# Assignment 10

## Object-Oriented Binary Search Tree ADT

**Date Due: Tuesday, 18 August 2022, 11:59pm** **Total Marks: 50**

**Important Note**

There will be no extensions for this assignment. Hand in what you can by Tuesday midnight.

**General Instructions**

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to com-pare every pair of student submissions.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help. It might help you, but it adds an extra step for the markers.
- Programs must be written in Python 3.
- **Assignments must be submitted to Canvas.** There is a link on the course webpage that shows you how to do this.
- **Canvas will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Read the purpose of each question. Read the Evaluation section of each question.

## Overview

- You must use Git for this assignment. See Question 0. Read it carefully. You will be expected to do all the work on Q1-4, and then produce the git-log for Question 0. Do your best to become comfortable with Git.
- This assignment has 2 major objectives: Binary Search Trees, and a kind of wrap-up of many of the ideas that we covered in this course.
- Question 1 has some debugging to do. But study the readings and the lecture notes. If you don't, you may end up wasting a lot of time.
- Questions 2 and 3 are short and pretty easy, if you understand why we're doing it.
- Question 4 is the wrap-up. You are given a script that reads a file, and you time the results. You compare A10Q2 and A10Q3 when they are set to the same task. You consider the two implementations formally with algorithm analysis, and practically, measuring runtime. You draw conclusion about the value of the two implementations. There is no coding here, unless you decide to modify the given script.
- Question 2 depends on Question 1, pretty strongly, but you can get started with Q2 before you have all the bugs out of Q1. Question 3 does not depend on Q1 or Q2, except for an understanding of the whole assignment.
- Question 4 depends on Q2 and Q3, to a degree, but you can still get some work done if there are bugs that elude you in Q1.

## Question 0 (5 points):

**Purpose:** To force the use of Version Control

**Degree of Difficulty:** Easy

Version Control is a tool that we want you to become comfortable using in the future, so we'll require you to use it in simple ways first, even if those uses don't seem very useful. Do the following steps.

1. Create a new PyCharm project for this assignment
2. Use `Enable Version Control Integration...` to **initialize** Git for your project.
3. Download the Python and text files provided for you with the Assignment, and **add** them to your project.
4. Before you do any coding or start any other questions, make an initial **commit**.
5. As you work on the assignment, use Version Control frequently at various times when you have implemented an initial design, fixed a bug, completed the question, or want to try something different. Make your most professional attempt to use the software appropriately.
6. When you are finished your assignment, open PyCharm's Terminal in your Assignment 4 project folder, and enter the command:

```
    git --no-pager log
```

   **WARNING**: Copy/Paste from PDFs can often add unwanted spaces. There are no spaces between dashes (-) in the above command!

7. Copy/Paste the output of the above command into a text file

**Notes:**

- Several Version Control videos are available on Canvas via the Lecture Videos link.
- No excuses. If you system does not have Git, or if you can't print the log as required, you will not get these marks. Installation of Git on Windows 10 machines is outlined in the "*Step-by-Step Windows 10 - PyCharm UNIX command-line*" video. Git is included in base installations of Linux & MacOS.
- If you are not using PyCharm as your IDE, you are still required to use Git. See the Version Control videos on Canvas that cover using Git on the command line & via GUI.

### What Makes a Good Commit Message?

A good commit message should have a **subject line** that describes *what* changed, a **body** that explains *why* the change was made, and any other *relevant* information. Read "*Writing a Good Commit Message*" linked on Canvas for more guidance.

## Examples of GOOD commit messages:

```
Initial commit for Assignment

* copied starter files over from Canvas
```

```
Completed Question 2

* full functionality complete after testing with generated files
```

```
Fixed a bug causing duplicate print messages

* extra print statements were being produce in loop
* updated logical condition to fix
```

## Examples of BAD commit messages:

```
fixes
```

```
add tests
```

```
updates
```

## What to Hand In

After completing and submitting your work for the Assignment, **follow steps 6 & 7 above** to create a text file named `a10-git.txt`. Be sure to include your name, NSID, student number, course number, and lecture section at the top of your `a10-git.txt` file.

