Department of Computer Science and Engineering
National Institute of Technology Calicut
CS2094D DATA STRUCTURES LABORATORY
(Second Semester MCA)

ASSIGNMENT 3

General Instructions

1. **Try to complete all the assignment questions and submit in eduserver platform**
2. **Your assignment is due by 11.30pm on 28/04/2022**
3. **All programs should be implemented in C language**
4. **You have to upload the complete code as a zip file named as Firstname-Regno-Assg#(e.g., John-m200034ca-Assg1. Zip)**
5. **Students who submit works that are found to have been plagiarized can be cause for an automatic 0 marks**
6. **All submitted code should be well-documented, and the code should be easily understood by reading the comments**

1. Write a program to create an AVL Tree A and perform the operations insertion, deletion, search and traversal. Assume that the AVL Tree A does not contain duplicate values. Your program should contain the following functions

- *Insert (A, k)* – Inserts a new node with key 'k' into the tree A
- *Search(A, k)* - Searches for a node with key k in A, and returns a pointer to the node with key k if one exists; otherwise, it returns NIL.
- *DeleteNode(A,k)* – Deletes a node with the key 'k' from the tree A.
  Note: The caller of this function is assumed to invoke Search() function to locate the node x.
- *GetBalance(A,k)* – Prints the balance factor of the node with k as key in the tree A
  Note:- Balance factor is an integer that is calculated for each node as:
  B_factor =height(left subtree)−height(right subtree)
- *LeftRotate(A,k)* – Perform left rotation in tree A, with respect to node k.
- *RightRotate(A,k)* – Perform right rotation in tree A, with respect to node k
- *IsAVL(A)* – Checks whether the tree pointed by A is an AVL tree or not.
- *PrintTree(A)* – Prints the tree given by A in the parenthesis format as: (t (left-subtree)(right-subtree)). Empty parentheses ( ) represent a null tree.
  Note: After each insertion on an AVL Tree, it may result in increasing the height of the

tree. Similarly, after each deletion on an AVL Tree, it may result in decreasing the height of the tree. To maintain the height-balanced property of the AVL tree, we need to implement rotation functions

Input Format:

- Each line contains a character from ‘i’, ‘d’, ‘s’, ‘b’, ‘c’, ‘p’ and ‘e’ followed by at most one integer.
- Character ‘i’ is followed by an integer separated by space; a node with this integer as key is created and inserted into A.
- Character ‘d’ is followed by an integer separated by space; the node with this integer as the key is deleted from A and the deleted node’s key is printed.
- Character ‘s’ is followed by an integer separated by space; find the node with this integer as key in A.
- Character ‘b’ is followed by an integer separated by space; find the balance factor of the node with this integer as key in A and print the balance factor.
- Character ‘c’ is to check whether tree A is AVL Tree or not.
- Character ‘p’ is to print the Parenthesis Representation of tree A.
- Character ‘e’ is to ‘exit’ from the program.

Output Format:

- The output (if any) of each command should be printed on a separate line.
- For option ‘d’, print the deleted node’s key. If a node with the input key is not present in A, then print FALSE.
- For option ‘s’, if the key is present in A, then print TRUE. If the key is not present in A, then print FALSE.
- For option ‘b’, if the key k is present in A, then print the balance factor of the node with k as the key. If the key is not present in A, then print FALSE.
- For option ‘c’, if tree A is an AVL Tree, then print TRUE. If tree A is not an AVL Tree, then print FALSE.
- For option ‘p’, print the space-separated Parenthesis Representation of the tree A

Sample Input:

```
i 4
i 6
i 3
i 2
i 1
s 2
```

p
b 4
d 3
p

Sample Output:

TRUE
(4 (2 (1 ( ) ( )) (3 ( ) ( ))) (6 ( ) ( )))
1
3
(4 (2 (1 ( ) ( )) ( )) (6 ( ) ( )))

2. Write a program to implement a Binomial Heap and perform the operations insertion, deletion, finding the minimum key and union.

Your program should contain the following functions:

- *MakeHeap()* - Creates and returns a new heap *H* containing no elements.
- *Insert(H, x)* – Inserts a new node with key *'x'* into the heap *H*.
- *Minimum(H)* – Returns the value of the smallest key in the heap *H*.
- *DecreaseKey(H, x, k)* – Decreases the value of the node with key *x* by *k,* if *x>=k*, and then returns the updated key value. Otherwise, the function returns -1.
- *Delete(H, x)* - Deletes the node with key *'x'* from the heap H. If node is present, it prints the deleted node else it prints -1.
- *Union(H1, H2)* - Create and return a new heap *H* that contains all the nodes of heaps *H1* and *H2*. Note that heaps *H1* and *H2* are "destroyed" by this operation.
- *Print(H)* – Prints the keys in *H* using level order traversal.

Input Format:

- Each line contains a character from *'i', 'd', 'p', 'm', 'r'* and *'e'* followed by at most two integers.
- *i k* - inserts element *k* into the heap
- *d k* - deletes the node with key *k* from the heap and prints the deleted node's key. Print '-1' if the node with key *k* is not found.
- *p* - prints the binomial heap in level order traversal
- *m* - prints the minimum element in the binomial heap
- *r y z* - decreases the value of the node with key *y* by *z,* and prints the updated key of the node. Print '-1' if the node with key *y* is not found.
- *e* - 'exit' from the program.

Output Format:

- The output (if any) of each command should be printed on a separate line.

Sample Input:

i 10

i 20

i 30

i 40

i 50

p

m

d 10

p

r 50 3

p

r 100 10

e

Sample Output:

50 10 30 20 40

10

10

20 30 50 40

47

20 30 47 40

-1

3. Given an array of *n* integers with any of these integers appearing any number of times. Write a program to sort these *n* integers in *O(nlog m)* time, where *m* is the number of distinct elements in array.

Hint: Use AVL tree. The idea is to extend the tree node to have a count of keys also. Each node in the tree should be of the following type.

```
Struct node {
        int key;
        int count;    //number of times a key appears in the array
        int height;
        struct node *left;
        struct node *right;
}
```

Input Format:

- The first line of the input contains an integer $n \in [1, 100]$, the number of elements in the array.
- Second line contains *n* integers (each separated by a space) corresponding to the elements of the array.

Output Format:

The *n* elements (each separated by a space) of the given array in **non-decreasing** order in a single line.

Sample Input:

12
100 12 100 1 1 12 100 1 12 100 1 1

Sample Output:

1 1 1 1 1 12 12 12 100 100 100 100