**LAB # 09**

**Task # 01: Implement bucket sort using linked list.**

**Solution:**

namespace BucketSort

{

public class Node

{

public int data;

public Node next;

public Node(int d)

{

data = d;

next = null;

}

}

public class LinkedList

{

public Node head;

public void printList()

{

Node n = head;

while (n != null)

{

Console.Write(n.data + " ");

n = n.next;

}

}

public void push(int naya\_data)

{

Node nai\_node = new Node(naya\_data);

nai\_node.next = head;

head = nai\_node;

}

public void append(int naya\_data)

{

Node nai\_node = new Node(naya\_data);

if (head == null)

{

head = nai\_node;

return;

}

nai\_node.next = null;

Node last = head;

while (last.next != null)

{

last = last.next;

}

last.next = nai\_node;

return;

}

public void insertAfter(Node prev\_node, int naya\_data)

{

if (prev\_node == null)

{

Console.WriteLine("Given previous node cannot be null");

}

Node new\_node = new Node(naya\_data);

new\_node.next = prev\_node.next;

prev\_node.next = new\_node;

}

public void Delete(int data)

{

Node temp = head, prev = null;

if (temp != null && temp.data == data)

{

head = temp.next;

}

while (temp != null && temp.data != data)

{

prev = temp;

temp = temp.next;

}

if (temp == null)

{

return;

}

prev.next = temp.next;

}

public void BucketSort()

{

Node i = head;

while (i.next != null)

{

Node small = i;

Node j = i.next;

while (j.next != null)

{

if (i.data > j.data)

{

small = j;

}

j = j.next;

}

int t = i.data;

i.data = small.data;

small.data = i.data;

i = i.next;

}

}

}

class Program

{

static void Main(string[] args)

{

LinkedList buk1 = new LinkedList();

LinkedList buk2 = new LinkedList();

LinkedList buk3 = new LinkedList();

Console.WriteLine("Enter no of values you want to enter:");

int n = int.Parse(Console.ReadLine());

int[] arr = new int[n];

for(int i = 0; i < n; i++)

{

Console.WriteLine("Enter values");

arr[i] = int.Parse(Console.ReadLine());

}

Console.WriteLine("Before Sorting");

for(int i = 0; i < n; i++)

{

Console.Write(arr[i]+" ");

}

Console.WriteLine();

for(int i = 0; i < n; i++)

{

if (arr[i] >= 0 && arr[i] < 5)

{

buk1.append(arr[i]);

buk1.BucketSort();

}

else if (arr[i] >= 5 && arr[i] < 10)

{

buk2.append(arr[i]);

buk2.BucketSort();

}

else if (arr[i] > 10)

{

buk3.append(arr[i]);

buk3.BucketSort();

}

}

Console.WriteLine("After sorting");

buk1.printList();

buk2.printList();

buk3.printList();

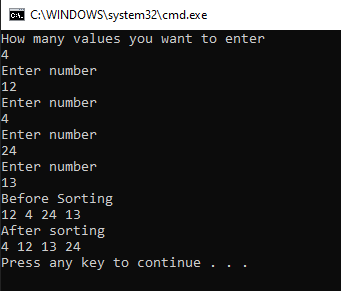
Console.WriteLine();

}

}

}

**Output:**



**Task # 02: Create static tree and perform inorder, preorder and post order traversal. Also search a required node in the tree.**

**Solution:**

namespace Tree

{

class Program

{

static int root = 0;

static string[] str = new string[10];

public void Root(string key)

{

str[0] = key;

}

public void Print()

{

for (int i = 0; i < 10; i++)

{

if (str[i] != null)

{

Console.Write(str[i]);

}

else

{

Console.Write("-");

}

}

Console.WriteLine();

}

public void Left(string key, int root\_index)

{

int leftindex = (root\_index \* 2) + 1;

if (str[root\_index] == null)

{

Console.WriteLine("No parent found at " + root\_index);

}

else

{

str[leftindex] = key;

}

}

public void Right(string key, int root)

{

int t = (root \* 2) + 2;

if (str[root] == null)

{

Console.WriteLine("No parent found at " + root);

}

else

{

str[t] = key;

}

}

public void InOrder(int root)

{

if (root < str.Length)

{

InOrder((root \* 2) + 1);

Console.Write(str[root]);

InOrder((root \* 2) + 2);

}

}

public void PreOrder(int root)

{

if (root < str.Length)

{

if (str[root] != "")

{

Console.Write(str[root]);

PreOrder((root \* 2) + 1);

PreOrder((root \* 2) + 2);

}

}

}

public void PostOrder(int root)

{

if (root < str.Length)

{

if (str[root] != "")

{

PostOrder((root \* 2) + 1);

PostOrder((root \* 2) + 2);

Console.Write(str[root]);

}

}

}

static void Main(string[] args)

{

Program p = new Program();

p.Root("d");

p.Right("y", 0);

p.Left("k", 0);

p.Right("q", 1);

p.Left("c", 2);

p.Right("h", 1);

p.Left("w", 2);

Console.WriteLine("Tree is:");

p.Print();

Console.WriteLine("InOrder of a tree:");

p.InOrder(0);

Console.WriteLine("\nPre Order of a tree:");

p.PreOrder(0);

Console.WriteLine("\nPost Order of a tree:");

p.PostOrder(0);

Console.WriteLine();

}

}

}

**Output:**

