Exercise 1.4.20: Bitonic search. An array is bitonic if it is comprised of an increasing sequence of integers followed immediately by a decreasing sequence of integers. Write a program that, given a bitonic array of N distinct int values, determines whether a given integer is in the array. Your program should use 3lg N compares in the worst case.

Solution:

- example array; [1,2,3,4,5,4,3,2]
- first, find bitonic point(the maximal/minimal point where an array starts increasing or decreasing on either side of it). We can do this using Binary Search.
- Search LHS and RHS of bitonic point to find the key.
- This results in 3 binary searches which means time complexity is $O(3logN)$

Pseudocode below

```
//PseudoCode: Bitonic search
//By Anando Zaman
//Example Bitonic Array [1,2,3,4,5,4,3,2], strictly increasing and then strictly decreasi

//finds the bitonic point in the array
function bitonicpoint_find(arr[], int left, int right):

    int mid = left + (right-left)/2

    if((arr[mid+1] < arr[mid]) && (arr[mid-1] < arr[mid])):
        //Base case
        //This point is larger than both adjacent points, therefore it is bitonic point
        return mid

    else if((arr[mid+1] > arr[mid]) && (arr[mid-1] < arr[mid])):

        //this means max lies on RHS since arr[mid+1] is larger than arr[mid].
        bitonicpoint_find(arr,left,mid+1)

    else if((arr[mid+1] < arr[mid]) && (arr[mid-1] > arr[mid])):
        //means max is on LHS
        bitonicpoint_find(arr,mid-1,right)


//searches Left half for the key
function searchLeft(arr,left,right,key):

    if(right>left):
        mid = left + (right-left)/2
        if(key == arr[mid]):
            return mid

        //Check RHS of Midpoint
        else if(key > arr[mid]):
            return searchLeft(arr,mid+1,right)

        //check LHS of Midpoint
        else if(key < arr[mid]):
            return searchLeft(arr,low,mid-1)

    return -1 //if right<left or if no match found
```

```
//searches Right half for the key
function searchRight(arr,left,right,key):

    if(right>left):
        mid = left + (right-left)/2
        if(key == arr[mid]):
            return mid

        //Check RHS of Midpoint
        else if(key > arr[mid]):
            return searchLeft(arr,mid+1,right)

        //check LHS of Midpoint
        else if(key < arr[mid]):
            return searchLeft(arr,low,mid-1)

    return -1 //if right<left or if no match found


function SearchBitonic(arr[], key, bitonic_point):
    //Base case
    if(key > arr[bitonic_point]):
        // since bitonic poiint is asolute max value
        //Therefire, the key CANNOT be larger. If larger, then return -1 as it
        //doesn't exist in the array
        return -1

    else if(key == arr[bitonic_point]):
        return bitonic_point

    else:
        temp = searchLeft(arr,0,bitonic_point-1,key)
        if(temp != -1):
            return temp;

        //Search RHS of Bitonic Point to see if the key exists
        else:
            return searchRight(arr, bitonic_point+1,arr.length-1,key)
```