

## CSP2348 Data Structures

## Assignment 3: Paired programming project

- A mini-programming project in Python
- Video presentation: demonstration and reflection

### Objectives from the Unit Outline

- Apply mathematical theory to algorithm analysis.
- Design algorithms using ADT and various data structures.
- Employ Python classes to encapsulate ADT and implement algorithms.

### General Information:

- Assignment 3 (or A3 for short) is the major assessment of the unit. It is a group assignment with teamwork and individual work. The teamwork component is a **team assessment** to be completed by a group of **up to two students**. Your group needs to complete a mini programming project and write a project report. The individual work component is a **video presentation** to be done by individual team members. Therefore, two batches of submissions are required.
  - The first batch of submission includes submission of the project implementation (including source code/s, and executables if applicable) and a written report. This part is a *group* task so it should be completed by all group members and only submitted by the team leader (that is, the other team member should *not* submit it repeatedly).
  - The second batch of submission consists of one *video link* and an *optional Peer-Review* document. This is an individual task/work so each team member of the group should prepare and submit their own version of the work separately. The video link is the web link to a video presentation, which should contain two parts: a demonstration of the project completed by the group, and an individual's *reflection* on the project work. You can also include an (optional) *peer-review* document within this batch of submission.
- Group formation for A3:
  - Groups for Assignment 3 will be randomly assigned in week 6. If you want to form your own team with a friend or classmate you know, you can either self-sign in a group via A3 group sign-up page on Canvas (via "People->Groups" section) or email your tutor your team members' details so that your tutor can help you set up a group (note that this should be done by week 6).
  - Once a group is formed, the team members will be locked on Canvas. - If you want to change the team members of your group, you must email your tutor to request a change. Group membership can only be changed within the first 8 weeks.
  - If you really wanted to do A3 yourself, that's fine. In such a case, you must let your tutor know your decision by or before week 8 and you should not expect a reduction in the workload of the assignment.
- The programming project consists of three tasks: The first task requests you to design and implement algorithms to generate BSTs with special features. The second task is to modify some existing tree-based algorithms/methods to implement some specified application scenario/s. The third is a dummy mini-project using AVL-tree ADT. Video presentations (to be submitted separately) are primarily for demonstrating your project completed and for a personal reflection on the work completed.

**Due:** (See Due Date in Assignments Section on Canvas)

**Marks: 100 marks** (which will be converted to 50% of unit mark)

# 1. The project

The project is divided in to three questions:

## Q1: Balanced BST and complete BST (15 marks)

**Q1-Task1** (this sub-task carries 10% marks):

Given a sequence of integers, e.g.,

9, -1, 45, 6, 8, 21, 34, 5, 55, 65, 543, 18, 90, 122, 132, 0, 66, 100, -12, 17

design an algorithm/method to re-arrange the order (i.e., form a new sequence) of the data items so that when the data items are inserted sequentially into an initially empty BST, the newly created BST will be a balanced BST.

- Design the algorithm (in pseudo code) and analyse your algorithm.
  - Write a Python method to implement your algorithm.
  - Write a program that invokes the above method, thereby demonstrating your algorithm. Your program should
    - print the original data sequence (i.e., the input array).
    - print the re-arranged data items (i.e., a new sequence) generated by the method you completed in step b), and
    - build a BST using the newly generated data sequence as input (that is, insert the data items, one-by-one, into an initially empty BST). Print the tree shape of the BST.
- (Note: The program should use at least two datasets to demonstrate the algorithm, one of which is the dataset given above).

**Q1-Task2**(this sub-task carries 5 marks):

A **complete BST** is a BST that is also a complete binary tree.

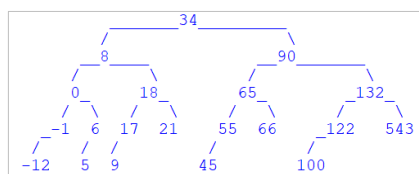
After completing Task 1 above, extend your algorithm to complete the following task/s:

Design an algorithm/method that, for a sequence of integers, re-orders the data items (i.e., forms a new sequence) so that when the data items are inserted in order into an initially empty BST, the newly created BST will be a complete BST. Then re-do the same subtasks b) and c) as outlined in Task 1.