## Evaluation

- 5 marks: The log file shows that you used Git as part of your work for this assignment.

  For full marks, your log file contains
  - Meaningful commit messages.
  - A minimum of 4 separate commits.

## Question 1 (20 points):

**Purpose:** To study the Binary Search Tree operations.

**Degree of Difficulty:** Easy to Moderate.

**Restrictions:** This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

### Task

On Canvas, you will find 2 files:

- `a10q1.py`
- `a10q1_test.py`

The first file is a partial implementation (with a few bugs thrown in) of the Binary Search Tree operations. The second file is a test script to help you complete and fix the implementation of these operations. Currently, of 100 test cases, only 5 test cases are successful.

Your task is to complete and correct the operations, by studying how they should work, and by fixing the given code.

**You are allowed to use the course readings and the lecture notes to complete this question. However, you should not consult any other student, nor any other expert or tutor for assistance in this exercise.**

### What to Hand In

- A file named `a10q1.py` containing the corrected and completed implementation. **Note: If you do not name the script appropriately, you will get zero marks.**

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

### Evaluation

- Your solution to this question must be named `a10q1.py`, or you will receive a zero on this question.
- 20 marks: When our scoring script runs using your implementation of `a10q1.py`, no errors are reported. Partial credit will be allocated as follows:

| Number of tests passed | Marks | Comment |
|---|---|---|
| 0-5 | 0 | the given a10q1 script already passes 5 tests total |
| 5-45 | 5 | |
| 46-69 | 10 | |
| 70-79 | 12 | Studying the lecture notes gets you here at least |
| 80-89 | 15 | |
| 90-98 | 18 | this is actually pretty good |
| 99-100 | 20 | |

For example, if your implementation passes 78 tests, you'll get 12/20. If your implementation passes 85 tests, you'll get 15/20. Our test script is based on the test scripts distributed with the assignment, but may not be exactly the same.

## Question 2 (10 points):

**Purpose:** To implement an Object Oriented ADT based on the Primitive binary search tree operations from Question 1.

**Degree of Difficulty:** Easy to Moderate.

**Restrictions:** This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

### Task

This question assumes you have completed A10Q1. You can get started on this even if a few test cases fail for A10Q1.

On Canvas, you will find 2 files:

- `a10q2.py`
- `a10q2_test.py`

The first file is a partial implementation of the TBase ADT. The second file is a test script to help you complete the implementation of these operations. Currently, of 20 test cases, only 1 test case is successful.

Your task is to complete and correct the operations, by studying how they should work, and by fixing the given code.

**You are allowed to use the course readings and the lecture notes to complete this question. However, you should not consult any other student, nor any other expert or tutor for assistance in this exercise.**

### TBase ADT Description

If you remember how we worked with node chains in Assignment 5, and LLists in Assignment 6, this will feel very familiar. The TBase ADT is simply a primitive binary search tree wrapped in a nice object with an interface that's a bit simpler the the functions from A10Q1. The good news is that because those functions are now all working, all you have to do is get the TBase methods to call those primitive functions properly.

The TBase object stores the root of a binary search tree, in much the same way that the LList record stored the head of a node chain. A more accurate analogy would be to the Stack ADT from Chapter 14, because we only stored the reference to the front of the node chain (the *top*). The TBase object also stores the number of values currently stored, in a private attribute. To help make this clear, the `__init__()` is given, and there is no need to add to it.

The TBase class in `a10q2.py` has five methods outlined with doc-strings, but the methods do not yet do what they should do. Here's what they should do:

- `is_empty(self)` Is the collection empty? Returns a boolean answer.
- `size(self)` How many unique values are stored? Returns an integer answer.
- `member(self, value)` Is the given value in the collection? Returns a boolean answer.
- `add(self, value)` Add the given value to the collection. Returns True if the value was added, and False only if the value is already there.
- `remove(self, value)` Remove the given value from the collection. Returns True if the value was removed, and False only if the value was not there to be removed.

This exercise should feel familiar, especially, `is_empty(self)` and `size(self)`. The remaining three methods should call the corresponding primitive functions from A10Q1 correctly. You do not need to implement them separately.

