**Data Structure Questions**

For this project, you will answer the **seven** questions laid out in the next sections. The questions cover a variety of topics related to the basic algorithms you've learned in this course. You will write up a clean and efficient answer in Python, as well as a text explanation of the efficiency of your code and your design choices.

A qualified reviewer, such as a software or data engineer, will look over your answer and give you detailed, written feedback on areas you're doing very well in and areas that could be improved. For example, is your solution the most efficient one possible? Are you doing a good job of explaining your thoughts? Is your code elegant and easy to read?

# Finding the Square Root of an Integer

Find the square root of the integer without using any Python library. You have to find the floor value of the square root.

For example if the given number is 16, then the answer would be 4.

If the given number is 27, the answer would be 5 because sqrt(5) = 5.196 whose floor value is 5.

The expected time complexity is O (log (n))

# Search in a Rotated Sorted Array

You are given a sorted array which is rotated at some random pivot point.

Example: [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]

You are given a target value to search. If found in the array return its index, otherwise return -1.

You can assume there are no duplicates in the array and your algorithm's runtime complexity must be in the order of O (log n).

Example:

Input: nums = [4,5,6,7,0,1,2], target = 0, Output: 4

# Rearrange Array Elements

Rearrange Array Elements so as to form two number such that their sum is maximum. Return these two numbers. You can assume that all array elements are in the range [0, 9]. The number of digits in both the numbers cannot differ by more than 1. You're not allowed to use any sorting function that Python provides and the expected time complexity is O(nlog(n)).

for e.g. [1, 2, 3, 4, 5]

The expected answer would be [531, 42]. Another expected answer can be [542, 31]. In scenarios such as these when there are more than one possible answers, return any one.

# Dutch National Flag Problem

Given an input array consisting on only 0, 1, and 2, sort the array in a single traversal. You're not allowed to use any sorting function that Python provides.

**Note:** O(n) does not necessarily mean single-traversal. For e.g. if you traverse the array twice, that would still be an O(n) solution but it will not count as single traversal.

# Autocomplete with Tries

# Building a Trie in Python

Before we start let us reiterate the key components of a Trie or Prefix Tree. A trie is a tree-like data structure that stores a dynamic set of strings. Tries are commonly used to facilitate operations like predictive text or autocomplete features on mobile phones or web search.

Before we move into the autocomplete function we need to create a working trie for storing strings. We will create two classes:

* A Trie class that contains the root node (empty string)
* A TrieNode class that exposes the general functionality of the Trie, like inserting a word or finding the node which represents a prefix.

Give it a try by implementing the TrieNode and Trie classes below!

# Finding Suffixes

Now that we have a functioning Trie, we need to add the ability to list suffixes to implement our autocomplete feature. To do that, we need to implement a new function on the TrieNode object that will return all complete word suffixes that exist below it in the trie. For example, if our Trie contains the words ["fun", "function", "factory"] and we ask for suffixes from the fnode, we would expect to receive ["un", "unction", "actory"] back from node.suffixes().

Using the code you wrote for the TrieNode above, try to add the suffixes function below. (Hint: recurse down the trie, collecting suffixes as you go.)

# Testing it all out

Run the following code to add some words to your trie and then use the interactive search box to see what your code returns.

# Max and Min in a Unsorted Array

In this problem, we will look for smallest and largest integer from a list of unsorted integers. The code should run in O(n) time. Do not use Python's inbuilt functions to find min and max.

**Bonus Challenge:** Is it possible to find the max and min in a single traversal?

Sorting usually requires O(n log n) time Can you come up with a O(n) algorithm (i.e., linear time)?

# HTTP Router using a Trie

For this exercise we are going to implement an HTTP Router like you would find in a typical web server using the Trie data structure we learned previously.

There are many different implementations of HTTP Routers such as regular expressions or simple string matching, but the Trie is an excellent and very efficient data structure for this purpose.

The purpose of an HTTP Router is to take a URL path like "/", "/about", or "/blog/2019-01-15/my-awesome-blog-post" and figure out what content to return. In a dynamic web server, the content will often come from a block of code called a handler.

**[[](https://classroom.udacity.com/nanodegrees/nd256/parts/da17de0f-f834-46f8-bb48-ee2705d95dc4/modules/bd252a0b-e9e7-473b-bcc1-bc7e3153568b/lessons/8ec390d0-e99d-44c0-88f9-f8f9faf467fc/concepts/8a8492a5-f76d-4dcc-a07c-3e2a2ca9b3c6)](https://classroom.udacity.com/nanodegrees/nd256/parts/da17de0f-f834-46f8-bb48-ee2705d95dc4/modules/bd252a0b-e9e7-473b-bcc1-bc7e3153568b/lessons/8ec390d0-e99d-44c0-88f9-f8f9faf467fc/concepts/8a8492a5-f76d-4dcc-a07c-3e2a2ca9b3c6)**

First we need to implement a slightly different Trie than the one we used for autocomplete. Instead of simple words the Trie will contain a part of the http path at each node, building from the root node /

In addition to a path though, we need to know which function will handle the http request. In a real router we would probably pass an instance of a class like Python's **[SimpleHTTPRequestHandler](https://docs.python.org/3/library/http.server.html" \l "http.server.SimpleHTTPRequestHandler" \t "_blank)** which would be responsible for handling requests to that path. For the sake of simplicity we will just use a string that we can print out to ensure we got the right handler

We could split the path into letters similar to how we did the autocomplete Trie, but this would result in a Trie with a very large number of nodes and lengthy traversals if we have a lot of pages on our site. A more sensible way to split things would be on the parts of the path that are separated by slashes ("/"). A Trie with a single path entry of: "/about/me" would look like:

(root, None) -> ("about", None) -> ("me", "About Me handler")

We can also simplify our RouteTrie a bit by excluding the suffixes method and the endOfWord property on RouteTrieNodes. We really just need to insert and find nodes, and if a RouteTrieNode is not a leaf node, it won't have a handler which is fine.

Next we need to implement the actual Router. The router will initialize itself with a RouteTrie for holding routes and associated handlers. It should also support adding a handler by path and looking up a handler by path. All of these operations will be delegated to the RouteTrie.

Hint: the RouteTrie stores handlers under path parts, so remember to split your path around the '/' character

Bonus Points: Add a not found handler to your Router which is returned whenever a path is not found in the Trie.

More Bonus Points: Handle trailing slashes! A request for '/about' or '/about/' are probably looking for the same page. Requests for '' or '/' are probably looking for the root handler. Handle these edge cases in your Router.