

GTU Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework 7 – Part 3

205008003010

Murat SARIBAŞ

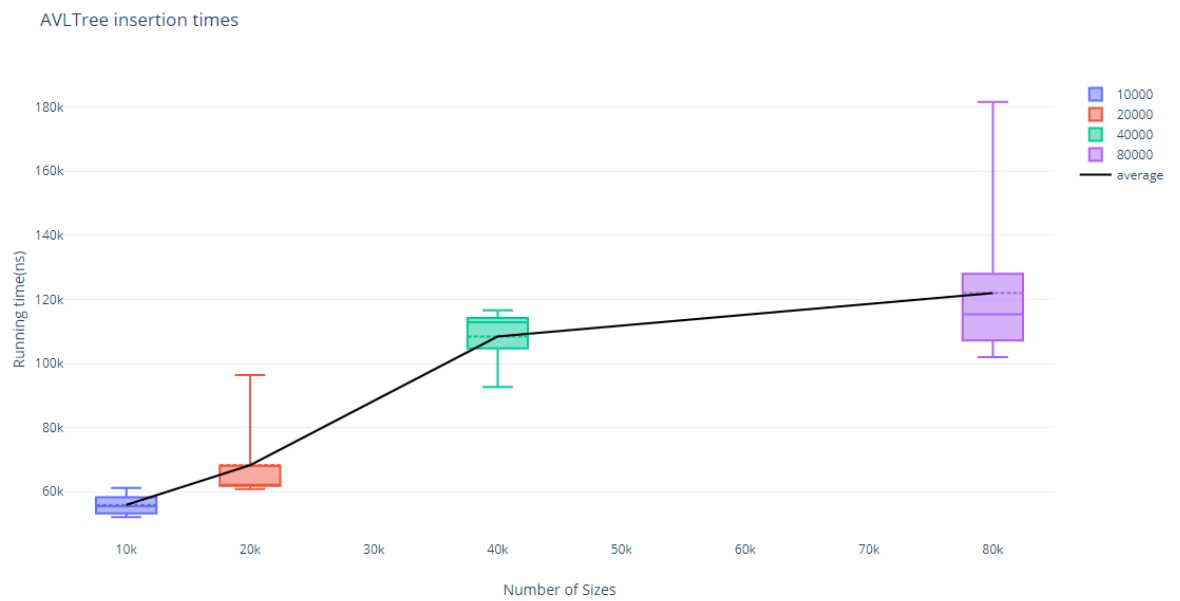
```

classDiagram
    class BinaryTree {
        +Node<E> root
        +BinaryTree()
        +BinaryTree(Node<E>)
        +BinaryTree(E, BinaryTree<E>, BinaryTree<E>)
        +toString() String
        +preOrderTraverse(Node<E>, int, StringBuilder) void
        +leaf boolean
        +data E
        +rightSubtree BinaryTree<E>
        +leftSubtree BinaryTree<E>
    }
    class BinarySearchTree {
        +addReturn boolean
        +deleteReturn E
        +add(E) boolean
        +add(Node<E>, E) Node<E>
        +contains(E) boolean
        +find(E) E
        +find(Node<E>, E) E
        +delete(E) E
        +delete(Node<E>, E) Node<E>
        +findLargestChild(Node<E>) E
        +remove(E) boolean
    }
    class AVLTree {
        +increase boolean
        +decrease boolean
        +add(E) boolean
        +add(AVLNode<E>, E) AVLNode<E>
        +delete(E) E
        +delete(AVLNode<E>, E) AVLNode<E>
        +findReplacementNode(AVLNode<E>) AVLNode<E>
        +findLargestChild(AVLNode<E>) E
        +incrementBalance(AVLNode<E>) void
        +rebalanceRight(AVLNode<E>) AVLNode<E>
        +rebalanceLeft(AVLNode<E>) AVLNode<E>
        +rebalanceRightL(AVLNode<E>) AVLNode<E>
        +rebalanceLeftL(AVLNode<E>) AVLNode<E>
        +decrementBalance(AVLNode<E>) void
    }
    class RedBlackTree {
        +fixupRequired boolean
        +add(E) boolean
        +add(RedBlackNode<E>, E) Node<E>
        +moveBlackDown(RedBlackNode<E>) void
        +delete(E) E
        +removeFromLeft(Node<E>, E) E
        +removeFromRight(Node<E>, E) E
        +findReplacement(Node<E>) Node<E>
        +findLargestChild(Node<E>) E
        +fixupRight(Node<E>) Node<E>
        +fixupLeft(Node<E>) Node<E>
    }
    class BTree {
        +root Node<E>
        +order int
        +newParent E
        +newChild Node<E>
        +BTree(int)
        +contains(E) boolean
        +find(E) E
        +find(Node<E>, E) E
        +add(E) boolean
        +insert(Node<E>, E) boolean
        +binarySearch(E[], E, int, int) int
        +insertIntoNode(Node<E>, int, E, Node<E>) void
        +splitNode(Node<E>, int, E, Node<E>) void
