Interview Cake

Sorting Algorithm Cheat Sheet

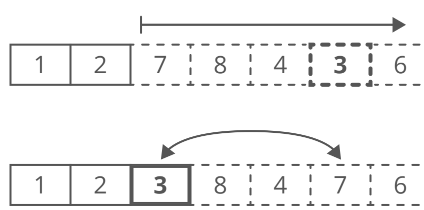
For coding interviews or computer science classes

A quick reference of the big O (/big-o-notation-time-and-space-complexity) costs and core properties of each sorting algorithm.

Expand all ()	worst	best	average	space

	worst	best	average	space
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$

Selection sort works by repeatedly "selecting" the next-smallest element from the unsorted array and moving it to the front.



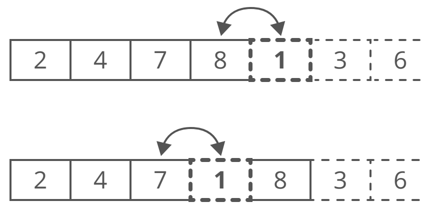
Full selection sort reference ➞

Expand all ()	worst	best	average	space

Insertion Sort

worst	best	average	space
$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$

Insertion sort works by inserting elements from an unsorted array into a sorted subsection of the array, one item at a time.

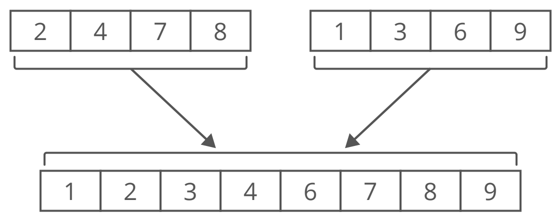


Full insertion sort reference ➞

Merge Sort

worst	best	average	space
$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$	$O(n)$

Merge sort works by splitting the input in half, recursively sorting each half, and then merging the sorted halves back together.



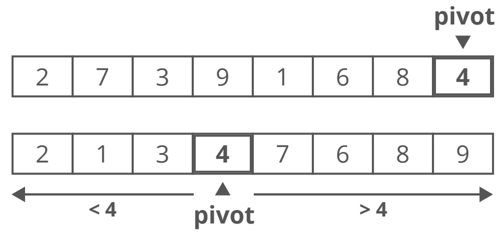
Full merge sort reference ➞

Quicksort

worst	best	average	space
$O(n^2)$	$O(n \lg n)$	$O(n \lg n)$	$O(\lg n)$

Expand all ()	worst	best	average	space

Quicksort works by recursively dividing the input into two smaller arrays around a pivot item: one half has items smaller than the pivot, the other has larger items.

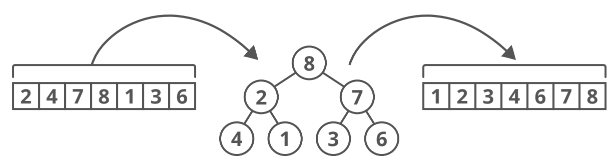


Full quicksort reference ➞

Heapsort

	worst	best	average	space
Heapsort	$O(n \lg n)$	$O(n)$	$O(n \lg n)$	$O(1)$

Heapsort is similar to selection sort—we're repeatedly choosing the largest item and moving it to the end of our array. But we use a heap to get the largest item more quickly.

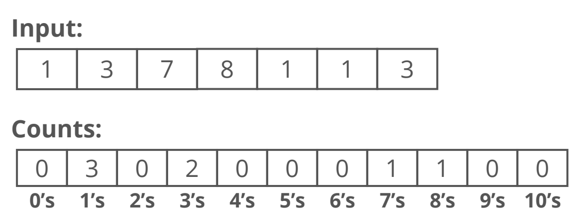


Full heapsort reference ➞

Counting Sort

	worst	best	average	space
Counting Sort	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Counting sort works by iterating through the input, counting the number of times each item occurs, and using those counts to compute each item's index in the final, sorted array.



Full counting sort reference ➞

Expand all ()	worst	best	average	space

	worst	best	average	space
Radix Sort	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Radix sort works by sorting the input numbers one digit at a time.

2**3**	3**1**	8**1**	2**7**	5**6**	3**7**	5**1**

3**1**	5**1**	8**1**	2**3**	5**6**	2**7**	3**7**

Full radix sort reference →

Which Sorting Algorithm Should I Use?

It depends. Each algorithm comes with its own set of pros and cons.

- Quicksort is a good default choice. It tends to be fast in practice, and with some small tweaks its dreaded $O(n^2)$ worst-case time complexity becomes very unlikely. A tried and true favorite.
- Heapsort is a good choice if you can't tolerate a worst-case time complexity of $O(n^2)$or need low space costs. The Linux kernel uses heapsort instead of quicksort (https://github.com/torvalds/linux/blob/master/lib/sort.c#L194) for both of those reasons.
- Merge sort is a good choice if you want a stable sorting algorithm (/concept/stable-sort). Also, merge sort can easily be extended to handle data sets that can't fit in RAM, where the bottleneck cost is reading and writing the input on disk, not comparing and swapping individual items.
- Radix sort *looks* fast, with its $O(n)$worst-case time complexity. But, if you're using it to sort binary numbers, then there's a hidden constant factor that's usually 32 or 64 (depending on how many bits your numbers are). That's often *way* bigger than $O(\lg(n))$, meaning radix sort

tends to be slow in practice.

- Counting sort is a good choice in scenarios where there are small number of distinct values to be sorted. This is pretty rare in practice, and counting sort doesn't get much use.

Each sorting algorithm has tradeoffs. You can't have it all.

So you have to know *what's important* in the problem you're working on. How large is your input? How many distinct values are in your input? How much space overhead is acceptable? Can you afford $O(n^2)$ worst-case runtime?

Once you know what's important, you can pick the sorting algorithm that does it best. Being able to compare different algorithms and weigh their pros and cons is the mark of a strong computer programmer and a definite plus when interviewing.