## What to Hand In

- A file named `a10q2.py` containing the corrected and completed implementation. **Note: If you do not name the script appropriately, you will get zero marks.**

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

## Evaluation

- Your solution to this question must be named `a10q2.py`, or you will receive a zero on this question.
- 10 marks: When our scoring script runs using your implementation of `a10q2.py`, no errors are reported. Partial credit will be allocated as follows:

| Number of tests passed | Marks | Comment |
|---|---|---|
| 0-1 | 0 | the given a10q2 script already passes 1 test total |
| 2-9 | 5 | |
| 10-18 | 8 | |
| 19-20 | 10 | |

For example, if your implementation passes 17 tests, you'll get 8/10. Our test script is based on the test scripts distributed with the assignment, but may not be exactly the same.

## Question 3 (10 points):

**Purpose:** To implement the same behaviour as TBase, but without using A10Q1.

**Degree of Difficulty:** Easy.

**Restrictions:** This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

### Task

This question assumes you have understood what's needed for A10Q2. You can get started on A10Q3 independently from A10Q1 and A10Q2.

You may be wondering if all this work is purely academic, or if there is some value to it. While our TBase objects store simple values, you could easily imagine using this class to store database records, or other application data. We won't build a real application, but we can imagine it.

The theoretical value of the TBase class is that it should be more efficient to use a binary search tree than using a Python list to keep track of values as they are added and removed from a collection. That's the theory. But what about in practice?

In Question 4, we will run an experiment to see if it really is more efficient practically. We'll want to compare how TBase performs to the performance on the same task using a Python list. In Question 4, we will build a single script that can easily be adapted, so that we can use TBase or a Python list. But before we get there, we need to make a Python list look like a TBase object. We called this an adaptor. Our code in Question 4 will require an object with the methods `member(self, value)`, `add(self, value)`, and `remove(self, value)`. We have one of those from Question 2.

In this question, we will define an alternative implementation of the TBase class, which uses Python lists instead of binary search trees. By building this adaptor, we can use the same experiment script (Question 4) for both implementations, A10Q2, and A10Q3.

On Canvas, you will find 2 files:

- `a10q3.py`
- `a10q3_test.py`

The first file is a partial implementation of the TBase ADT, very similar to the one from A10Q2. The second file is a test script to help you complete the implementation of these operations (also very similar to A10Q2). Currently, of 20 test cases, only 1 test case is successful.

**You are allowed to use the course readings and the lecture notes to complete this question. However, you should not consult any other student, nor any other expert or tutor for assistance in this exercise.**

### Alternate TBase ADT Description

This file is almost identical to A10Q2. The methods have the same names, and the same interface (as described by the doc-strings). This time, you should implement the methods assuming that the TBase object stores everything in a Python list.

The TBase object stores the list of values (the stuff). The methods described below simply access this list to perform the needed behaviours. To help make this clear, the `__init__()` is given, and there is no need to add to it.

The TBase class in `a10q3.py` has five methods outlined with doc-strings, but the methods do not yet do what they should do. Here's what they should do:

- `is_empty(self)` Is the collection empty? Returns a boolean answer.
- `size(self)` How many unique values are stored? Returns an integer answer.
- `member(self, value)` Is the given value in the collection? Returns a boolean answer.
- `add(self, value)` Add the given value to the collection. Returns True if the value was added, and False only if the value is already there.
- `remove(self, value)` Remove the given value from the collection. Returns True if the value was removed, and False only if the value was not there to be removed.

You have some decisions to make. Which list methods should you use for these? Your decisions here will have some effect on the results from Question 4. Keep it simple, until you have Question 4 working. Then if you have time, revisit these methods.

## What to Hand In

- A file named `a10q3.py` containing the corrected and completed implementation. **Note: If you do not name the script appropriately, you will get zero marks.**

Be sure to include your name, NSID, student number, course number and laboratory section at the top of all documents.

## Evaluation

- Your solution to this question must be named `a10q3.py`, or you will receive a zero on this question.
- 10 marks: When our scoring script runs using your implementation of `a10q3.py`, no errors are reported. Partial credit will be allocated as follows:

| Number of tests passed | Marks | Comment |
|---|---|---|
| 0-1 | 0 | the given a10q3 script already passes 1 test total |
| 2-9 | 5 | |
| 10-18 | 8 | |
| 19-20 | 10 | |

For example, if your implementation passes 17 tests, you'll get 8/10. Our test script is based on the test scripts distributed with the assignment, but may not be exactly the same.

