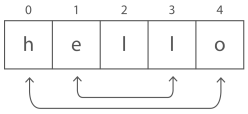
Getting started

Solve it "by hand" on the whiteboard

Notice the process you use. Look for patterns, and think about how to implement your process in code.

Reversing a string:



Hmmm . . . it looks like we're swapping pairs of characters,

Solve a simpler version of the problem

Remove or simplify one of the requirements of the problem. Once you have a solution, see if you can adapt that approach for the original question.

Trying to find the k-largest element in a set: Walk through finding the largest *element, then the* second *largest, then the* third *largest. Finally, generalize your approach to find the* k-largest.

Start with an inefficient solution

Even if it feels stupidly inefficient, it's often helpful to start with something that'll return the right answer. From there, you just have to optimize your solution.

Trying to find the combined median of two lists of sorted numbers? It's obviously not the most efficient answer, but you could just concatenate the arrays together, sort that new array, and return the middle item.

Finding optimizations

Look for repeat work

If your current solution goes through the same data multiple times, you're doing unnecessary repeat work. See if you can save time by looking through the data just once.

Are you walking through your array multiple times to try to find specific items? Instead, you could convert the array to a lookup table to dramatically improve your runtime.

Find hints in the problem

Details about the problem can carry huge hints about the solution. If it didn't matter, your interviewer wouldn't have brought it up. It's a strong sign that the best solution to the problem exploits it.

Suppose you're asked to find the first occurrence of a number in a sorted array. The fact that the array is sorted is a strong hint—take advantage of that fact by using a binary search.

Throw some data structures at it

Look at the requirements of the problem and ask yourself if there's a data structure that has those properties.

Can you save time by using the fast lookups of a hash table? Can you express the relationships between data points as a graph?

Establish bounds on space and runtime

Think out loud about the parameters of the problem. Try to get a sense for how fast your algorithm *could possibly* be.

I have to at least look at all the items, so I can't do better than O(n) time
"The answer will contain n^2 items, so I must at least spend that amount of time."