Fall 2021 – CS 303 Algorithms and Data Structures
Lab 8
Homework Deadline:

Notes:
- Implement the algorithm and analyze the results using the give input files
- Deliverables: Report.pdf file and your code file (please do not send a zip file. If you have more than one class in your code, then submit each file separately through Canvas.)
- Homework report must follow the guidelines provided in the sample report uploaded in Canvas

Objectives:
- Insert elements in a Binary Search Tree
- In-order traversal of the BST to print elements in a sorted order
- Search an element in a Binary Search Tree
- Use the Binary Search Tree in a real application

Problems
1. Implement a method Insert (T data) to insert a node having value of key = data in a Binary Search Tree.

2. Implement In-order Traversal () that prints the elements of the BST in an in-order format. If your insertion of elements in the BST is correct, then the in-order traversal output should print the elements in a sorted order.

3. Write a driver program to test the Binary Search Tree program. Make sure you use all the methods implemented above.

4. Implement Search (T data) to search for a node having value of key = data in a Binary Search Tree.

5. Use the Binary Search Tree to build a search tree using the given input file that consists of two fields: a **UPC key** and the corresponding description. Use the search tree created to find the description associated with a given set of UPC keys. The input file ***`UPC.csv`*** provides the key and corresponding descriptions in a comma separated file and the various search keys are provided in the file ***`input.dat`***. First test the program by entering couple of keys manually and print the description. Once you are convinced the program is working correctly, test the program for the given search keys and determine the total time taken to complete the search.

```
TREE-INSERT($T, z$)
    $y$ = NIL
    $x = T.root$
    while $x \neq$ NIL
        $y = x$
        if $z.key < x.key$
            $x = x.left$
        else $x = x.right$
    $z.p = y$
    if $y$ == NIL
        $T.root = z$        // tree $T$ was empty
    elseif $z.key < y.key$
        $y.left = z$
    else $y.right = z$
```

```
INORDER-TREE-WALK($x$)
    if $x \neq$ NIL
        INORDER-TREE-WALK($x.left$)
        print $x.key$
        INORDER-TREE-WALK($x.right$)
```

```
TREE-SEARCH($x, k$)
    if $x$ == NIL or $k$ == $x.key$
        return $x$
    if $k < x.key$
        return TREE-SEARCH($x.left, k$)
    else return TREE-SEARCH($x.right, k$)
```

OR

```
ITERATIVE-TREE-SEARCH( $x, k$)
    while $x \neq$ NIL and $k \neq x.key$
        if $k < x.key$
            $x = x.left$
        else
            $x = x.right$
    return $x$
```