**Project Report – B00268411**

This project was completed by Chloe Madden, Adam Stanton and myself, Paul McLaughlin.

Our aim was to create two binary search trees, one of which was an array implementation, and the other was just a regular implementation, we worked off a lab example to achieve this.

A few advantages and disadvantages of our approach, are, with the array implementation, using an index, makes the nodes easily accessible and it meant you could search through the whole tree. However, in order to access a given entry, there would always have to be a traversal of the tree. We found some issues in reading writing the code as an array, because there is an extra part where “next” node goes inside the array brackets.

**Critical Appraisal**

Overall, I feel that the project went well, with a few minor setbacks we were able to overcome.  These include: implementing the iterator, which we struggled with due to the fact that we were unsure how iterators are implemented at the start, removing from the tree, we were able to rotate the entries, however, they were not actually being removed from the array, and this also caused some issues with printing.

The rest of the project seems to work well, and prints everything the way we wanted it to, something I like about the project is how we choose between the two implementations, it looks tidy with the abstractTree class.

Binary Tree Array Test Cases:

Adam (B00266256), Chloe (B00286864), Paul (B00268411)

# Test Inputs:

* Add(10)
* Add(5)
* Add(2)
* Add(7)
* Add(13)
* Add(65)
* Add(14)
* Add(9)
* Contains(10) – Should Return True
* Contains(65) – Should Return True
* Contains(99) – Should Return False
* CountEntries() – Should Return 8
* CountLeaves() – Should Return 3
* PrintTree() – Will Provide Slightly Different Formatting.
* PrintInOrder() – Should Provide Matching Results.

# Test Outputs:

|  |  |
| --- | --- |
| **BinarySearchTreeArray** | **BinarySearchTree** |
| run:  Welcome To The BinarySearchTreeArray Java Application by B00286864, B00266256 & B00268411.  Please decide between using an array-based implementation or a regular binary search tree implementation.  0: quit  1: use an array-based Binary Search Tree  2: use a regular Binary Search Tree  option: 1  Using The Array Implementation...  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 10  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 5  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 2  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 7  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 13  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 65  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 14  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 9  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 2  Checking For A Specific Value...  element to be checked: 10  Tree Contains Element: true  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 2  Checking For A Specific Value...  element to be checked: 65  Tree Contains Element: true  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 2  Checking For A Specific Value...  element to be checked: 99  Tree Contains Element: false  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 3  Printing The Size Of The Tree...  No. Of Entries: 8  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 4  Printing The Number Of Leaves...  No. Of Leaves: 3  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 6  Printing The Tree...  0) Element: 10 Parent: -1 Left: 1 Right: 4  1) Element: 5 Parent: 0 Left: 2 Right: 3  2) Element: 2 Parent: 1 Left: -1 Right: -1  3) Element: 7 Parent: 1 Left: -1 Right: 7  4) Element: 13 Parent: 0 Left: -1 Right: 5  5) Element: 65 Parent: 4 Left: 6 Right: -1  6) Element: 14 Parent: 5 Left: -1 Right: -1  7) Element: 9 Parent: 3 Left: -1 Right: -1  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 7  Printing The Tree In Order Of Insertion...  2, 5, 7, 9, 10, 13, 14, 65, | run:  Welcome To The BinarySearchTreeArray Java Application by B00286864, B00266256 & B00268411.  Please decide between using an array-based implementation or a regular binary search tree implementation.  0: quit  1: use an array-based Binary Search Tree  2: use a regular Binary Search Tree  option: 2  Using The Regular Implementation...  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 10  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 5  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 2  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 7  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 13  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 65  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 14  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 1  Adding To The Tree...  New Value: 9  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 2  Checking For A Specific Value...  element to be checked: 10  Tree Contains Element: true  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 2  Checking For A Specific Value...  element to be checked: 65  Tree Contains Element: true  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 2  Checking For A Specific Value...  element to be checked: 99  Tree Contains Element: false  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 3  Printing The Size Of The Tree...  No. Of Entries: 8  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 4  Printing The Number Of Leaves...  No. Of Leaves: 3  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 6  Printing The Tree...  Element: 2 Parent: 5 Left: null Right: null  Element: 5 Parent: 10 Left: 2 Right: 7  Element: 7 Parent: 5 Left: null Right: 9  Element: 9 Parent: 7 Left: null Right: null  Element: 10 Parent: null Left: 5 Right: 13  Element: 13 Parent: 10 Left: null Right: 65  Element: 14 Parent: 65 Left: null Right: null  Element: 65 Parent: 13 Left: 14 Right: null  0: return  1: add  2: contains  3: count entries  4: count leaves  5: remove  6: display  7: display in order  option: 7  Printing The Tree In Order Of Insertion...  2, 5, 7, 9, 10, 13, 14, 65, |

# Afterword:

All test cases provide the expected/correct outputs, and both the BinarySearchTreeArray & BinarySearchTree implementations return matching results (with the exception of printTree() where the array implementation is preceeded by its indexes).

The remove() method returns an error, so it has not been included in these test cases.