Path Between Numbers

Given two **+ve integers *X* & *Y***, it's desired to convert ***X*** into ***Y***. Three types of operations are allowed:

1. **Multiply** the current number by 2 (i.e. replace the number ***X*** by **2 × *X***)
2. **Subtract** 1 from the current number (i.e. replace the number ***X*** by ***X* – *1***)
3. **Append** the digit 1 to the right of current number (i.e. replace the number ***X*** by **10 × *X* + 1**).

Find the **minimum number of operations** to convert X into Y?

Input:

- X = from 1 to 5000
- Y = from 500 to 5000000

Function to Implement

```
static int Find(int X, int Y)
```

`PathBetweenNumbers.cs` includes this method.

<returns> min number of operations to convert X into Y

Example

```
X = 4;
Y = 7;
expected = 2;

X = 2;
Y = 31;
expected = 3;
```

C# Help

Queues

Creation

To create a queue of a certain type (e.g. string)

```
Queue<string> myQ = new Queue<string>() //default initial size

Queue<string> myQ = new Queue<string>(initSize) //given initial size
```

Manipulation

1. `myQ.Count` ➔ get actual number of items in the queue
2. `myQ.Enqueue("myString1")` ➔ Add new element to the queue
3. `myQ.Dequeue()` ➔ return the top element of the queue (FIFO)

Lists

Creation

To create a list of a certain type (e.g. string)

```
List<string> myList1 = new List<string>() //default initial size

List<string> myList2 = new List<string>(initSize) //given initial size
```

Manipulation

4. `myList1.Count` ➔ get actual number of items in the list
5. `myList1.Sort()` ➔ Sort the elements in the list (ascending)
6. `myList1[index]` ➔ Get/Set the elements at the specified index
7. `myList1.Add("myString1")` ➔ Add new element to the list
8. `myList1.Remove("myStr1")` ➔ Remove the 1st occurrence of this element from list
9. `myList1.RemoveAt(index)` ➔ Remove the element at the given index from the list
10. `myList1.Contains("myStr1")` ➔ Check if the element exists in the list

Dictionary (Hash)

Creation

To create a dictionary of a certain key (e.g. string) and value (e.g. array of strings)

```
//default initial size
Dictionary<string, string[]> myDict1 = new  Dictionary<string, string[]>();

//given initial size
Dictionary<string, string[]> myDict2 = new  Dictionary<string, string[]>(size);
```

Manipulation

1. `myDict1.Count` ➔ Get actual number of items in the dictionary
2. `myDict1[key]` ➔ Get/Set the value associated with the given key in the dictionary
3. `myDict1.Add(key, value)` ➔ Add the specified key and value to the dictionary
4. `myDict1.Remove(key)` ➔ Remove the value with the specified key from the dictionary
5. `myDict1.ContainsKey(key)` ➔ Check if the specified key exists in the dictionary

Creating 1D array

```
int [] array = new int [size]
```

Creating 2D array

```
int [,] array = new int [size1, size2]
```

Length of 1D array

```
int arrayLength = my1DArray.Length
```

Length of 2D array

```
int array1stDim = my2DArray.GetLength(0)

int array2ndDim = my2DArray.GetLength(1)
```

Sorting single array

Sort the given array in ascending order

```
Array.Sort(items);
```

Sorting parallel arrays

Sort the first array "master" and re-order the 2nd array "slave" according to this sorting

```
Array.Sort(master, slave);
```