Here are three tips to Solve General data structure and algorithm problems :-

**step one - give yourself enough time to clearly understand the given question**

- Of course the first one is to read the question carefully. Then look out for specific keywords. for example if the question mentions something like:

-> searching in sorted array - think of binary search

->A duplicate element - thing of HashSet or Set depending on the language

-> reversing an elements or string - Stack

-> nth element in linked list, starting of circle in cyclic linked list - two pointer(fast and slow pointers)

->find the smallest\largest element - thing of Min\Max heap

-> All possible combination of elements, permutation, get the nth value - can be solved by recursion but keep in mind that any recursive solution can be implemented by iterative approach thought sometimes can be very hard to.

-> level by level tree transversal, tracking each level of tree - Breadth-First Search(BFS), queue

-> space and time constrains - Optimization, Dynamic programming

for more detailed explanation on looking pattern in solving the coding problems look at this link below.

[Grokking the Coding Interview: Patterns for Coding Questions](https://www.educative.io/courses/grokking-the-coding-interview?affiliate_id=5073518643380224)

**step two - Breaking down the Question**

lets say the question is asking to write some function. what you will do first is to write down what is the input type, output expected, example input and output(to help you understand the question better), then the data structure and algorithm you are going to use (you already have something in your mind from the first tip) and at last write down a pseudo-code in simple English on how you can solve the problem. After this, it is simple to convert your pseudo-code solution to any desire programming language.

**step three - Optimizing your solution and Check for Edge case**

Most of the time optimization is not an easy as it involves deeper understanding of various data structure and algorithm and it usually comes naturally with experience of solving many problems. And Even more sometimes you may also need to know a specific algorithm to efficiently solve a given problem. for example, one popular question is [finding the largest sum of contiguous sub-array of given array in O(n) time complexity and return the sub-array list](https://leetcode.com/problems/maximum-subarray/). In order to solve this problem easily you need to have a prior knowledge of [Kadane's algorithm](https://en.wikipedia.org/wiki/Maximum_subarray_problem" \l "Kadane%27s_algorithm" \o "en.wikipedia.org" \t "_blank). The important thing is to look your finish solution again and try to find a room for optimization. Thing if you can find a way to do it in O(n) to your O(n^2) solution or in nLogn to O(n^2) solution. Whether if you can do it in place O(1) without extra space. Look on where you are repeating the calculation and try to cache that. Those things are usually build up on experience and like I said it will naturally come to you as you face and try to solve lots of problems. Coming to edge case most of time it's written to properly filter inputs\arguments before processing it. for example, prior checking for empty or null value or value out of specific range e.g. negative number may not be accepted by the function. these things are important to be handled correctly as it can fail your solution catastrophically.

**Step -four - Nothing beats practice, so practice, practice and practice….**