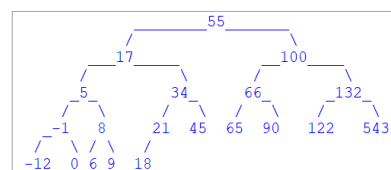
For example, for an integer sequence of

45, -8, 21, 34, 55, 65, 9, 14, 0, 18, 90, 46, 49, 82, 84, 99, 80, 132, 57, 66

your Task 1 may produce a balanced BST as shown in Figure 1 (a) or (b) (note that there can be many other balanced BSTs), but your Task 2 will only produce a tree shape as shown in Figure 1 (b).



(a)



(b)

**Figure 1:** (a) A balanced BST. (b) A BST that is not only a balanced BST but also a complete BST

(Note: Two codes are required, one for Q1-task1 and one for Q1-task2, even if the output of Q1-task1 code is already a complete BST)

## Q2: Application to binary tree traversal & BST (25 marks)

The *in-order* traversal algorithm is one of the three typical binary tree traversal algorithms. It traverses the binary tree in the order of “left-subtree; root node; right-subtree” (see *CSP2348\_M6\_Binary Trees.pptx* in Module 6). If you apply the *in-order* traversal algorithm to a BST (of integer keys), it will print the integer keys in *ascending* order.

An *inverse-in-order* traversal algorithm traverses a binary tree in the order of “right-subtree; root node; left-subtree”. As a comparison, when it is applied to a BST (of integer keys), it prints the integer keys in *descending* order.

Please refer to the Lab code, [TreeTest.py](#), in Module 06. For a given BST, rooted at N, it prints the *Pre-order*, *In-order*, and *Post-order* traversal sequences of the BST. Using the *TreeNode* class and *BST* class defined in the Lab code/s in Module 6,

- Modify the *in-order* traversal algorithm to implement the *inverse-in-order* traversal algorithm. Then, convert the algorithm to a method (e.g., [inverse\\_inorder\(self\)](#) if in Python) and add it into the *BST* class. Use some test data to test the new method.
- Modify the *in-order* traversal algorithm to form two code versions (e.g., [leaf\\_BST\(self\)](#) and [non\\_leaf\\_BST\(self\)](#), if in Python): one prints all leaf nodes, and the other prints all non-leaf nodes of the BST;
- Modify the *pre-order* traversal algorithm to form a new method (e.g., with method head of [total\\_nodesBST\(self, N\)](#), if in Python) such that for a given node *N* in a BST, it counts the total number of nodes of the sub-tree rooted at *N*, and prints all nodes, including *N*, of the sub-tree.
- Modify the *search* tree node algorithm to form a new method (e.g., with method head of [depth\\_nodeBST\(self, N\)](#), if in Python) so that for a given node *N* in a BST, it calculates the depth of the node *N* (in the BST).
- Modify the *post-order* traversal algorithm to form a new method (e.g., with method head of [depth\\_subtreeBST\(self, N\)](#), if in Python) so that for a given node *N* in a BST, it calculates the depth of the sub-tree rooted at *N*.
- Refer to the PPT [CSP2348\\_M6\\_Binary Trees.pptx](#) in Module 6 to design an algorithm for deleting a node from a BST. Then, convert the algorithm to a method (e.g., [delete\\_\(self, key\)](#) if in Python) and add it into the *BST* class. Use some test data to test the new method.

Up on completion of the above steps, write Python code/s to implement a tiny BST application system that allows user to first build a BST and then perform specific operations on the BST.

The system first displays a menu (let’s call the level-1 menu, see below) to build a BST.

- |  |
|--|
| <ol style="list-style-type: none"> <li>1. Pre-load a sequence of integers to build BST</li> <li>2. Manually enter integer keys one by one to build BST</li> <li>3. Exit</li> </ol> |
|--|

Figure 2: Q2 level-1 menu.

