CS562
Kazuki Kaneoka
Student ID: 932-277-488

## Part 1: Proposal For What To Test

### -What To Test

I would like to choose "bintrees 2.0.4" as my SUT. The library provides Binary tree, AVL tree, and Red-Black tree implemented by Python and Cython/C. This library can be found at https://pypi.python.org/pypi/bintrees/2.0.4.
Trees implemented by Python are:

- BinaryTree, which is unbalanced binary tree
- AVLTree, which is balanced AVL tree
- RBTree, which is balanced Red-Black tree

Trees implemented by Cython/C are:

- FastBinaryTree, which is unbalanced binary tree
- FastAVLTree, which is balanced AVL tree
- FastRBTree, which is balanced Red-Black tree

All trees provide the same interface like following functions:

- insert(self, key, value)
- remove(self, key)
- get(self, key, default=None)
- pop_min(self)
- pop_max(self)
- intersection(self, *trees)
- union(self, *trees)
- difference(self, *trees)
- symmetric_difference(self, tree)

### -How To Test

I'll test the correctness of the interfaces provided by the library for each tree. Also, I will work on the performance test among each tree using random numbers to investigate which tree is fast for which input data.

### -Briefly Descriptions Of Tree Data Structure

I will start describing the concept of tree, which is one of fundamental data structures in computer science. Then, I will focus on binary search tree. It is a special type of tree. Finally, I'll talk about AVL tree and Red-Black tree. They are the most useful binary search trees in real world.

### -Tree

A tree is a data structure that represents a hierarchical structure with node. In tree, data is represented by node. Each node contains its value and a list of references to other nodes. We call node A is parent of node B if node A has references to node B and node B is a child of node A if node B is referenced by node A. There is a special node called, root. Every tree has exactly one root and it is stored at the top of its hierarchy. From root, we can access all nodes in tree by using references. There is one important constraint for tree, which child node has no reference to parent node. It means the direction of reference is one way starting from root. This is the reason why we say tree as a graph with no cycle.

**-Binary Search Tree**
Binary search tree is a tree such that each node has exactly 2 children except leaf node, which is a node at the bottom of its hierarchy and the value of each node is greater than its left children and smaller than its right children. So, in binary search tree, node with largest value is always at the right most in the structure and node with smallest value is always at the left most in the structure. There are 3 basic operations for binary search tree: insert, delete, and search. The time complexities of those 3 operations always depend on height of tree, which is the maximum number from root to leaf. So, the tree with smaller height produces better complexity. Theoretically, the height of tree can be O(n) in worst case and can be O(n/2) in best case where n is the number of nodes in tree. Also, we say binary search tree is height balanced if the difference between height of left children and height of right children is no more than 1. If binary search tree is height balanced, we know the height of tree is small, and then, we know that insert, delete, and search operations is very fast, O(logn). We would like to have the height-balanced tree.

**-AVL Tree**
AVL Tree is one of the binary search trees that guarantee its height balance. It keeps the height difference between left and right in every nodes is not greater or smaller than 1.

**-Red-Black Tree**
Red-Black tree is another binary search tree that provides practically height-balanced tree by keeping the following properties:

- Every node is either red or black node.
- All leaf is black. If it has no child, we assume it has 2 children with black nodes.
- 2 children of red node are black nodes.
- Among nodes in same level, the number of black nodes from a node to leaf is always same.