You can use any and all existing python modules and python functions to assist you in these.

Commit a repository called implementations to https://classroom.github.com/a/IcS_PZtR. Include a folder for each week. You will received feedback throughout the term and you may keep improving them over the term. They will only be graded at the end of term.

Week 1

1. Implement a basic measure of tracking the time taken to run an algorithm.
2. Implement a basic measure of tracking the space used by an algorithm.
3. Implement the following 3 algorithms:
    a. Find the maximum value in a list
    b. Make each letter in a string lowercase
    c. Sort a list of integers (using the inbuilt python method)
4. For each of the above algorithms, **plot graphs** showing how the time and space taken as the input size changes from length 1 to length 100. (Lengths of lists and lengths of strings). You don't need all values - just a few (e.g. 3) is enough.
5. **Estimate** how long each algorithm would take for for inputs of size 1,000,000. Write your estimate clearly in the code.

Useful modules to import: time, timeit, memory_profiler, heapy, pympler, random, numpy, matplotlib

Other Resources: **StackOverFlow**, **Another Stack OverFlow**

Week 2
Implement the following functions and create multiple unit tests showing that they work for all possible cases. Discuss the time and space complexity of each. Make sure to use names of functions which make sense.

6. Input a string of text and output the same text replacing all whitespace with just a single space. E.g. f("hello   my   friend") outputs "hello my friend
7. Input a string and return a dictionary with the number of occurrences of each character in the string. E.g. f("hi") outputs {'h':1,'i':1}
8. Input a list of lists and return a single list with each element occurring exactly once.
9. Input 2 lists - listA with n elements and listB which has all elements of listA except one (but the rest are in the same order). Outputs the missing element. E.g. f([8,1,2,3],[8,1,3]) outputs 2
10. Input an integer n and outputs a list with 1 once, 2 twice, 3 three times….n n-times e.g. f(3) = [1,2,2,3,3,3]

1st Attempt Implementation Deadline - **Friday 31st January**

Week 3

Implement the following creating multiple unit tests showing they work for all possible cases. Discuss the time and space complexity of each. You can use any built-in types to help you

1. Create a simple array class where each element must be an integer type and:
    a. has a .len() method which outputs the length of the array
    b. has a .get(i) method which outputs the value at a given index
    c. has a a .set(val,i) method which replaces the val at index i
2. Create a dynamic array subclass which also has the following basic methods:
    a. add(val) which adds an element to the end
    b. del() which removes the last element of the array
3. Define the following functions using your dynamic array class as an input. Use only the methods that you have defined (len, get, set, add, del) - not built-in methods.
    a. contains(array, val) which returns True if val is in the array
    b. reverse(array) which reverses the entire array (outputting a new array)
    c. insert(array, val, i) which inserts a value at the given index
4. Build an alist (associative list) which has the following methods
    a. add(key,value)
    b. remove(key)
    c. modify(key,newvalue)
    d. lookup(key)

1st Attempt Implementation Deadline - **Friday 7th February**

Week 4

Implement the following creating multiple unit tests showing they work for all possible cases. Discuss the time and space complexity of each. You can use any built-in types to help you. **DO NOT copy and paste or GOOGLE** how to do these. Try to solve these yourself using your own knowledge. Try to understand what methods are necessary and how you would implement them.

1. Implement a stack class and all of the relevant primary methods.
2. Implement a queue class and all of the relevant primary methods.

1st Attempt Implementation Deadline - **Friday 14th February**

Week 5

Implement the following creating multiple unit tests showing they work for all possible cases. Discuss the time and space complexity of each. **DO NOT copy and paste or GOOGLE** how to do these. Try to solve these yourself using your own knowledge. Try to understand what methods are necessary and how you would implement them. **Do not use other built-in classes for these.**

1. Implement a linked-list class (with a tail reference) and all of the relevant primary methods. You will need to build a node class to help you.
2. Implement a doubly-linked list class and implement the following methods
    a. doublylinkedlist.insert(node, prevNode) - insert node after prevNode
    b. doublylinkedlist.delete(node) - delete node from doublylinkedlist
3. Use either the linked-list class or the doubly-linked list class to implement a stack class
4. Use either the linked-list class or the doubly-linked list class to implement a queue class

1st Attempt Implementation Deadline - **Friday 21st February**

Week 6

Build the following functions using recursion. Be sure to write unit tests and analyse space/time complexity. Any built-in or imported classes are fine to use. Make sure to use recursion. Use Google to help you understand the problem but don't look for or use code you find online.

