Bubble Sort

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SortingDemos

{

class BubbleSortDemo

{

public static void Main()

{

int[] num = new int[] { 12, 4, 25, 10, 2, 13, 18, 80 };

Console.WriteLine("Elements before Sorting");

PrintElements(num);

BubbleSort(num);

Console.WriteLine("Elements after Sorting");

PrintElements(num);

}

static void PrintElements(int[] num)

{

foreach(int x in num)

Console.WriteLine(x);

}

static void BubbleSort(int [] num)

{

int n = num.Length;

int temp;

for (int i = 0; i < n - 1;i++)

{

for(int j=0;j<n-i-1; j++)

{

if(num[j] > num[j+1])

{

temp = num[j];

num[j] = num[j + 1];

num[j + 1] = temp;

}

}

}

}

}

}

Insertion Sort

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SortingDemos

{

class InsertionSortDemo

{

static void Main()

{

int[] num = new int[] { 12, 4, 25, 10, 2, 13, 18, 80 };

Console.WriteLine("Elements before Sorting");

PrintElements(num);

InsertionSort(num);

Console.WriteLine("Elements after Sorting");

PrintElements(num);

}

static void PrintElements(int[] num)

{

foreach (int x in num)

Console.WriteLine(x);

}

static void InsertionSort(int[] num)

{

int x, j = 0;

for (int i = 1; i < num.Length; i++)

{

x = num[i];

j = i - 1;

while (j >= 0 && num[j] > x)

{

num[j + 1] = num[j];

j = j - 1;

}

num[j + 1] = x;

}

}

}

}

Selection Sort

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SortingDemos

{

class SelectionSortDemo

{

static void Main()

{

int[] num = new int[] { 12, 4, 25, 10, 2, 13, 18, 80 };

Console.WriteLine("Elements before Sorting");

PrintElements(num);

SelectionSort(num);

Console.WriteLine("Elements after Sorting");

PrintElements(num);

}

static void PrintElements(int[] num)

{

foreach (int x in num)

Console.WriteLine(x);

}

static void SelectionSort(int[] num)

{

int temp;

for(int i=0;i<num.Length-1;i++)

{

for(int j=i+1; j<num.Length;j++)

{

if(num[i] > num[j])

{

temp = num[i];

num[i] = num[j];

num[j] = temp;

}

}

}

}

}

}

Merge Sort

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SortingDemos

{

class MergeSortDemo

{

static void Main()

{

int[] array1 = new int[] { 4, 14, 25, 40, 75, 83, 88, 92 };

int[] array2 = new int[] { 1, 3, 15, 18, 90 };

int[] sortedArray = new int[(array1.Length + array2.Length)-1];

Console.WriteLine("Elements before Sorting");

// PrintElements(array1, array2);

MergeSort(array1,array2, sortedArray);

Console.WriteLine("Elements after Sorting");

PrintElements(sortedArray);

}

static void PrintElements(int[] num)

{

foreach (int x in num)

Console.WriteLine(x);

}

static void MergeSort(int[] array1, int [] array2 , int [] sortedArray)

{

int i=0, j=0, k=0;

Console.WriteLine(array1.Length);

Console.WriteLine(array2.Length);

while(i<array1.Length && j < array2.Length)

{

if(array1[i] > array2[j])

{

sortedArray[k] = array2[j];

j++;

k++;

}

else

{

sortedArray[k] = array1[i];

i++;

k++;

}

}

Console.WriteLine("i is " + i);

Console.WriteLine("j is " + j);

if(i==array1.Length-1 && j < array2.Length)

{

while(j<array2.Length)

{

sortedArray[k] = array2[j];

j++;

k++;

}

}

else if (i < array1.Length-1 && j == array2.Length)

{

while (i < array1.Length)

{

sortedArray[k] = array1[i];

i++;

k++;

}

}

}

}

}

BST

using System;

namespace ConsoleApp1

{

class Node

{

public int Data;

public Node Left;

public Node Right;

public void Display()

{

Console.WriteLine(Data + "->");

}

}

class BinarySearchTree

{

public Node root;

public BinarySearchTree()

{

root = null;

}

public void Insert(int data)

{

Node newNode = new Node();

newNode.Data = data;

if (root == null)

{

root = new Node();

root = newNode;

Console.WriteLine("Root Node added");

}

else {

Node ptr=root;

Node parent;

while (true)

{

parent = ptr;

if (data < ptr.Data)

{

ptr = ptr.Left;

if (ptr == null)

{

parent.Left = newNode;

Console.WriteLine("Added at left");

break;

}

}

else

{

ptr = ptr.Right;

if (ptr == null)

{

parent.Right = newNode;

Console.WriteLine("Added at right");

break;

}

}

}

}

}

// Left Root Right

public void InOrderTraversal(Node parent)

{

if(parent!=null)

{

InOrderTraversal(parent.Left);

Console.WriteLine(parent.Data);

InOrderTraversal(parent.Right);

}

}

public void PreOrderTraversal(Node parent)

{

if (parent != null)

{

Console.WriteLine(parent.Data);

PreOrderTraversal(parent.Left);

PreOrderTraversal(parent.Right);

}

}

public void PostOrderTraversal(Node parent)

{

if (parent != null)

{

PostOrderTraversal(parent.Left);

PostOrderTraversal(parent.Right);

Console.WriteLine(parent.Data);

}

}

}

class Program

{

static void Main(string[] args)

{

BinarySearchTree searchTree = new BinarySearchTree();

searchTree.Insert(10);

searchTree.Insert(12);

searchTree.Insert(15);

searchTree.Insert(5);

searchTree.Insert(7);

Console.WriteLine("InOrder");

searchTree.InOrderTraversal(searchTree.root);

Console.WriteLine("PreOrder");

searchTree.PreOrderTraversal(searchTree.root);

Console.WriteLine("PostOrders");

searchTree.PostOrderTraversal(searchTree.root);

}

}

}