Lab: Sorting and Searching Algorithms

This document defines the **in-class exercises** assignments for the "Algorithms" course @ Software University.

Part I – Sorting

1. Merge Sort

Sort an array of elements using the famous merge sort

Examples

Input	Output
5 4 3 2 1	1 2 3 4 5

Hints

Create your **`Mergesort`** generic class with a single Sort method

```
public class Mergesort<T> where T : IComparable
{
    public static void Sort(T[] arr)
    {
    }
}
```

Create an **auxiliary array** that will help with merging subarrays

```
private static T[] aux;
```

Now to implement the **`Merge()`** method

```
private static void Merge(T[] arr, int lo, int mid, int hi)
```

As the two subarrays are sorted, if the **largest element in the left** is smaller than the **smallest in the right**, the two subarrays are **already merged**

```
if (IsLess(arr[mid], arr[mid + 1]))
{
    return;
}
```

If they are not, however, **transfer all elements to the auxiliary array**

```
for (int index = lo; index < hi + 1; index++)
{
    aux[index] = arr[index];
}
```

Then **merge them back** in the main array

```
int i = lo;
int j = mid + 1;
for (int k = lo; k <= hi; k++)
{
    if (i > mid)
    {
        arr[k] = aux[j++];
    }
    else if (j > hi)
    {
        arr[k] = aux[i++];
    }
    else if (IsLess(aux[i], aux[j]))
    {
        arr[k] = aux[i++];
    }
    else
    {
        arr[k] = aux[j++];
    }
}
```

Now to create the recursive **`Sort()`** method

```
private static void Sort(T[] arr, int lo, int hi)
```

If there is **only one element** in the subarray, it is **already sorted**

```
if (lo >= hi)
{
    return;
}
```

If not, however, you need to **split it into two subarrays**, **sort them recursively** and then **merge them on the way up** of the recursion (as a post-action)

```
Sort(arr, lo, mid);
Sort(arr, mid + 1, hi);
Merge(arr, lo, mid, hi);
```

You can now call the **`Sort()`** method

```
public static void Sort(T[] arr)
{
    aux = new T[arr.Length];
    Sort(arr, 0, arr.Length - 1);
}
```

2. Quicksort

Sort an array of elements using the famous quicksort

Examples

Input	Output
5 4 3 2 1	1 2 3 4 5

Hints

You can learn about the Quicksort algorithm from Wikipedia.

A great tool for visualizing the algorithm (along with many others) is available at Visualgo.net.

The algorithm in short:

- Quicksort takes unsorted partitions of an array and sorts them
- We choose the **pivot**
  - We pick the first element from the unsorted partition and move it in such a way that all smaller elements are on its left and all greater, to its right
- With pivot moved to its correct place, we now have two unsorted partitions – one to the left of it and one to the right
- **Call the procedure recursively** for each partition
- The bottom of the recursion is when a partition has a size of 1, which is by definition sorted

First, define the **class** and its **sorting method**:

```
public class Quick
{
    public static void Sort<T>(T[] a) where T : IComparable<T>
    {
        // TODO: Shuffle
        Sort(a, 0, a.Length - 1);
    }

    private static void Sort<T>(T[] a, int lo, int hi) where T : IComparable<T>
    {
    }
}
```

Now to implement the private **`Sort()`** method. Don't forget to handle the **bottom of the recursion**

```
private static void Sort<T>(T[] a, int lo, int hi) where T : IComparable<T>
{
    if (lo >= hi)
    {
        return;
    }
}
```

First, find the pivot index and rearange the elements, then sort the left and right partitions recursively:

```
int p = Partition(a, lo, hi);
Sort(a, lo, p - 1);
Sort(a, p + 1, hi);
```

Now to choose the pivot point.. we need to create a method called **`Partition()`**

```
private static int Partition<T>(T[] a, int lo, int hi) where T : IComparable<T>
{
}
```

If there is **only one element**, it is already partitioned and the index of the pivot is the index of its only element

```
if (lo >= hi)
{
    return lo;
}
```

Finding the pivot point involves **rearanging all elements** in the partition so it satisfies the condition **all elements to the reft of the pivot to be smaller** from it, and **all elements to its right to be greater** than it

```
int i = lo;
int j = hi + 1;
while (true)
{
    while (Less(a[++i], a[lo]))
    {
        if (i == hi) break;
    }

    while (Less(a[lo], a[--j]))
    {
        if (j == lo) break;
    }

    if (i >= j) break;
    Swap(a, i, j);
}

Swap(a, lo, j);
return j;
```

Part II – Implement Binary Search

3. Implement Binary Search

Implement an algorithm that finds the index of an element in a sorted array of integers in logarithmic time

Examples

Input	Output	Comments
1 2 3 4 5 1	0	Index of 1 is 0
-1 0 1 2 4 1	2	Index of 1 is 2

Hints

First, if you're not familiar with the concept, read about binary search in Wikipedia.

Here you can find a tool which shows visually how the search is performed.

In short, if we have a **sorted collection** of comparable elements, instead of doing linear search (which takes linear time), we can eliminate half the elements at each step and finish in logarithmic time.

Binary search is a **divide-and-conquer** algorithm; we start at the middle of the collection, if we haven't found the element there, there are three possibilities:

- The element we're looking for is smaller – then look to the left of the current element, we know all elements to the right are larger;
- The element we're looking for is larger – look to the right of the current element;
- The element is not present, traditionally, return -1 in that case.

Start by defining a class with a method

```
public class BinarySearch
{
    public static int IndexOf(int[] arr, int key)
    {
    }
}
```

Inside the method, define two variables defining the bounds to be searched and a while loop

```
int lo = 0;
int hi = arr.Length - 1;
while (lo <= hi)
{
    // TODO: Find index of key
}

return -1;
```

Inside the while loop, we need to find the midpoint

```
int mid = lo + (hi - lo) / 2;
```

If the key is to the left of the midpoint, move the right bound. If the key is to the right of the midpoint, move the left bound

```
if (key < arr[mid])
{
    hi = mid - 1;
}
else if (key > arr[mid])
{
    lo = mid + 1;
}
else
{
    return mid;
}
```

That's it! Good job!