This allows the user either to build a BST in one of the two ways or exit the system. For the first two options, the system takes a list of integers (i.e., either via an array or by key-in manually) and inserts them one by one into an initially empty BST (note: do not balance the BST). It then displays a *Menu* (called *leve-2 menu*, see below) that allows the user to browse and execute the following *Menu* options:

1. Display the tree shape of current BST, and then display the *pre-order*, *in-order*, *post-order* and *inverse-in-order* traversal sequences of the BST
2. Display all leaf nodes of the BST, and all non-leaf nodes (separately)
3. Display a sub-tree and count the number of nodes
4. Display the depth of a given node in the BST
5. Display the depth of a subtree of the BST
6. Insert a new integer key into the BST
7. Delete an integer key from the BST
8. Return to the main menu

**Figure 3:** Q2 level-2 menu.

The system continues to loop and prompt user to choose one of the options until the user chooses menu option 8. At this time, the system jumps out the current menu and returns to the main menu.

For menu option 1, you can use the `printTree (...)` method (which is included in one of the Lab codes, e.g., [TreeTest\\_withTreeShape.py](#), or develop your own method) to display the current tree shape of the BST. To display the inverse-in-order traversal order, the system invokes the method/algorithm developed in a).

For menu option 2, the system invokes the algorithm/method developed in b) to display all leaf nodes and non-leaf nodes of the BST respectively.

For menu option 3, prompt the user to enter an integer N (as the key value of the BST). The system then searches the integer N from the BST: If found, the sub-tree rooted at N is displayed and the total number of nodes of the sub-tree it also counted and printed (e.g., by invoking the algorithm/method developed in c)). Otherwise, it displays "ERROR: Node <N> not found!", where <N> is the key value entered.

For menu option 4, the user is prompted to enter an integer N (as the key of a node). The system then searches for node N from the BST: If found, the depth of the node N in the BST is calculated and printed by invoking the relevant algorithm/method developed in d). Otherwise, it prints "ERROR: Node <N> not found!".

For menu option 5, the user is prompted to enter an integer N (as the key of a node). The system then searches for a node N from the BST: If found, the depth of the subtree rooted at N is calculated and printed by invoking the relevant algorithm/ method developed in e). Otherwise, it prints "ERROR: Subtree rooted at node <N> not found!".

For menu option 6, the user is prompted to enter an integer N. The system then searches the BST for the integer N: If found, it prints a message "ERROR: node key <N> already exists in the BST!". Otherwise, it inserts N as a new node of the BST (and it then displays the inverse-in-order traversal sequence of nodes of the BST to verify that the new node has been inserted).

For menu option 7, the user is prompted to enter an integer N. The system then searches the BST for an integer N: If found, the node is deleted from the BST (by calling the algorithm/method developed in f). the system then displays the inverse-in-order traversal sequence of nodes of the BST, thereby verifying that the node has been deleted). Otherwise, it prints "ERROR: Node <N> not found!".

Menu option 8 will exit the current level-2 menu and force the program to return to the level-1 menu.

To test your code, use the following list of integers (as a *Sample dataset*) to build a BST:

58, 84, 68, 23, 38, 82, 26, 17, 24, 106, 95, 48, 88, 54, 50, 51, 53, 49, -6, -46

**(Go to the next page)**

### Q3: AVL tree: deleting a node (20 marks)

Refer to the *AVLTree* class given in the Lab code (see [testAVLTree.py](#)) in Module 07. For a given list of integers, the code inserts the integers one by one into an initially empty AVL tree. It then prints the *in\_order traversal* sequence of the AVL tree, and finally displays the structure of the AVL tree in a specific format (e.g., for each node, it shows its height, balance factor, together with left-subtree, right-subtree, and leaf node indicators, etc. in a particular hierarchical structure).

Read the code and make sure you understand all methods in the *AVLTree* class, including the methods [insert\(...\)](#), [rebalance\(...\)](#), [lrotate\(...\)](#), [rrotate\(...\)](#), and [inorder\\_traverse\(...\)](#), [display\(...\)](#), etc.