        +remove(E) boolean
        +delete(E) E
        +toString() String
        +preOrderTraverse(Node<E>, int, StringBuilder) void
    }
    class SearchTree {
        +add(E) boolean
        +contains(E) boolean
        +find(E) E
        +delete(E) E
        +remove(E) boolean
    }
    class SkipList {
        +head SLNode<E>
        +size int
        +maxLevel int
        +maxCap int
        +LOG2 double
        +MIN int
        +rand Random
        +SkipList()
        +size() int
        +search(E) SLNode<E>[]
        +find(E) E
        +add(E) boolean
        +remove(E) boolean
        +logRandom() int
        +computeMaxCap(int) int
        +toString() String
    }
    class SLNode {
        +links SLNode<E>[]
        +data E
        +SLNode(int, E)
        +toString() String
    }
    class Node {
        +size int
        +data E[]
        +child Node<E>[]
        +Node(int)
        +toString() String
    }
    class BinarySearchTreeNode {
        +rotateRight(Node<E>) Node<E>
        +rotateLeft(Node<E>) Node<E>
    }
    class RedBlackNode {
        +isRed boolean
        +RedBlackNode(E)
        +toString() String
    }
    class AVLNode {
        +LEFT_HEAVY int
        +BALANCED int
        +RIGHT_HEAVY int
        +balance int
        +AVLNode(E)
        +toString() String
    }
    class main {
        +main(String[]) void
    }

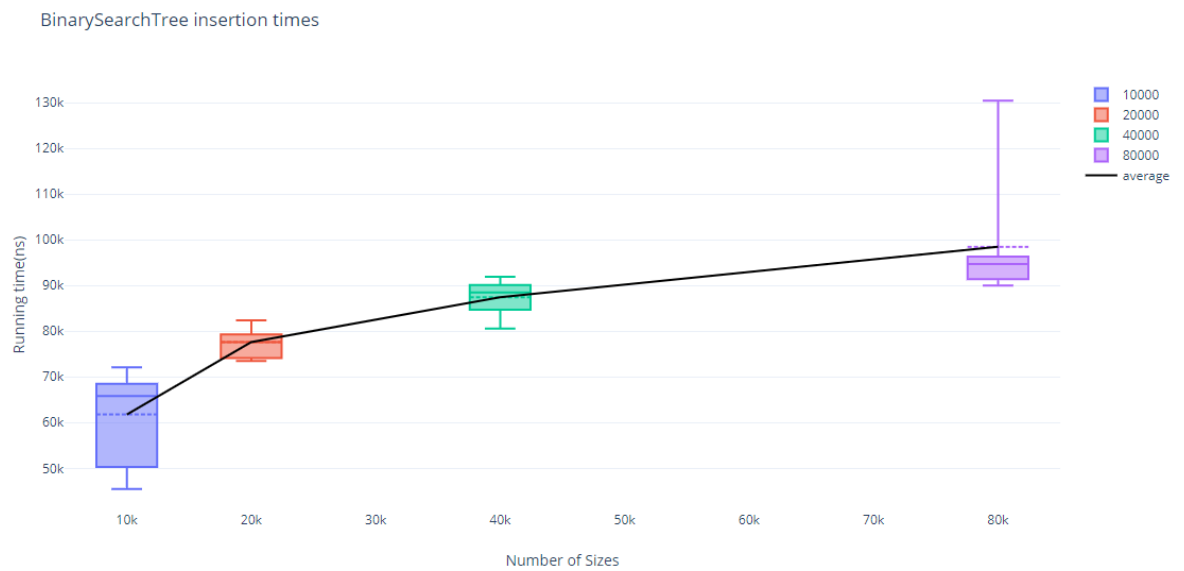
    BinaryTree --> BinarySearchTree
    BinaryTree --> AVLTree
    BinaryTree --> RedBlackTree
    BinaryTree --> BTree
    BinaryTree --> SearchTree
    BinaryTree --> SkipList
    BinaryTree --> Node
    BinarySearchTree --> BinarySearchTreeNode
    AVLTree --> AVLNode
    RedBlackTree --> RedBlackNode
    BTree --> Node
    SearchTree --> Node
    SkipList --> SLNode
    BinarySearchTreeNode --> BinarySearchTreeNode
    AVLNode --> AVLNode
    RedBlackNode --> RedBlackNode
    BinarySearchTreeNode --> AVLNode
    BinarySearchTreeNode --> RedBlackNode
    BinarySearchTreeNode --> Node
    BinarySearchTreeNode --> AVLTree
    BinarySearchTreeNode --> RedBlackTree
    BinarySearchTreeNode --> BTree
    BinarySearchTreeNode --> SearchTree
    BinarySearchTreeNode --> SkipList
    BinarySearchTreeNode --> Node
    BinarySearchTreeNode --> main
  
