**Homework 3 Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

AuE 8930: Computing and Simulation for Autonomy

Instructor: Prof. Bing Li, Clemson University, Department of Automotive Engineering

\* Refer to [Syllabus](https://docs.google.com/document/d/1ekwIf3SZGUO1toGO_iUMmZqzQb7wOpZjfq427YnCjyc/edit?usp=sharing) for homework (late) submission, grading and plagiarism policies;

\* Submission due Mon. 10/9/2023 11:59 pm via Canvas, include:

* This document (with answers), and with your program results/visualization;
* A .zip file of (modified) source code and data if any, which the TA might run.

\* All time complexity should in the big O notation.

\* Control your code versions in Github or Bitbucket git repositories.

\* Code template is in [this Github repo](https://github.com/fengziyue/CU-Computing-Autonomy/tree/master/Homework3), which can be used as a baseline for your homework.

\* If you complete this homework using C++, your final grade will be with a bonus scale 105%.

In this case, for the question templates, you’ll also need to transfer it into C++.

\* Development/coding environment for your programming:

* + IDE like [PyCharm](https://www.jetbrains.com/help/pycharm/installation-guide.html) for Python with [Anaconda](https://www.anaconda.com/products/distribution) as Python installations, or
  + CLion for C++ if use C++, or any other tools you prefer.

\* The extra questions are optional. You max score is capped as 100.

\* This homework includes two parts, and you are supposed to complete both A and B parts.

Part-A

**Question 1 (10’)**

Given an array of integers, find two numbers in it such that they can add up to a specific number.

You may assume there are exactly one solution, you can’t use the same element twice. (Only time-complexity optimized solution gets full grade)

Example:

Given [2, 7, 11, 4], Target = 13.

The answer is 2 and 11.

Modify the “solution” function in the question1.py.

(Analyze your time complexity)

**Question 2 (10’)**

Given a binary tree, find the max depth of it. Modify the “solution” function in the question2.py (Analyze your time complexity, and only time-complexity optimized solution gets full grade)

**Question 3 (5’)**

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example:

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

Explanation: 342 + 465 = 807.

Modify the “solution” class in question3.py, you may design your input to test it.

**Question 4 (5’)**

Given a string s, find the length of the longest substring without repeating characters. You can expect the string length is less than 100, and only contains English letters.

Example 1:

Input: s = "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

Modify the “solution” class in question4.py, you may design your input to test it.

**Question 5 (5’)**

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Every close bracket has a corresponding open bracket of the same type.

Modify the “solution” function in the question5.py. (Analyze your time complexity)

**Question 6 (5’)**

Use OpenCV to do a bilateral filter to an image, modify from question6.py, you may use your favorite image, visualize the images before and after the filtering using matplotlib.

**Question 7 (10’)**

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. (Note: A leaf is a node with no children.)

Example:

Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ \

7 2 1

return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

Modify the “solution” class in question7.py, test the above example and design your test case.

**Question 8 (10’)**

Given two strings s and t, return true *if* t *is an anagram of* s*, and* false *otherwise*.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Example 1:**

**Input:** s = "anagram", t = "nagaram"

**Output:** true

**Example 2:**

**Input:** s = "rat", t = "car"

**Output:** false

**Constraints:**

1 <= s.length, t.length <= 5 \* 104

s and t consist of lowercase English letters.

Modify the “solution” function in the question8.py. (Analyze your time complexity)

**Extra Question 1 (2’)** (***Extra***: means it is optional for you to do)

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*.

Example:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: [4,-1,2,1] has the largest sum = 6.

You may design your input to test it.

**Extra Question 2 (2’)**

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example:

Input: lists = [[1,4,5],[1,3,4],[2,6]]

Output: [1,1,2,3,4,4,5,6]

Modify the “solution” class in extra\_question2.py, you may design your input to test it.

**Extra Question 3 (2’)**

Write a NumPy program to get the values and indices of the elements that are bigger than 10 in a given array. - Modify the extra\_question3.py

**Extra Question 4 (2’)**

Use template matching with OpenCV to find Messi’s face in an image, try all 6 methods and plot the result. - Modify the extra\_question4/code.py

**Extra Question 5 (2’)**

Write a NumPy program to add, subtract, multiply, divide two arrays element-wise.

After you *import numpy*

The first step is to use NumPy to create two arrays: *a*, *b*

*a* and *b* should be the same dimensioned.

You initialize the values for *a* and *b* when you use *numpy.array* to create them.

Then *a* and *b* can apply the *+ - \* /* , like in the ways that two integers can do.

*numpy* will do the corresponding element-wise math operations implicitly.

Please do it for both *a* and *b* for two cases: they are 1D arrays and 2D arrays, such as *a* and *b* both are 4x1, and both are 4x4.

**Extra Question 6 (2’)**

Use SciPy for an application of Discrete Fourier Transform (DFT), modify the extra\_question6.py, and apply DFT to the array “a” and visualize both original and result signals.

Part-B

**Demo existing and revise search algorithms (40’)**

* Reference code repo:

<https://github.com/fengziyue/CU-Computing-Autonomy/tree/master/Homework3/map-path-search>

* The TA will run and test your algorithms results using the GUI;

[a] For this question, you have the existing reference:

Occupancy gridmap class library:

*Homework3/map-path-search/gridmap.py*

Occupancy gridmap-based A\* (A-start path searching algorithm) implementation:

*Homework3/map-path-search/a\_star\_occupancy.py*

As the default, you can use the 8 connectivity for this whole question.

1) Demo existing reference and get to know the behaviour of its path search. (2’)

The demo run file is: *examples/occupancy\_map\_8n.py*

2) Implement occupancy gridmap-based Dijkstra for same functionality as (1a) (18’)

If you prefer, you can use this as the template to revise:

*Homework3/map-path-search/a\_star\_occupancy.py*

[b] For this question, you have the existing reference:

Quadtree-map class library:

*Homework3/map-path-search/quadtreemap.py*

Quadtree map-based Dijkstra path searching algorithm implementation:

*Homework3/map-path-search/dijkstra\_quadtree.py*

3) Demo existing reference and get to know the behaviour of its path search. (2’)

The demo run file is: *examples/quadtree\_map\_8n.py*

4) Implement Quadtree map-based A\* for same functionality as (2a) (18’)

If you prefer, you can use this as the template to revise:

*Homework3/map-path-search/dijkstra\_quadtree.py*

[c] Extra credits (optional to complete):

* 5) Try a few of different granularity, and describe the potential affect (2’);
* 6) Show both Dijkstra and A\* algorithms into the GUI for the same mouse events (2’);
* 7) Analyze your time complexity of each algorithm (2’);

Below are some visualization hints for your references:

 