#### Your Task:

- Refer to the PPT [CSP2348\\_M7\\_AVL Trees.pptx](#) in Module 7 (and the content covered by Module 6) to design an algorithm for deleting a node from an AVL tree. Then, convert the algorithm into a method (e.g., [delete\\_\(self, key\)](#) if in Python) and add the new method into the *AVLTree* class. Use some test data to test the new method.
- Refer to Q2 and add the *pre-order* and *post-order* traversal methods to the *AVLTree*.

After completing the above steps, write Python code/s to implement a tiny *AVLTree* application system, allowing users to build an AVL tree first and then perform the specified operations on the AVL tree.

To build an AVL tree, the system accepts (as input) a list of integers and inserts them one by one into an initially empty AVL tree. Similar to Q2, you need to provide the user with two options to build an AVL tree, e.g., via a level-1 menu of:

- |  |
|--|
| <ol style="list-style-type: none"> <li>1. Pre-load a sequence of integers to build AVL tree</li> <li>2. Manually enter integer keys one by one to build AVL tree</li> <li>3. Exit</li> </ol> |
|--|

Figure 4: Q3 level-1 menu.

This allows the user to either build an AVL tree in one of the two ways or exit the system. For the first two options, the system takes a list of integers (i.e., either via an array or by manually typed) and inserts them one by one into an initially empty AVL tree. It then displays a *leve-2 menu* (see below) that allows the user to navigate through and execute the following *Menu* options (on the AVL tree built):

- |   |
|---|
| <ol style="list-style-type: none"> <li>1. Display the AVL tree, showing the height and balance factor for each node.</li> <li>2. Print the <i>pre-order</i>, <i>in-order</i>, and <i>post-order</i> traversal sequences of the AVL tree.</li> <li>3. Print all leaf nodes of the AVL tree, and all non-leaf nodes (separately).</li> <li>4. Insert a new integer key into the AVL tree.</li> <li>5. Delete an integer key from the AVL tree.</li> <li>6. Return to the main menu</li> </ol> |
|---|

Figure 5: Q3 level-2 menu.

(Note: you can use the codes/methods in Module 6 & 7, e.g., the display method for menu option 1).

The system continues to loop and prompt the user to choose one of the options until the user chooses menu option 6. At this time, the system jumps out the level-2 menu and returns to the main menu.

The functionalities of level-2 menu options are clear – please refer to the menu options similar to the level-2 menu options in Q2.

To test your code, use the following list of integers (as a *Sample dataset*) to build your initial AVL tree:  
60, 80, -30, 19, 41, 76, 30, 6, 0, -1, 98, 94, 44, 85, 54, 47, 48, 49, 45, 75, 91.

### Some requirements/restrictions on coding:

- This assignment requires the use of the Python programming language to implement the coding tasks. If you want to use a programming language other than Python, you should get a permission from your tutor beforehand, and you may be asked to demonstrate your work at the end.
- Please use one single programming language for all your programming tasks.
- Your codes must be self-supporting: you can use the standard Python library, the codes or snippets from the unit's website, and the codes from our textbook. You can import some packages that are related to algorithm analysis, such as *time*, *random*, and *math*, however you should not import any other packages (such as those formatting related, like *tabulate*, *numpy*, etc.) into your code. - If you do need to include any particular package than the above, please discuss this with your tutor beforehand. If you imported any non-standard package/s into your code that prevents your code from running in our lab environment, you do so at your own risk (of losing marks). In other words, you should make sure your codes work in our lab environment. So, it is important to test your codes in our lab environment before your assignment submission.
- The program should contain a portion of code for building a BST/AVL tree for testing with the sample dataset/s provided. You should also show the effects of using the sample dataset/s in your video demo.

## 2. The written report (16 marks)

Write a project report to cover (but not limited to):

- A brief introduction of the project and requirement/s.
- Algorithm design and implementation procedure (separated by Q1, Q2 and Q3).
- (Optional) User manual: application setup and usage.  
(Note: If your project is implemented in a programming language other than Python, you must include this section to guide your tutor in installing the required programming environment and your project code/s so that they can test your project code/s).
- Test cases to be used in your video demo (including *input/s* and *expected output/s* of each test case). Make sure to include test cases against the sample dataset/s provided in individual questions.
- Snapshots showing application running status (e.g., with inputs and outputs).
- Conclusions or summaries of the completed project.