## Question 4 (5 points):

**Purpose:** To compare the two implementations from Q2 and Q3. To draw many parts of the course together in as single demonstration.

**Degree of Difficulty:** Easy. There is no coding here, but you may have to read through the code to understand what's going on.

**Restrictions:** This question is homework assigned to students and will be graded. This question shall not be distributed to any person except by the instructors of CMPT 145. Solutions will be made available to students registered in CMPT 145 after the due date. There is no educational or pedagogical reason for tutors or experts outside the CMPT 145 instructional team to provide solutions to this question to a student registered in the course. Students who solicit such solutions are committing an act of Academic Misconduct, according to the University of Saskatchewan Policy on Academic Misconduct.

### Task

In Question 2 we implemented the TBase class using binary search trees. In Question 3 we implemented the TBase class using Python lists. They have the same methods, and the same purposes. But which one is more efficient? A formal algorithm analysis tells us that binary search trees should be very efficient for storing and retrieving and deletions by data values. But does the practice match the theory? This question will answer that.

Ideally, we would have a real application requiring the kinds of methods TBase defines, like a student database, or something like that. Instead, you will use a little simulator that only adds, searches, and removes data from the TBase, with no other useful functionality. The simulator will read a file containing a lot of simple commands, and see how long it takes to execute the whole file.

Because it is so close to the end of semester, I have provided such a simulation for you. You will use it, and your answers to A10Q2 and A10Q3, to explore the efficiency of these two implementation of the the same ADT.

### Data

On Canvas, you will find a file `a10q4.zip` containing a bunch of example text files. Each file consists of a number of lines, with each line being a simple command:

```
a 2
a 7
r 2
m 2
m 7
```

Each line consists of 2 tokens, a character (`a`, `r`, or `m`), and an integer. The character indicates the command:

- `a` means to add the integer to the TBase object
- `r` means to remove the integer from the TBase object
- `m` means to check if the integer is in the TBase object

The example above says to add 2 and 7, then remove 2, and finally check if 2 and 7 are in the TBase object. The example files are very long, so that we can get a sense of the efficiency of our class's methods on modern computers.

## Simulation

On Moodle you will find a Python script `a10q4.py` which opens the data files, runs the simulations, and then outputs some data to the console, and displays some plots to the screen.

This experiment script should use the TBase object. Once you have the script working, you are to compare the practical efficiency of the two implementations (A10Q2 and A10Q3) using the same script. Because the two implementations have the same methods, they can be exchanged simply by changing the import line.

## Reflection Questions

- Provide a table of results, showing the run-time for both methods on all example files. You can copy-/paste from the console, into a document.
- Explain the results of the experiment. If one implementation is faster, why? If they are the same, explain how that could happen. Use what you know about the time complexity of Binary Search Tree operations in your explanation.

Note: You answers could change depending on how you implemented A10Q3!

Here's a template for the table you want to produce. The column for $N$ gives the length of the scripts, in terms of the number of commands. The two empty columns should be filled with execution times.

| $N$ | A10Q2 | A10Q3 |
|---:|---|---|
| 1000 | | |
| 2000 | | |
| 3000 | | |
| 4000 | | |
| 5000 | | |
| 6000 | | |
| 7000 | | |
| 8000 | | |
| 10000 | | |
| 12000 | | |
| 14000 | | |

## What to Hand In

- A file named `a10q4_reflections.TXT` (but DOCX, PDF, and RTF files are all acceptable, especially if you want to get fancy with plots) containing your table of results, the answers to the reflection questions.

Be sure to include your name, NSID, student number, and course number at the top of all documents.

## Evaluation

- 7 marks: Your results and explanations.
  - Your Table of results is neat and clear.
  - Your explanation draws together concepts of algorithm analysis, and the study of binary search trees.
  - You made some plausible assumptions about Python list methods' efficiency, or you have used external sources (cite them).