L4 Problems on Functional Recursion

Sunday, December 11, 2022 11:12 AM

In this Lecture we practice some Recursion problems for understanding the Recursion Indepth ☺

1) Reverse the Array

Now our task is Reverse the array using Recursion

Input = [1,2,3,4,5]
output = [5,4,3,2,1]

Thought Process

Now the first approach that comes to the mind is two pointers Approach Creating a two pointers

→ left pointer
→ Right Pointer

→ And swap to each others till the left and Right pointers comes to the same Index

Example

arr = [1,2,3,4,5] (0 1 2 3 4)
L R

swap

arr = [1,2,3,4,5] ⇒ arr = [5,2,3,4,1] ⇒ arr = [5,4,3,2,1] ⇒ arr = [5,4,3,2,1]

L=0, R=4 → after swap L=L+1, R=R-1
L=1, R=3 → after swap L=L+1, R=R-1
L=2, R=2 → after swap L=L+1, R=R-1

LR → that is the point where stop the operation

→ Answer ☺

```
myfunction (left, right){
    if left >= Right {
        return;
    }
    Swap (arr[left], arr[right]);
    myfunction(left+1, right-1);
}
```

Base case

Swap function

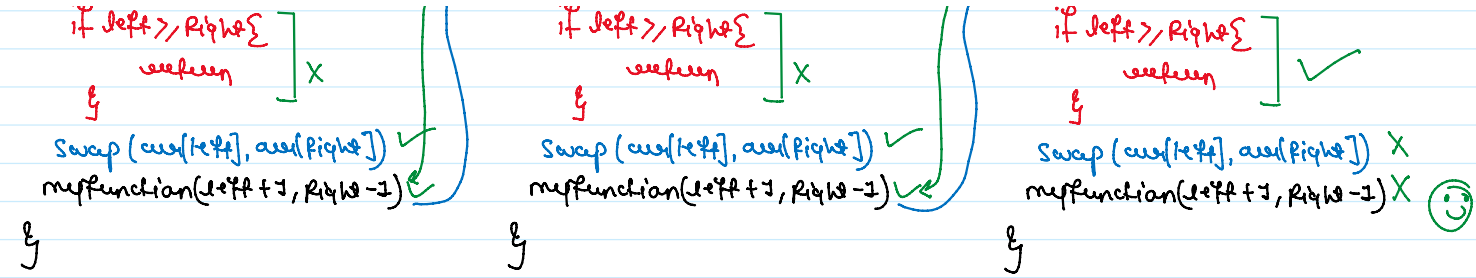
→ Recursion call ☺

Visualized that Recursion

arr = [1,2,3,4,5]
left = 0
Right = 4
myfunction(left, right){
if left >= Right{ return; } X

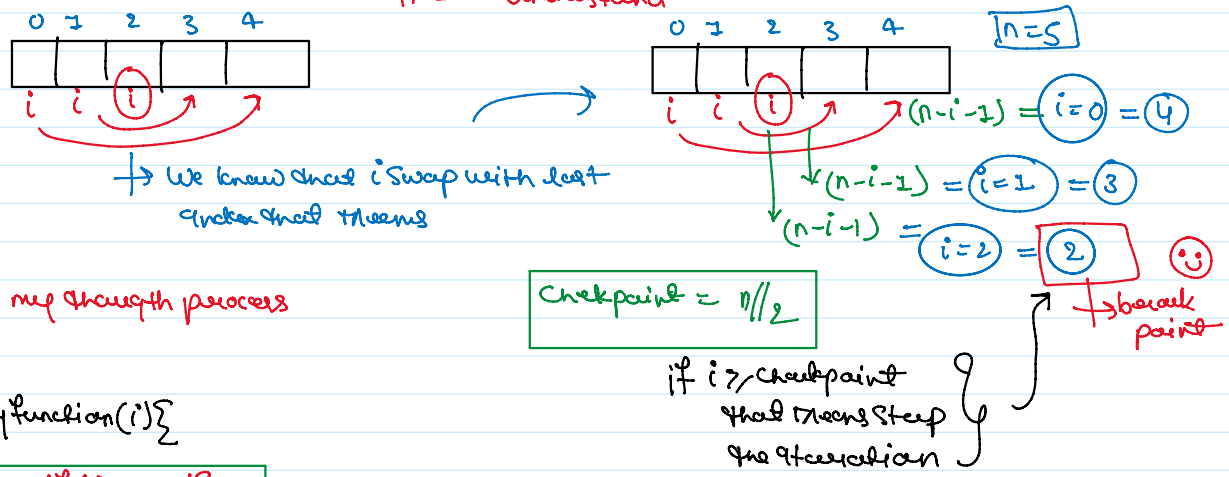
arr = [5,2,3,4,1]
left = ~~0~~ 1
Right = ~~4~~ 3
myfunction(left, right){
if left >= Right{ return; } X

arr = [5,4,3,2,1] → answer
left = ~~0~~ 2
Right = ~~4~~ 2
myfunction(left, right){
if left >= Right{ return; } ✓



Now that Approach used ② variable so that my Question is Can we Solved that problem using a Single Variable ??

