29.5 Summary

	Best Case Runtime	Worst Case Runtime	Space	Demo	Notes
Selection Sort	$\Theta(N^2)$	$\Theta(N^2)$	$\Theta(1)$	Link	
Heapsort (in place)	$\Theta(N)$*	$\Theta(N \log N)$	$\Theta(1)$	Link	Bad cache (61C) performance.
Mergesort	$\Theta(N \log N)$	$\Theta(N \log N)$	$\Theta(N)$	Link	Fastest of these.
Insertion Sort (in place)	$\Theta(N)$	$\Theta(N^2)$	$\Theta(1)$	Link	Best for small N or almost sorted.

A summary of the sorts covered so far.

Overview

Inversions. The number of pairs of elements in a sequence that are out of order. An array with no inversions is ordered.

Selection sort. One way to sort is by selection: Repeatedly identifying the most extreme element and moving it to the end of the unsorted section of the array. The naive implementation of such an algorithm is in place.

Naive Heapsort. A variant of selection sort is to use a heap based PQ to sort the items. To do this, insert all items into a MaxPQ and then remove them one by one. The first such item removed is placed at the end of the array, the next item right before the end, and so forth until that last item deleted is placed in position 0 of the array. Each insertion and deletion takes O(log N) time, and there are N insertions and deletions, resulting in a O(N log N) runtime. With some more work, we can show that heapsort is θ(N log N) in the worst case. This naive version of heapsort uses θ(N) for the PQ. Creation of the MaxPQ requires O(N) memory. It is also possible to use a MinPQ instead.

In place heapsort. When sorting an array, we can avoid the θ(N) memory cost by treating the array itself as a heap. To do this, we first heapify the array using bottom-up heap construction (taking θ(N) time). We then repeatedly delete the max item, swapping it with the last item in the heap. Over time, the heap shrinks from N items to 0 items, and the sorted list from 0 items to N items. The resulting version is also θ(N log N).

Mergesort. We can sort by merging, as discussed in an earlier lecture. Mergesort is θ(N log N) and uses θ(N) memory.

Insertion Sort. For each item, insert into the output sequence in the appropriate place. Naive solution involves creation of a separate data structure. The memory efficient version of this algorithm swaps items one-by-one towards the left until they land in the right place. The invariant for this type of insertion sort is that every item to the left of position i is in sorted order, and everything to the right has not yet been examined. Every swap fixes exactly one inversion.

Insertion Sort Runtime. In the best case, insertion sort takes θ(N) time. In the worst case, θ(N^2) time. More generally, runtime is no worse than the number of inversions.

Previous
29.4 Insertion Sort

Next
29.6 Exercises

Last updated 1 year ago