```

-Running and Results

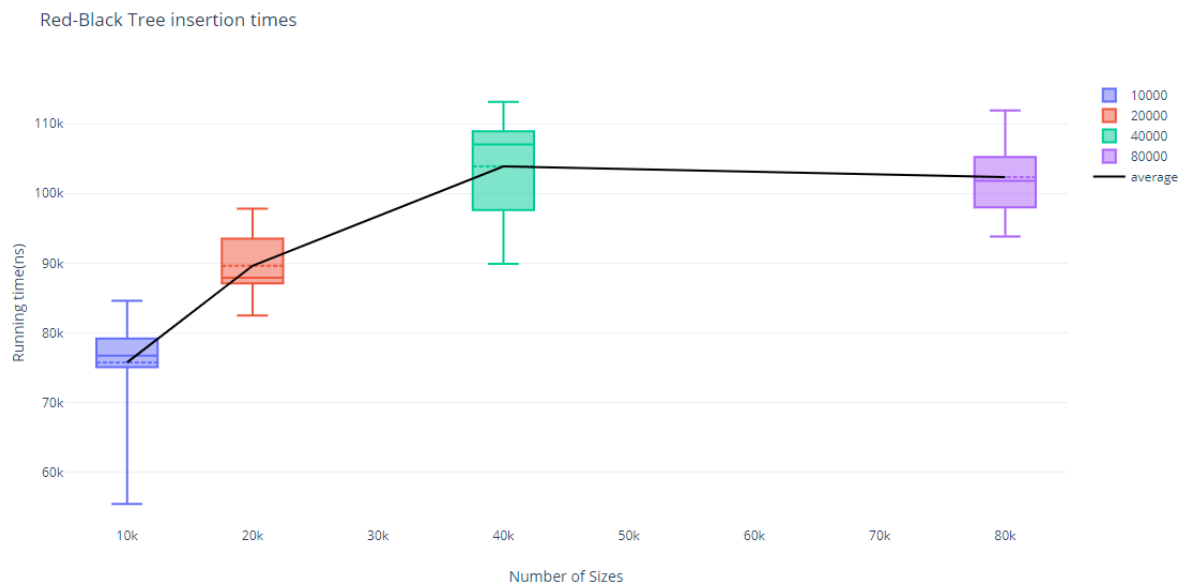
- **AVLTree**



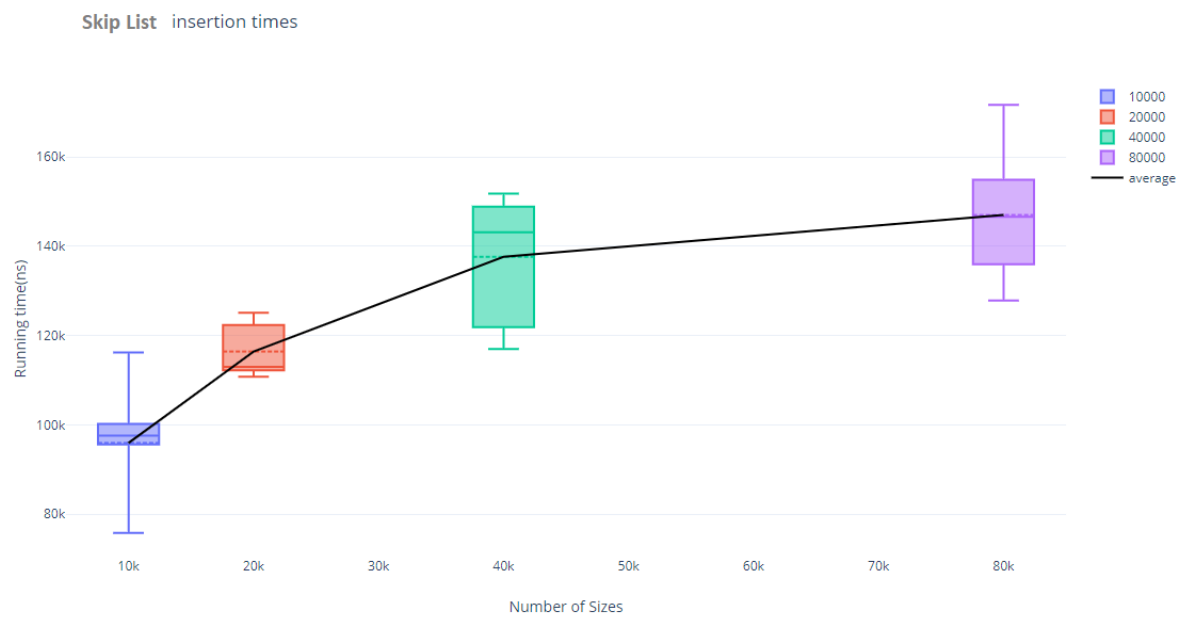
- **BST**



• Red-Black Tree



• Skip List



Average Running Times Comparison