Let's Understand



→ That is my thought process

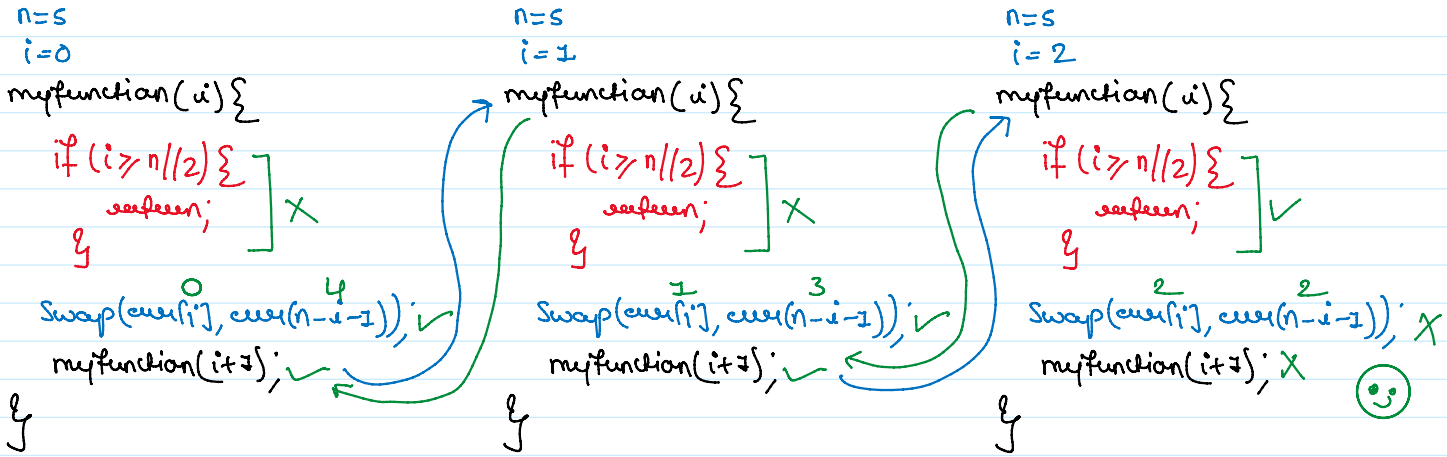
Checkpoint = n//2

if i >= checkpoint
that means Stop
the Operation

```
myfunction(i){
    if(i>=n//2){
        return;
    }
    Swap(arr[i], arr[i-1-1]);
    myfunction(i+1);
}
```

base case

Visualize that Recursion



Time Complexity = O(N)

→ Because linear Operation

→ Because linear Allocation

Space Complexity = O(N)

→ Recursion Stack Space

#Implementation Python

```python
# Problem 1: Reverse the Array

class ProblemsOnRecirsion:

    def swap(self,left, right,arr):
        arr[left], arr[right] = arr[right], arr[left]

    # using 2 pointer
    def reversetheArray1(self,left, right,arr):
        if left >= right:
            return

        self.swap(left,right-1,arr)
        self.reversetheArray1(left+1,right-1,arr)

        return arr

    # using 1 pointer
    def reversetheArray2(self,pointer,n,arr):

        if pointer >= (n//2):
            return

        self.swap(pointer,n-pointer-1,arr)

        self.reversetheArray2(pointer+1,n,arr)

        return arr

arr = [1,2,3,4,5,6,7]
n = len(arr)
ans = ProblemsOnRecirsion()
print(ans.reversetheArray1(0,n,arr))
```

using two pointer

using 1 pointer

#Implementation Java

```java
import java.util.Arrays;

public class L4_Problems_on_Functional_Recursion {
    public static void main(String[] args) {

        int[] arr = {1,2,3,4,5,6,7};
        int n = arr.length;

        ReverseArray(arr,0,n);
        System.out.println(Arrays.toString(arr));

        ReverseArray1(arr,0,n);
        System.out.println(Arrays.toString(arr));


    }

    public static void ReverseArray1(int[] arr, int pointer, int n) {

        if(pointer >= n/2){
            return;
        }

        swap(pointer,n-pointer-1,arr);
        ReverseArray1(arr,pointer+1,n);
    }

    public static void ReverseArray(int[] arr, int start, int end) {

        if(start >= end){
            return;
        }

        swap(start,end-1,arr);
        ReverseArray(arr,start+1,end-1);
}

    public static void swap(int start, int end, int[] arr) {

        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;

    }
    }
```

(2) Check Steeing is Palandreme Ore Nott

In this problem we check if the Steeing in palandreme ore nott using the recursion

Example a = 'MAM' Reverse 'MAM' ✓ (that is palandreme)

Example a = 'MAM' ⇌ (Reverse) 'MAM' ✓ (that is palandrome)

→ if we reversed the string then reversed string == to the Original string
that is palandrome string

→ Logic and concept are same that we learned previously similar concept that used here also

Thought Process

String = "MADAM"
(same, same)
→ that similar concept we used in previously Question

Used here we can pointer Approach for this problem ✓

TimeComplexity = O(N)
→ Linear Iteration

Space Complexity = O(N)
→ Recursion Stack Space

Implementation Python

```
class PalandromeorNot:

    def check(self,string,pointer,n):

        if pointer >= (n//2):
            return True

        if string[pointer] != string[n-pointer-1]:
            return False

        return self.check(string,pointer+1,n)

name = "mom"
ans =PalandromeorNot()
print(ans.check(name,0,len(name)))
```

Implementation Java

```
import java.util.Arrays;

public class L4_Problems_on_Functional_Recursion {
    public static void main(String[] args) {

        String name = "mom";
        if (PalandromeorNot(name,0,name.length()) == true){
            System.out.println("Given Stirng is Palandrome");
        }
        else {
            System.out.println("Not a Palandrome String");
        }


    }

    private static boolean PalandromeorNot(String name, int
pointer, int n) {

        if(pointer >= n/2){
            return true;
        }
        if(name.charAt(pointer) != name.charAt(n-pointer-1)){
            return false;
        }

        return PalandromeorNot(name,pointer+1,n);
    }
}
```