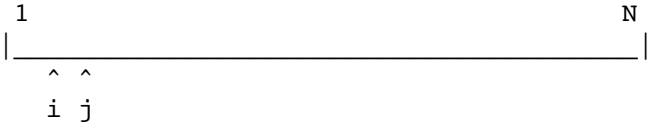


Look at each element in turn and insert it in order on the left. To make room, elements on the right may have to be shifted right.

For every `i`, compare it to every `j` (worst case). N^2/4 comparisons and N^2/8 exchanges on average. Linear for almost sorted lists.

Bubble Sort

Take the next element and pairwise sort it with the previous element; recur on the elements of the list. Then recur on the list until no pairwise changes.

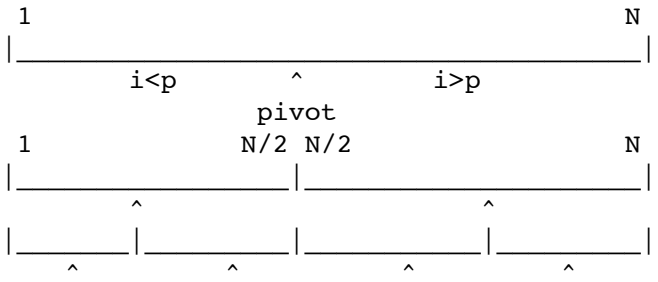


Look at each pair of adjacent elements and swap them if they are not in order. Pass through the entire list until no exchanges.

Look at every pair and sort the two if necessary. Repeat. N^2/2 comparisons and N^2/2 exchanges worst case and average case. Always slower than insertion sort.

Quicksort

Partition the list into two subsections. Sort each recursively.



Select a pivot. Exchange each `i` on the left of the pivot that is greater than the pivot with a `j` on the right of the pivot which is less than the pivot. Recur. 2NlogN comparisons on average

Mergesort

Divide the list in half, sort each half recursively, then merge the results. Merging is combining two sorted files into one sorted file. Merging two sorted files takes O(N) effort. Since the quicksort "front-end" to mergesort takes O(NlogN) steps, the cost of merging is relatively insignificant.

Hashing

The technique is to make arithmetic transformations on the search key, which then corresponds to table addresses. A **hash function** computes the key transform (examples: mod M, sum of digits). The **hash key** is then an index into a table which can be accessed in O(1) time.

Hashing is a time-space tradeoff, providing faster access by using more memory space. The storage part of a hashing algorithm is a type of bin sorting (i.e. sorting objects into bins by some criteria). The retrieval part of hashing is a type of searching (i.e. finding the right bin and then searching for the desired object within that bin).

When the hash function computes the same key for two records, the table entry is not unique. Further search is then required within the table index. This is called **collision detection**.

Radix search is similar to hashing, but the key itself is used rather than an arithmetic function of the key.