**Task # 1:** Implement bucket sort using linked list.

**Solution**

**Radix Sort**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bucket\_and\_Radix\_Sort

{

class Program

{

public static void radixSort(int[] input, int radix, int width)

{

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

{

radixSingleSort(input, i, radix);

}

}

public static void radixSingleSort(int[] input, int position, int radix)

{

int numItems = input.Length;

int[] countArray = new int[radix];

foreach (int value in input)

{

countArray[getDigit(position, value, radix)]++;

}

// Adjust the count array

for (int j = 1; j < radix; j++)

{

countArray[j] += countArray[j - 1];

}

int[] temp = new int[numItems];

for (int tempIndex = numItems - 1; tempIndex >= 0; tempIndex--)

{

temp[--countArray[getDigit(position, input[tempIndex], radix)]] =

input[tempIndex];

}

for (int tempIndex = 0; tempIndex < numItems; tempIndex++)

{

input[tempIndex] = temp[tempIndex];

}

}

public static int getDigit(int position, int value, int radix)

{

return value / (int)Math.Pow(radix, position) % radix;

}

static void Main(string[] args)

{

int[] radixArray = { 75, 1, 7, 89, 74, 152, 2, 0, 45, 78, 100 };

Console.WriteLine("WITHOUT SORTING");

foreach (var item in radixArray)

{

Console.Write(item + " ");

}

Console.WriteLine();

radixSort(radixArray, 10, 4);

Console.WriteLine("USING RADIX SORT");

for (int i = 0; i < radixArray.Length; i++)

{

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

}

Console.WriteLine();

}

}

}

**Text, letter

Description automatically generatedOutput**

**Bucket Sort**

using System;

using System.Collections.Generic;

namespace Bucket\_and\_Radix\_Sort

{

class Program

{

public static void BucketSort(ref int[] arr)

{

int min = int.MaxValue;

int max = 0;

for (int i = 0; i < arr.Length; i++)

{

if (arr[i] < min)

min = arr[i];

if (arr[i] > max)

max = arr[i];

}

List<int>[] b = new List<int>[max - min + 1];

for (int i = 0; i < b.Length; i++)

{

b[i] = new List<int>();

}

for (int i = 0; i < arr.Length; i++)

{

b[arr[i] - min].Add(arr[i]);

}

int k = 0;

for (int i = 0; i < b.Length; i++)

{

if (b[i].Count > 0)

{

for (int j = 0; j < b[j].Count; j++)

{

arr[k] = b[i][j];

k++;

}

}

}

}

static void Main(string[] args)

{

int[] Array = { 75, 1, 7, 89, 74, 152, 2, 0, 45, 78, 100 };

Console.WriteLine("WITHOUT SORTING");

foreach (var item in Array)

{

Console.Write(item + " ");

}

Console.WriteLine();

BucketSort(ref Array);

Console.WriteLine("USING BUKET SORT");

BucketSort(ref Array);

for (int i = 0; i < Array.Length; i++)

{

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

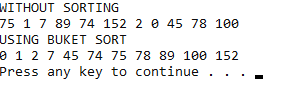
}

Console.WriteLine();

}

}

}

**Output**

**Task # 2**. Create static tree and perform inorder, preorder and post order traversal. Also search a required node in the tree.

**Solution**

using System;

namespace Bucket\_and\_Radix\_Sort

{

class Program

{

class TreeNode

{

public int key;

public TreeNode left, Right;

public TreeNode(int key)

{

this.key = key;

this.left = this.Right = null;

}

public TreeNode getLeft()

{

return this.left;

}

public int GetKey()

{

return this.key;

}

public TreeNode getRight()

{

return this.Right;

}

public void setLeft(int key)

{

this.left = new TreeNode(key);

}

public void setRight(int key)

{

this.Right = new TreeNode(key);

}

}

class BinaryTree

{

public TreeNode Root;

public BinaryTree()

{

this.Root = null;

}

public BinaryTree(int key)

{

this.Root = new TreeNode(key);

}

public void setRoot(int key)

{

this.Root = new TreeNode(key);

}

public TreeNode getRoot()

{

return this.Root;

}

public void Inorder(TreeNode Node)

{

if (Node == null)

return;

else

{

Inorder(Node.left);

Console.Write(Node.GetKey() + " ");

Inorder(Node.Right);

}

}

public void Preorder(TreeNode Node)

{

if (Node == null)

return;

else

{

Console.Write(Node.GetKey() + " ");

Preorder(Node.left);

Preorder(Node.Right);

}

}

public void PostOrder(TreeNode Node)

{

if (Node == null)

return;

else

{

PostOrder(Node.left);

PostOrder(Node.Right);

Console.Write(Node.GetKey() + " ");

}

}

}

static bool searchtree(TreeNode node, int key)

{

if (node == null)

return false;

if (node.key == key)

return true;

bool res1 = searchtree(node.left, key);

if (res1) return true;

bool res2 = searchtree(node.Right, key);

return res2;

}

static void Main(string[] args)

{

BinaryTree tree = new BinaryTree();

tree.setRoot(7);

tree.Root.setLeft(5);

tree.Root.setRight(4);

tree.Root.left.setRight(9);

tree.Root.left.setLeft(20);

tree.Root.Right.setRight(19);

tree.Root.Right.setLeft(26);

tree.Root.Right.setLeft(106);

System.Console.WriteLine("InOrder Traversal");

tree.Inorder(tree.Root);

System.Console.WriteLine();

System.Console.WriteLine("PreOrder Traversal");

tree.Preorder(tree.Root);

System.Console.WriteLine();

System.Console.WriteLine("PostOrder Traversal");

tree.PostOrder(tree.Root);

Console.WriteLine();

Console.WriteLine("Which number do you search(present yes/No) In Answer");

int key = Convert.ToInt32(Console.ReadLine());

if (searchtree(tree.Root, key))

{

Console.WriteLine("YES");

}

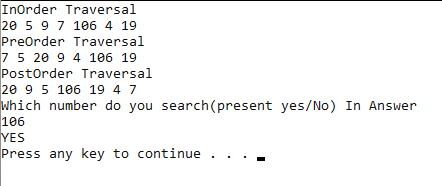
else

{

Console.WriteLine("NO");

}

} }}

**Output**