Find Target

In a new board game of size NxN, a single player should reach a target. However, the player has only four allowed movements:

1. Two right and three up.
2. Two right and three down.
3. Two left and three up.
4. Two left and three down.

Given the location of the player and the target, it's desired to calculate **the minimum number of moves** that allows the player to reach its target. If can't reach the target, return **-1**.

Note

- Player can't move outside the board.
- Position (1, 1) is the top-left corner and position (N, N) is the bottom-right corner.

Input:

- N = from 2^2 to 2^{13}

Function to Implement

```
static int Play(int N, Location src, Location target)
```

`FindTarget.cs includes this method.`

- "N": board dimension
- "src": current location of the player
- "target": target location

<returns> min number of moves to reach the target OR -1 if can't reach the target

Example

```
N = 8;
src.X = 1; src.Y = 1;
dst.X = 5; dst.Y = 1;
expected = 2;
```

```
N = 4;
s.X = 2; s.Y = 3;
d.X = 4; d.Y = 3;
expected = -1;
```

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);
```