Quintin B. Rozelle
5/21/2025
3-2 Milestone Two
CS-499

- **Briefly describe the artifact. What is it? When was it created?**

The artifact chosen for enhancement one is the binary search tree (BST) assignment from CS-300 Data Structures and Algorithms. This is a quick program that will load a CSV file into memory as a BST and perform basic insertion, search, deletion, and traversal operations on it. It was originally meant as an assignment to gain experience working with BST. I chose to use this one for enhancement one specifically for a few different reasons:

1. I wanted to convert something from one language to another for this artifact. Upon thinking about this, I decided to convert something into Python because it has been a while since I used Python to any significant degree and this enhancement provided me with the opportunity to brush up on my Python skills. Additionally, if I do go into something involving big data, I will need to be proficient in Python.
2. In addition to brushing up on Python, I also wanted to take the opportunity in this class to gain more experience with data structures and algorithms (DSA) since it has been a while since I specifically reviewed/studied those. While enhancement two does cover this, using something related to DSA during another enhancement would give me additional experience.
3. Upon reviewing artifacts that fit the above goals, I realized that there were many areas that I could improve this one specifically. This gives me the chance to showcase what I have learned through this degree since I originally created the original code.

- **Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in software development? How was the artifact improved?**

As stated above, this artifact fits the requirements I was looking for in this enhancement (i.e., conversion to Python, DSA, and opportunities to improve). Further, in planning the additional enhancements, I realized that I could use this for all three which gives me the opportunity to show how I can take code and improve it over time to produce a polished deliverable for a client.

While reviewing the original code, I discovered some significant areas of improvement. These include exception handling, input validation, and better documentation, in addition to converting it to Python. An example of added error handling is in the loadBids function. This loads a file and uses the csv module to parse that file. These options have the possibility to throw errors if something goes wrong. To fix this, the section that could throw errors is wrapped in a try/except block which prints meaningful feedback to the user of what caused the error:

```
try:
    with open(csvPath) as csvFile:
        # detects csv dialect and presence of header
        dialect = csv.Sniffer().sniff(csvFile.read(1024))
        csvFile.seek(0)
        headerPresent = csv.Sniffer().has_header(csvFile.read(1024))
        csvFile.seek(0)
        csvReader = csv.reader(csvFile, dialect)
        # check for presence of header
        if not headerPresent:
            raise FileFormatError("No header found in CSV file")
        else:
            rowNumber = 1
            for row in csvReader:
                if rowNumber == 1:
                    bidIdColumn = row.index('Auction ID')
                    titleColumn = row.index('Auction Title')
                    fundColumn = row.index('Fund')
                    bidAmountColumn = row.index('Winning Bid')
                    rowNumber = rowNumber + 1
                else:
                    bst.insert(Bid(int(row[bidIdColumn]),
                                   row[titleColumn],
                                   row[fundColumn],
                                   float((row[bidAmountColumn][1:]).replace(',',''))))
except Exception as error:
    print("Error loading file")
    print(f"Error type: {type(error)}")
    print(f"Error message: {error}")
finally:
    return bst
```

An example of input validation can be seen in the displayMainMenu function which displays the main menu and accepts input which is returned to the main function for further processing. This function will only return the entered input if it is found in a list of acceptable choices, otherwise it tells the user the input was invalid and loops back to the start of the function:

```
def displayMainMenu() -> int:
    choice : str = "0"
    while choice == "0":
        print("Menu:")
        print("  1. Load Bids")
        print("  2. Display All Bids")
        print("  3. Find Bid")
        print("  4. Remove Bid")
        print("  9. Exit")
        choice = input("Enter choice: ")

        if choice in ["1", "2", "3", "4", "9"]:
            return int(choice)
        else:
            print("Invalid choice. Please try again")
            choice = "0"
```

Lastly, examples of the improved documentation can be seen throughout the code in the inclusion of docstrings to document what a function does to make using this code easier for another developer:

```
def search(self, key: Any) -> Any:
    """
    Searches for a new key in the BST

    Parameters
    ----------
    key : Any
        The key to be searched for in the BST

    Returns
    -------
    Any
        The full key if found, otherwise None.
        Allows for returning full key if keys are complex objects
        and search was performed with dummy key containing only
        the attributed used for comparison.
    """
```

While making these improvements, I realized additional areas that could be improved which I worked into the final enhancement. First, the original code was all in one file which was

good for the original intent, but splitting portions into separate modules adds further improvements as it better encapsulates related code, makes it easier to read and maintain, and provides for future reuse. On this last point specifically, I realized that since I will be adding a red-black tree to the second enhancement, if I split the BST and node classes into their own module, I can add the red-black tree to that and use the code for loading a file and displaying the menu almost unchanged. Next, I incorporated the use of unit testing. I created separate files which test the functionality of the main modules. This greatly speeds up the testing process.

- **Did you meet the course outcomes you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?**

I planned to target outcomes 1, 4, and 5 with this enhancement and have met all three with the enhancements I made to this artifact. For outcome 1, I have included well written comments and docstrings throughout the code to convey to another developer what functions are for. For outcome 4, I have successfully implemented and used modules that others have created to create my solution. Lastly, for outcome 5, I have added increased security and reliability through data validation and exception handling. All this being said, I won't officially count outcomes 1 and 4 completed yet as I do have them down for the other enhancements in this course.

At this point, I do not have any updates to make to the outcome-coverage plan.

- **Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?**

In addition to the stated learning goals above (i.e., more practice with Python and DSA), I learned a few additional things. First, I chose to better use unit testing while creating this artifact which historically is something I haven't done much. While it did take time to learn unit testing

in Python and develop those unit tests prior to coding the enhancement, it did produce better results and helped to guide my development. Next, I learned that Python has a limit to the number of recursive functions that can be added to the stack which appears to be around 1000. I originally made my BST functions use recursion, but found that when loading a 12,000-line, presorted CSV file, the code would fail due to this limit. Converting the functions to iterative versions solved this issue. Lastly, I learned that the format of a source file can have drastic changes to the functioning code. Once I fixed the issue with recursion, I decided to test my code on the same CSV file but when it was randomly sorted. The presorted file took my program about 13.5 seconds to load while the randomly sorted file took about 0.1 seconds to load. I believe this is because the presorted build an incredibly tall BST (all nodes essentially having only right children) while the randomly sorted file created a more well-balanced BST. This cut down on the iterations during the insertions which created a faster tree.