Deakin University

Data Structures and Algorithms

OnTrack Submission

Implementation of recursive sorting algorithms

Submitted By:
Dongqi Shen
shendong
2019/08/03 15:23

Tutor:
Soudeh Kasiri bidhendi

August 3, 2019

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Vector
{
    class RandomizedQuickSort : ISorter
    {
        public void Sort<K>(K[] sequence, IComparer<K> comparer) where K : IComparable<K>
        {
            QuickSort(sequence,comparer,0,sequence.Length-1);
        }
        public void QuickSort<K>(K[] sequence, IComparer<K> comparer, int a, int b) where K : IComparable<K>
        {
            if (a >= b) return;
            int left = a;
            int right = b-1;
            K pivot = sequence[b];
            K temp;
            while (left <= right)
            {
                while (left <= right && comparer.Compare(sequence[left], pivot) < 0) left++;
                while (left <= right && comparer.Compare(sequence[right], pivot) > 0) right--;
                if (left <= right)
                {
                    temp = sequence[left]; sequence[left] = sequence[right]; sequence[right] = temp;
                    left++; right--;
                }
            }
            temp = sequence[left];
            sequence[left] = sequence[b];
            sequence[b] = temp;
            QuickSort(sequence, comparer, a, left - 1);
            QuickSort(sequence, comparer, left + 1, b);
        }
    }

}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Vector
{
    class MergeSortTopDown:ISorter
    {
        public void Sort<K>(K[] sequence, IComparer<K> comparer) where K :
        ↪  IComparable<K>
        {
            mergeSort(sequence, comparer);
        }
        public void merge<K>(K[] S1,K[] S2,K[] S,IComparer<K> comparer) where K :
        ↪  IComparable<K>
        {

            int i = 0, j = 0;
            while (i + j < S.Length)
            {
                if (j == S2.Length || (i < S1.Length && comparer.Compare(S1[i],
                ↪  S2[j]) < 0))
                {
                    S[i + j] = S1[i++];
                }
                else
                {
                    S[i + j] = S2[j++];
                }
            }
        }
        public void mergeSort<K>(K[] S,IComparer<K> comparer) where K :
        ↪  IComparable<K>
        {
            int n = S.Length;
            if (n < 2) return;

            int mid = n / 2;
            K[] S1 = new K[mid];
            K[] S2 = new K[S.Length-mid];
            Array.Copy(S, 0, S1, 0, mid);
            Array.Copy(S, mid, S2, 0, mid);

            mergeSort(S1, comparer);
            mergeSort(S2, comparer);
            merge(S1, S2, S, comparer);
        }
    }
}
```

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Vector
{
    class MergeSortBottomUp:ISorter
    {
        public void Sort<K>(K[] sequence, IComparer<K> comparer) where K : IComparable<K>
        {
            MergesortBottomUp(sequence, comparer);
        }
        public void Merge<K>(K[] a, K[] b, IComparer<K> comparer, int start, int inc) where K : IComparable<K>
        {
            int end1 = Math.Min(start + inc, a.Length);
            int end2 = Math.Min(start + 2 * inc, a.Length);

            int x = start;
            int y = start + inc;
            int z = start;
            while (x < end1 && y < end2)
            {
                if (comparer.Compare(a[x], a[y]) < 0)
                {
                    b[z++] = a[x++];
                }else
                {
                    b[z++] = a[y++];
                }

            }
            if (x < end1)
            {
                Array.Copy(a, x, b, z, end1 - x);
            }
            else if(y<end2)
            {
                Array.Copy(a, y, b, z, end2 - y);
            }
        }
        public void MergesortBottomUp<K>(K[] orig, IComparer<K> comparer) where K : IComparable<K>
        {
            int n = orig.Length;
            K[] src = orig;
            K[] dest = new K[n];
            K[] temp;
            for(int i =1; i<n;i*=2)
            {
                for(int j=0;j<n;j+=2*i)
                {
                    Merge(src, dest, comparer, j, i);

                }
                temp = src; src = dest; dest = temp;
            }
            if(orig!=src)
            {
                Array.Copy(src, 0, orig, 0, n);
            }
        }
    }
}
```