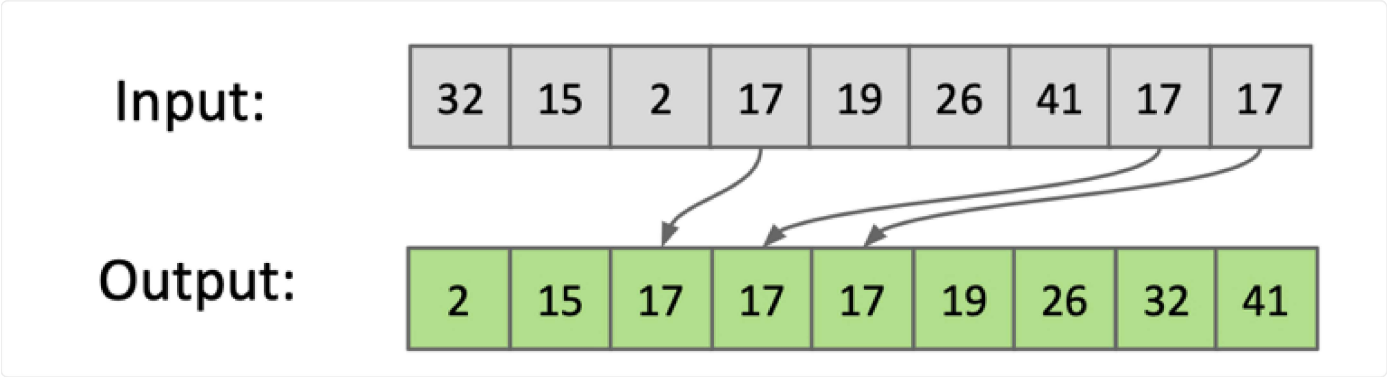
29.4 Insertion Sort

Naive Insertion Sort

In insertion sort, we start with an empty output sequence. Then, we select an item from the input, inserting into the output array at the correct location.

Naively, we can do this by creating a completely seperate output array and simply putting items from the input there. You can see a demo of this algorithm here.



Naive insertion sort with a seperate output array

For the naive approach, if the output sequence contains k items so far, then an insertion takes $O(k)$ time (shifting every item over in the output array).

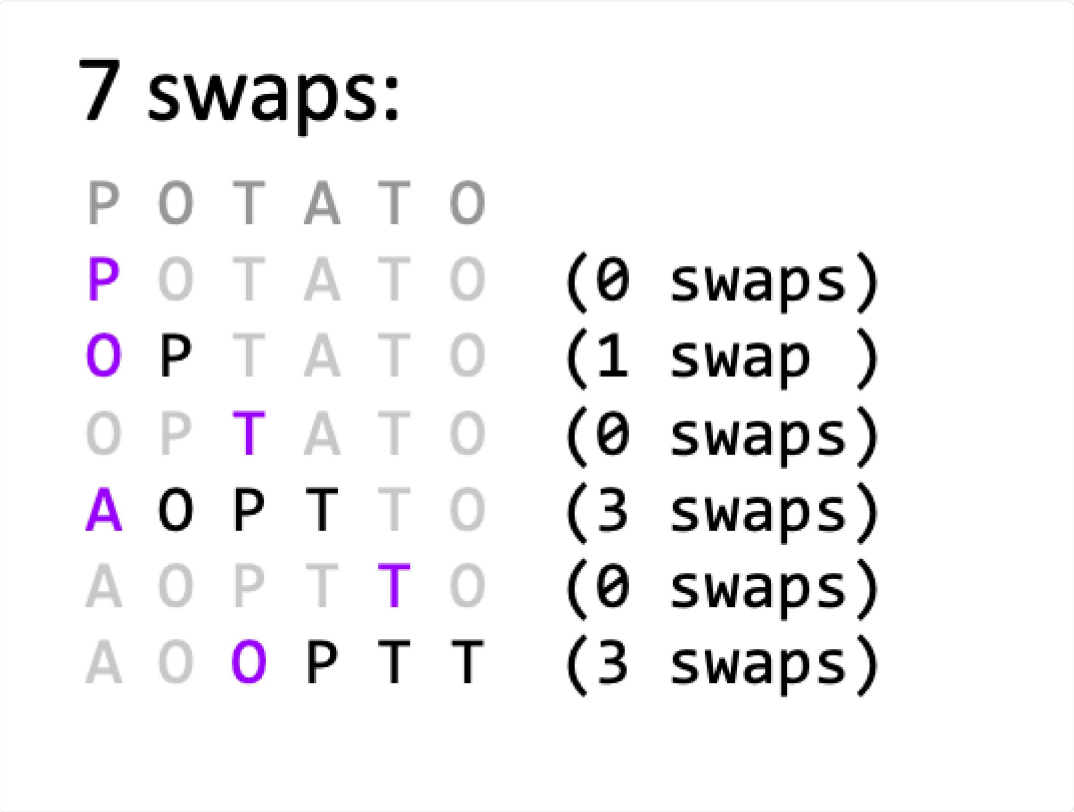
In-place Insertion Sort

We can improve the time and space complexity of insertion sort by using in-place swaps instead of building a seperate output array.

You can see a demo of this algorithm here.

Essentially, we move from left to right in the array, selecting an item to be swapped each time. Then, we swap that item as far to the front as it can go. The front of our array

gradually becomes sorted until the entire array is sorted.



Insertion sort algorithm. Purple items are the selected item being swapped to the front, and black items are the ones with which the purple items are being swapped.

Insertion Sort Runtime

The best-case runtime of insertion sort is $\Theta(N)$--when there are no swaps, we simply do a linear scan through the array. The worst-case runtime of insertion sort is $\Theta(N^2)$--in a reverse-sorted array, we have to swap every item all the way to the front.

Insertion Sort Advantages

Note that on sorted or almost-sorted arrays, insertion sort does very little work. In fact, the number of swaps that it does is equal to the number of inversions in the array.

The left array has only 5 inversions, and the right array has only 3. Insertion sort does very little work.

In other words, on arrays with a small number of inversions, insertion sort is probably the fastest sorting algorithm. The runtime is $\Theta(N + K)$, where K is the number of inversions in the array. If we define an almost-sorted array as one where the number of inversions $K < cN$ for some constant c, then insertion sort runs in linear time.

A less obvious empirical fact is that insertion sort is extremely fast on small arrays, usually of size 15 or less. The analysis of this is beyond the scope of the course, but the general idea is that divide-and-conquer algorithms (like heapsort and mergesort) spend too much time on the "dividing" phase, whereas insertion sort starts sorting immediately. In fact, the Java implementation of mergesort uses insertion sort when the split becomes less than 15 items.

Previous
29.3 Mergesort

Next
29.5 Summary

Last updated 1 year ago