1. Create a linked-list.search(val) method from your linked-list class which searches for the value within your linked-list.
2. Write a gcd(m,n) function which finds the greatest common divisor of positive integers m and n. You can use the Euclidean Algorithm to help you.
3. Write a fibonacci(n) function that finds the nth Fibonacci Number
4. Write a det(A) function that takes an nxn matrix A (can be a list of lists, numpy array, pandas data frame etc you chose) and finds the determinant.
5. Write a binarySearch(array, val) function that inputs a sorted array/list (ascending order) and returns if the value is in the array using BinarySearch (should be O(logN) time)

1st Attempt Implementation Deadline - **Friday 6th March**

Week 7

Write unit tests for these and discuss the time and space complexity.

1. Write a search(array,item) function that returns True if item is in array and False otherwise. Do not use in-built solutions like in, contains, find etc. You may use a loop. Use linear/sequential search.
2. Write a SpecialSort(alist) function that inputs an association list (each element of the alist has a key and a value - both integers) and sorts by the value and if the values are the same, then the key. You may use in-built functions like sort and sorted to help you. This may help.

3. Implement 2 sorting algorithms of your choice. It should input an array of integers and output the same array with the integers in ascending order. You should not use in-built sorting functions - write your own..

1st Attempt Implementation Deadline - **Friday 13th March**

<u>Week 8</u>
Write unit tests for these and discuss the time and space complexity.

1. Write a function that inputs a list of lists and returns if it can form a tree with the root as the first element of the list and each other element a subtree.
2. Create BinaryTreeNode and BinaryTree Classes including the primary methods.
3. Write an algorithm to compute the height of the binary tree.
4. Write the following 3 traversal methods (using recursion) which input binary trees (as defined above) and output a list of integers:
    a. Pre-order traversal
    b. In-order traversal
    c. Post-order traversal
5. Write a function that inputs a list of integers and outputs a Binary Search Tree. Confirm that this works by inputting your BST, applying the in-order traversal and seeing that you get back a sorted list of the same integers.

1st Attempt Implementation Deadline - **Friday 27th March (Not strict but please attempt if you have internet)**

<u>Week 9</u>
Write unit tests for these and discuss the time and space complexity.

1. Write a function that inputs a list of integers and returns True if it is a min heap (parents are always less than children) and False if not.
2. Write a function that inputs a list of integers and swaps elements in place until the list is now a min heap.
3. Create a min-heap class which is initialised by a list of integers and then applies the function in question 2. Add the following methods
    a. GetMinimum() - outputs the minimum value in the heap.
    b. ExtractMinimum() - removes minimum from the heap (and returns the min)
    c. Insert(val) - inserts value into the heap (at the last place)
    d. Delete(index) - deletes value at a specific index from the heap.
4. Write a heap sort function (you can use what you did in Week 7 if you already did it)

1st Attempt Implementation Deadline - **Friday 3rd April (Not strict but please attempt if you have internet)**

<u>Week 10</u>
Write unit tests for these and discuss the time and space complexity.

1. Create a Directed Weighted Graph class using Adjacency Lists along with the relevant primary methods. (Insert/Delete edge, Insert/Delete vertex, Update weight)
2. Create an Undirected Weighted Graph class using Adjacency Matrices along with relevant primary methods. (Insert/Delete edge, Insert/Delete vertex/Update weight)
3. Write a function FindPath(v1,v2) that inputs two vertices in your directed weighted graph and finds a path from v1 to v2 outputting an array of edges starting from v1 and ending at v2 such that there is a directed edge from v1 to the next and so on until v2. Return False if no path exists.
4. Write a function FindShortestLength2(UWG) that inputs an Undirected Weighted Graph and finds the shortest (in terms of sums of weight) path of length 2 outputting the 3 vertices that form the path of length 2. Edges may not be repeated in the path.
5. Write a function FindFurthest(DWG, v, metric) where DWG is a directed-weighted-graph type, v is a vertex in the DWG, and metric is either:
    a. ‘level’ - in which you should find a vertex in DWG that is reachable using directed edges from v but is the furthest away in terms of number of edges.
    b. ‘weight’ - in which you should find a vertex in DWG that is reachable using directed edges from v but is the furthest away in terms of the sum of the weight of edges in the shortest path to it.

1st Attempt Implementation Deadline - **Friday 17th April (Not strict but please attempt if you have internet)**