**Sorting Assignments in C#:**

**Assignment 1: Bubble Sort**

**pe**

**Problem Statement:** Sort the below list using Bubble sort:

67, 23,93,21,34,78,90,30

Refer the below sample code snippet:

|  |
| --- |
| public List<int> BubbleSort(List<int> a)  {  int temp;  **// traversing each element of the list**  for(int i=1; i<=a.Count; i++)  for(int j=0; j<a.Count-i; j++)  **--put logic here if first element is greater than second element then swap**  return a;  } |

**Note:**

* You call the above method from a console application

or

You can create a form and put a button and on this button click you can call the below “BubbleSort” method.

* Each element of the unsorted list is compared to the next element and if the value of first element is greater than the value of the second element, then they are swapped.

**Assignment 2: Quick Sort**

**pe**

**Problem Statement:** Sort the below list using Quick sort:

2 5 -4 11 0 18 22 67 51 6

Output should be:

-4 0 2 5 6 11 18 22 51 67

Step 1: Pick an element, called a Pivot, from the list.

Step 2: Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position (This is called the **partition** operation)

|  |
| --- |
| private static int Partition(int[] arr, int left, int right)  {  int pivot = arr[left];  while (true)  {  while (arr[left] < pivot)  {  left++;  }  while (arr[right] > pivot)  {  right--;  }  --complete the logic of reordering  else  {  return right;  }  }  } |

Step 3: Recursively sort the sub-list of lesser elements and the sub-list of greater elements.

|  |
| --- |
| private static void Quick\_Sort(int[] arr, int left, int right)  {  if (left < right)  {  int pivot = Partition(arr, left, right);  --put Recursively sorting logic here    } |

**Assignment 3: Merge sort**

**pe**

**Problem Statement:** Sort the below list using Merge sort:

38, 27, 43,3,9,82,10

* If the list is of length 0 or 1, then it is already **sorted**. Otherwise:
* **Divide** the unsorted list into two sublists of about **half** the size.
* Sort each sublist **recursively** by re-applying merge sort.

|  |
| --- |
| private static List<int> MergeSort(List<int> unsorted)  {  if (unsorted.Count <= 1)  return unsorted;  List<int> left = new List<int>();  List<int> right = new List<int>();  int middle = unsorted.Count / 2;  ---put the logic to divide the unsorted list    left = MergeSort(left);  right = MergeSort(right);  return Merge(left, right);  } |

* **Merge** the two sublists back into one sorted list.

**Assignment 4: Insertion sort**

**pe**

**Problem Statement:** Sort the below array of strings using Insertion sort:

"John Doe", "Doe John", "Another Name", "Name Another"

Hints:

|  |
| --- |
| class Program  {  static void Main()  {  --take a string array  InsertSort(names);  foreach (var item in names)  {  Console.WriteLine(item);  }  }  --using IComparable do sorting  static void InsertSort(IComparable[] array)  {  int i, j;  for (i = 1; i < array.Length; i++)  {  IComparable value = array[i]  //put the insertion sort logic here  }  }  } |

**Search in C# Assignments:**

**Assignment 1: Linear Search**

**pe**

100, 50, 20, 40, 10, 60, 80, 70, 90, 30, 50, 20, 40, 10, 60, 80, **Problem Statement:** Find 25 from the below list:

12 13 10 25 47 45 66 87 77 30100, 50, 20,

**Note:** In linear/sequential search, we search given item, sequentially one by one, if we found item, then we return location. It may also possible item is not find till last item of list.

**Assignment 2: Binary Search**

**pe**

**Problem Statement:** Find 30 from the below list:

10 20 30 40 50 60 70 80 90 100

* Analyse the below algorithm and refer it to implement in C#:

|  |
| --- |
| 1st comparison:  LOW = 1  HIGH = 10  MID = (1+10) /2 = 5  Now if LIST[MID] == ITEM, (50 == 30) No Match.  Then find new indices by dividing list. Here ITEM is less than mid value then:  LOW = 1  HIGH = MID -1 =4  MID = (1+4)/2 = 2    2nd comparison:  IF LIST[MID] == ITEM, (20==30) No Match.  Then find new indices by dividing list. Here ITEM is greater than mid value then:  LOW = MID + 1 = 3  HIGH = 4  MID = (3+4)/2 = 3  3rd comparison:  IF LIST[MID] == ITEM, (30==30), No ITEM found at position 3. |

Note:

The above implementation is the core logic of binary search, instead of implementing the above logic this is already implemented by the built in method “BinarySearch” in C#.

After implementing above logic, try to use “BinarySearch” method, it will reduce number of lines of code.

--Thanks--