Please refer to the **Format requirement of the assignment report** in page 7.

Notes: (1) Project source codes are required and must be included separately in the submission zip file.

(2) When forming test cases, please consider the following points:

- How does your code react to correct input?
- How does your code react to incorrect input?
- Can you exit the program safely after testing a chosen task (or sub-task)?
- Do the test cases cover all primary functions of the system?

- Do all menu options function properly?
- Does the program exit as expected?
- Is it formatted well enough?

### Format requirement of the assignment report (2 marks):

Required Content	<p><b>Cover page</b> Must show unit code/name, assignment title, the student IDs and names of all team members, due date, etc.</p>
	<p><b>Executive Summary (optional)</b> This should represent a snapshot of the entire report that your tutor will browse through and should contain the most vital information needed.</p>
Required Content	<p><b>Table of Contents (ToC)</b> This must accurately reflect the content of your report and should be generated automatically in Microsoft Word with clear page numbers.</p>
	<p><b>Introduction</b> Introduce the report, define its scope, and state any assumption if any ; Use in-text citation/reference where appropriate.</p> <p><b>The main body</b> (note: you should use appropriate section title/s) This part should contain (but not limited to):</p> <ul style="list-style-type: none"> <li>• A short description of the programming tasks (e.g., the project scope: what are to be done, and what are excluded from the project, etc.)</li> <li>• Any strategies used to solve the problems (e.g., an approach to develop a solution, if any).</li> <li>• The questions being solved/answered, for example, <ul style="list-style-type: none"> <li>Q1. (should include but not limited to) <ul style="list-style-type: none"> <li>(i) The algorithms and analysis.</li> <li>(ii) The snapshots: Input data sequence. New data sequence generated by the algorithm. The tree shape generated.</li> </ul> </li> <li>Q2: (should include but not limited to) <ul style="list-style-type: none"> <li>(i) (modified) algorithm/s (in pseudo codes) for sub-question a) ~ f).</li> <li>(ii) for the main system, a few screen shots (of running results of the code/s, e.g., one screenshot per menu option).</li> <li>(iii) Test cases that your team members use to demonstrate the project in the video demo, etc.</li> </ul> </li> <li>Q3: (should include but not limited to) <ul style="list-style-type: none"> <li>(i) Algorithm/s (in pseudo codes) for sub-question a).</li> <li>(ii) for the main system, take a few screen shots (of running results of the code/s).</li> <li>(iii) Test cases that your team members use to demonstrate the project in the video demo, etc.</li> </ul> </li> </ul> </li> <li>• For all questions, you may also include: <ul style="list-style-type: none"> <li>(iv) Discussion/critique of the solutions achieved, if any.</li> <li>(v) Any other contents you think necessary to be included.</li> </ul> </li> </ul> <p><b>Conclusion</b> Summary of the works done in this project assignment.</p> <p><b>References</b> A list of end-text reference, if any, formatted according to the ECU requirements using the APA format (Web references are also OK).</p>
Other requirements	<p>The document/report should be no more than 8 pages (excluding references, snapshots, and diagrams). The text must use font <b>Times New Roman</b> and <b>be not smaller than 12pt</b>. Python codes should not be included in body of the written report. They should be saved as separate runnable codes and included in the submission zip file.</p>

### 3. The video demonstration and reflection (20 marks)

This part of the assignment is individual work (not a group work), therefore each team member should produce their own version of video/s. The video should cover two parts of contents: a video demo and a personal video reflection.

(Note: It is fine if you wish to separate the video in two short video clips, one for the project demo and the other video reflection. In this case you should submit two video links) .

#### **(a) The video demonstration**

This (first part of the) video should mainly be a demonstration of the project completed by your group, although you may also present other contents such as explanation of the project report, etc.

