TUTORIAL 9

Recap: Basic sorting algorithms.

There are 3 simple algorithms for sorting a list:

1. **Selection sort**: Find the $(i+1)^{th}$ smallest item in the array and swap it with the item in index i. Repeat until sorted.

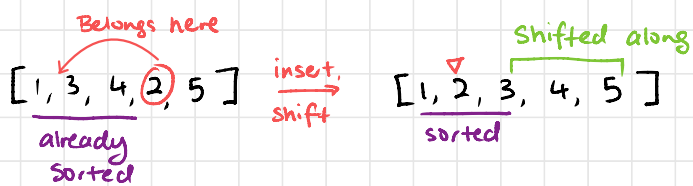
$$[1, 2, \underline{5}, 4, \underline{3}] \xrightarrow{\text{swap}} [1, 2, 3, 4, 5].$$

$\underline{5}$: 2nd index in array; $\underline{3}$: 3rd largest element

2. **Bubble sort**: Compare and swap adjacent pairs of items and swap them if they are out of order. Each pass of the list moves the biggest element to the end of the array. Repeat until sorted.

$$[1, 2, \underline{4}, \underline{3}, 5] \xrightarrow{\text{swap}} [1, 2, 3, 4, 5]$$

3. **Insertion sort**: Consider a portion $A[0 .. i]$ to be sorted, and find the correct place for $A[i+1]$ in this sorted portion. Shift along values in the array as necessary to fit $A[i+1]$ in. Repeat until sorted.



All of these algorithms are **worst case $O(n^2)$ complexity**, where n is the length of the array being sorted.

4. Selection sort:

Iteration	Array	# comparisons
0	[4 3 6 8 2]	—
1	[2 3 6 8 4]	4
2	[2 3 6 8 4]	3
3	[2 3 4 8 6]	2
4	[2 3 4 6 8]	1
		10

Bubble sort:

Iteration	Array	# comparisons
0	[4 3 6 8 2]	—
1	[3 4 6 2 8]	4
2	[3 4 2 6 8]	3
3		2
4		1
		10

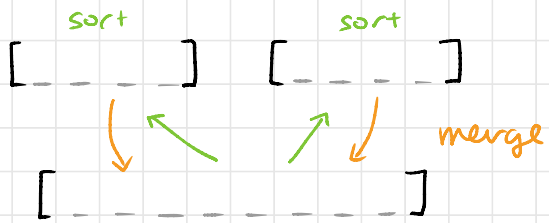
Insertion sort:

Iteration	Array	# comparisons
0	[4 3 6 8 2]	—
1	[3 4 6 8 2]	1
2	[3 4 6 8 2]	1
3	[3 4 6 8 2]	1
4	[2 3 4 6 8]	4
		7

Recap: merge sort.

Another faster sorting algorithm is merge sort. It relies on the basic principle that merging 2 sorted arrays into 1 big sorted array is efficiently doable, so if we recursively sort 2 halves of the array, they can be put back together in O(n) time.

1. Split the array in half, until the halves are of length 1
2. Take two halves and recombine them into a sorted array:



Overall complexity is O(n log n) worst case, but needs O(n) space during the merge steps for a temporary array.

5. Complete the merge phase for the array

A = [1 4 5 6 7 2 3 4 7 9]

tmp = [1 2 3 4 4 5 6 7 7 9]

Recap: quick sort.

A faste algorithm for sorting an array is **quick sort**. Given an array A[lo .. hi], we first partition and then recursively sort around it. Formally,

1. Choose an array item A[i] to be the pivot and swap it to A[lo]. There are many strategies for choosing the pivot, including just taking A[lo] itself, choosing one randomly, or taking the median of A[lo], $A\left[\frac{lo+hi}{2}\right]$ = A[mid] and A[hi].

2. Partition the array such that all values less than the pivot are to the left, and all values greate are to the right:

3. Recursively sort [< pivot] and [> pivot].

Most of the work is in the partition phase, which is a series of swaps using 2 pointers to track the parts of the array with items < pivot and > pivot (see tutorial questions for more).

Overall complexity is $O(n \log n)$ on average, but $O(n^2)$ worst case when the pivot choice is bad. Still used in practice as the observed running time is often more competitive than others. (e.g. C's inbuilt qsort() function)

b. Partition the array

[4 3 1 2 5 6 9 8 9 7]