Practical 06
Recursion

1. Fibonacci Sequence

The Fibonacci sequence is a sequence of integer values in which the first two values are 0, 1, and each subsequent value is the sum of the two previous values.

For example, the first few terms of the sequence are:

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 …

The n^{th} Fibonacci number can be computed by the recurrence relation (for $n \geq 0$):

$$fib(n) = \begin{cases} fib(n-1) + fib(n-2), & \text{if } n > 1 \\ n, & \text{if } n = 1 \text{ or } n = 0 \end{cases}$$

Design and implement a <u>recursive</u> Python function – $fib(n)$, to compute the n^{th} Fibonacci number.

2. Recursive Binary Search

We have seen an <u>iterative implementation</u> of the Binary Search algorithm in “Practical 03 – Search”. The code for that implementation is listed below:

```python
# An iterative implementation of Binary Search
def binarySearch( theValues, target ):
    # Start with the entire sequence of elements
    low = 0
    high = len( theValues ) - 1

    # Repeatedly subdivide the sequence in half
    # until the target is found
    while low <= high:
        # Find the midpoint of the sequence
        mid = (high + low) // 2

        # Does the midpoint contain the target?
        # If yes, return midpoint (i.e. index of the list)
        if theValues[mid] == target:
            return mid
        # Or is the target before the midpoint?
        elif target < theValues[mid]:
            high = mid - 1
        # Or is the target after the midpoint?
        else:
            low = mid + 1

    # If the sequence cannot be subdivided further,
    # target is not in the list of values
    return -1
```

The following code below is an incomplete implementation of the Binary Search algorithm using recursion.

```
# A recursive implementation of Binary Search
def recBinarySearch( target, theValues, first, last ):
    # If the sequence of values cannot be subdivided further,
    # we are done
    if __________________:         # BASE CASE #1
        return False

    else:
        # Find the midpoint of the sequence
        mid = ____________________

        # Does the element at the midpoint contain the target?
        if ____________________________:
            return True              # BASE CASE #2

        # or does the target precede the element at the midpoint?
        elif target < theValues[mid]:
            ________________________________________________

        # or does the target follows the element at the midpoint?
        else:
            ________________________________________________
```

Complete the above recursive implementation of the Binary Search algorithm by providing the rest of the required code.

3. Recursive Re-arrangement of Sequence

Write a short recursive Python function that re-arranges a sequence of integer values so that all the even values appear before all the odd values.

Sample Output:

```
Original List:          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Re-arranged List:       [10, 8, 2, 6, 4, 5, 7, 3, 9, 1]

Process finished with exit code 0
```

-- End of Practical --