The requirements include:

- The duration of video should be between 5 and 10 minutes.
- Student identity verification (SIV): For academic integrity, a SIV is required for the video demonstration. When the video starts, you should take a few seconds to briefly introduce yourself while your computer camera captures your face, or your student ID card.
- The video should cover demonstrations over some test cases:
  - (i) the case/s where a sequence of valid data is entered for BST;
  - (ii) the case/s where a sequence of valid data is entered for AVL tree;
  - (iii) the cases with some invalid data input (e.g., inserting a duplicated integer into a BST/AVL tree, deleting/searching a non-existing key from a binary tree, etc.), thus triggering the system to handle/display error messages, etc.
- Any special features of the project that you want to show.
- You should mention which part of the work (e.g., coding or report) is primarily your contribution to the group project.

#### **(b) The video reflection**

This is the second part of the video. This portion of the video should not exceed 5 minutes in length. In this video section, you should explain your own contribution to the teamwork, providing a personal commentary on the teamwork. The following is a list of points you might consider in your reflection:

- What was your project about?
- How successful was your team?
- What was your key role in completing this project?
- How did completion of this task link to your work in this unit or other units in your study?
- What was your key takeaway from the teamwork/task?
- What did you learn from the process?
- What difficulties you (or your team) encountered during the project development and how did you overcome them?
- Which parts of the tasks were the most challenging, if any?
- Which parts were done well, and which parts of the work could be improved if you are given a chance? etc.

(Note: This video section may also be used to support un-even mark distribution between team members working in the group if the project contributions made by team members differ significantly.)

### 4. The Peer review (optional)

This is to grade performance for each team member, including yourself

This task is *optional* – If you choose to do it, please download the *peer review* form template from Canvas, fill in the form and make necessary remarks/comments, and finally submit it.



## Submission Instructions (2 marks)

- **Project & written report submission** (one submission per team)
  - This submission should include all codes implementing the project (e.g., 3 or more Python source codes, SQL codes, and executable/s if applicable), a written report and any additional documentation associated with the report/project. This submission should be submitted by the team leader only (i.e., through the team leader's Canvas submission portal - the other team member should not submit duplicated assignment version).
    - This submission should be one single compressed file (e.g., in .zip format), containing all your documents (i.e., the written report, source codes, executable files and any other support documents, if any). The written report file must be in Word or PDF format (please refer to the section of *Format requirement of the assignment report*). Please rename the .zip file in a format of

*<Team-leader's Student ID>\_< team-leader's last & first name>\_< the other team member's ID\_ last name>\_CSP2348\_A3.zip.*

If the assignment is done by an individual, please rename the .zip file in a format of

*<Student ID>\_last name>\_< first name>\_CSP2348\_A3.zip.*

*As an example, if the team leader's ID is 12345678, his name is Ben SMITH, the other team member's ID is 15555555 and his last name is MAK, the submission file should be named*  
*12345678\_SMITH Ben\_15555555\_MAK\_CSP2348\_A3.zip.*




*If the assignment is done by Ben SMITH alone, the submission file should be named*  
*12345678\_SMITH Ben\_CSP2348\_A3.zip*

- Note that files found to contain viruses or other infecting mechanisms shall be awarded a ZERO mark (to all team members).
- Your attention is drawn to the university rules governing cheating/plagiarism and referencing. Please refer to the section of *Academic Integrity and Misconduct* in page 9.
- ECU rules/policies will apply for late submission of this assignment.
- **The Video submission** (one per team member)  
You should submit the video link/s only. The video link/s should be properly renamed and contains your own student ID and name.
- **Peer review submission** (optional):  
Fill in the blank form and submit it (note that a form template is provided on Canvas).

Note that separate submission links are available for each submissions.

## Academic Integrity and Misconduct

- This assignment is a **group assignment** (with individual tasks). Your entire assignment must be your own work of the team (unless quoted otherwise) and produced for the current instance of the unit. Any use of uncited content that was not created by your team constitutes *plagiarism* and is regarded as Academic Misconduct. All assignments will be submitted to plagiarism checking software, which compares your submission version with previous copies of the assignment, and the work submitted by all other students in the unit (in the current semester or previous years/semesters).
- *Never give any part of your assignment to someone else or any other team*, even after the due date or the results are released. Do not work on your team assignment with other students or teams. – You can help someone (in other team) by explaining concepts, demonstrating skills, or directing them to the relevant resources. But doing any part of the assignment for them or with them or showing them your work is inappropriate. An unacceptable level of cooperation between students/groups on an assignment is *collusion* and is deemed an act of Academic Misconduct. If you are not sure about plagiarism, collusion or referencing, simply contact your lecturer or unit coordinator.
- You may be asked to explain and demonstrate your understanding of the work you have submitted. Your submission should accurately reflect your understanding and ability to apply the unit content and the skills learnt from this unit.
- Before submitting your assignment 3 (e.g., project and report), team leader and individuals should make sure they have checked all items in the **Academic Integrity tick-before-submit checklist** below and watch the video by clicking the link next to the checklist image:

ACADEMIC INTEGRITY TICK-BEFORE-SUBMIT CHECKLIST	
<b>PLAGIARISM</b> <ul style="list-style-type: none"> <li>✓ I have not copy and pasted from external sources without appropriate citation</li> <li>✓ My in-text and end-text citations follow APA 7 guidelines</li> <li>✓ I have not used my own or other student's previous assignment work</li> </ul>	
<b>COLLUSION</b> <ul style="list-style-type: none"> <li>✓ I have not worked with any other students on this assignment unless permitted</li> <li>✓ My assignment is not based on or derived from the work of any other students</li> <li>✓ I have not shown or provided other student(s) with my assignment at any point</li> </ul>	
<b>CONTRACT CHEATING</b> <ul style="list-style-type: none"> <li>✓ I have not asked or paid someone to do this assignment for me</li> <li>✓ I have not used any content from a "study notes" or "tutoring" service / website</li> <li>✓ I have not had a friend or family member assist me with this assignment</li> </ul>	
<b>IF YOU ARE UNSURE ABOUT ANY OF THE ABOVE, DO <u>NOT</u> SUBMIT YOUR ASSIGNMENT BEFORE SPEAKING WITH YOUR UNIT COORDINATOR OR ECU LEARNING ADVISOR</b>	

[Watch this video before submitting your assignment](#)

## Indicative Marking Guide:

	Description	Allocated Marks	You got
<b>The project (Q1~Q3)</b>	Q1: Balanced BST & complete BST generation Algorithm/s, analysis, and implementation. Python codes run fine and work for the given exemplar inputs. Overall quality of the work.	<b>15</b>	
	Q2: BST based application system development: Algorithm design & analysis for tasks a) ~ f). Implementation of <i>Menu</i> options (of 2 levels). Code runs and functions fine, works for the given exemplar input Overall quality of the work.	<b>25</b>	
	Q3: AVL tree-based application system development: Algorithm design & Analysis for task a) (i.e., deleting a node from an AVL tree) Code/s for task b) Implementation of <i>Menu</i> options (of 2 levels). Code runs and functions (and works for the given exemplar input) Overall quality of the work.	<b>20</b>	
<b>The written report</b>	Executive summary: abstraction of the report & vital information as a whole; Thought/idea organization: conjunctions & cohesion; Clarity: discussion flow and integrity; Citations to References; Test cases; Conclusions achieved, etc. Quality of the report: Report presented as per Format requirement; Report length – not too long and too short (e.g., 1500-3000 words, in 6-10 pages?).	<b>16</b>	
Report format	Document presented as per format requirement	<b>2</b>	
Project/report Submission	All submissions (project code & report) are submitted as per submission requirement	<b>2</b>	
Video demo & reflection	As per requirement	<b>20</b>	
Peer review	(optional)	<b>0</b>	
<b>Penalty</b>	(Possible Plagiarism or Late submission penalty)		
<b>Total</b>	Total mark of 100 which is then converted to 50% of unit mark	<b>100 (/50%)</b>	

## The END of the Assignment Description