L6 Recursion on Subsequences

Saturday, December 17, 2022 5:00 PM

Now previously we learned the multiple Recursion but the question is where is the used of the concept of Multiple Recursion

→ The straightforward answer is Subsequence and Dynamic Programming

Subsecquance

Subsecquance Means follow the particular Order its can be Contagious or non-Contagious

Subarray = that is Contigeous Not Non-Contigeous ✓
Subsecquance = that is both Contigeous or Non-Contigeous ✓

Example We have Array and print all the Subsecquencies with Maintain the Order

arr = [3,1,2]

Important Point
☆ 2 1 3
→ that is also a Subsecquance but Not follow the Order of the Array

Subsecquance are
that are all follow the array order

3
1
2
3 1
1 2
→ Contigeous Subsequence

3 2 → Non-Contigeous Subsequence
3 1 2

→ Contigeous

{ } → Also that is the Subsecquance because we can not take any element from the array

How can we do this problem

[Total 8 Subsequence of this array possible]

→ Now this problem we have Very popular Algorithm that is know as Power Set (Using this Algorithm we find all the Subsecquance of the array)

✓ → But here we Solved the using the Concept of the Recursion

Let's Understand the Pattern

→ Now we Work on the principle of taking and Not taking because on every Index I have
(2) Option
→ Taking that Index
→ Not take that Index

Example

array		Subsecquance
[3 1 2]		
X X X	=	{ }
✓ X X	=	3
X ✓ X	=	1
X X ✓	=	2
✓ ✓ X	=	3 1
✓ X ✓	=	3 2
X ✓ ✓	=	1 2
✓ ✓ ✓	=	3 1 2

where
X → Not taking
✓ → taking

Now I Observed that every Index We have tuple of option take or Not take that is the Situation

3 1 2

Now I observed that every index we have tuple of option take or Not take that is the intuition behind this problem

```
myfunction(index, mylist){
    if(index >= n){
        print(mylist)
        return;
    }
    mylist.add(arr[index])
    myfunction(index+1, mylist);
    mylist.remove(arr[index])
    myfunction(index+1, mylist);
}
```

Base case

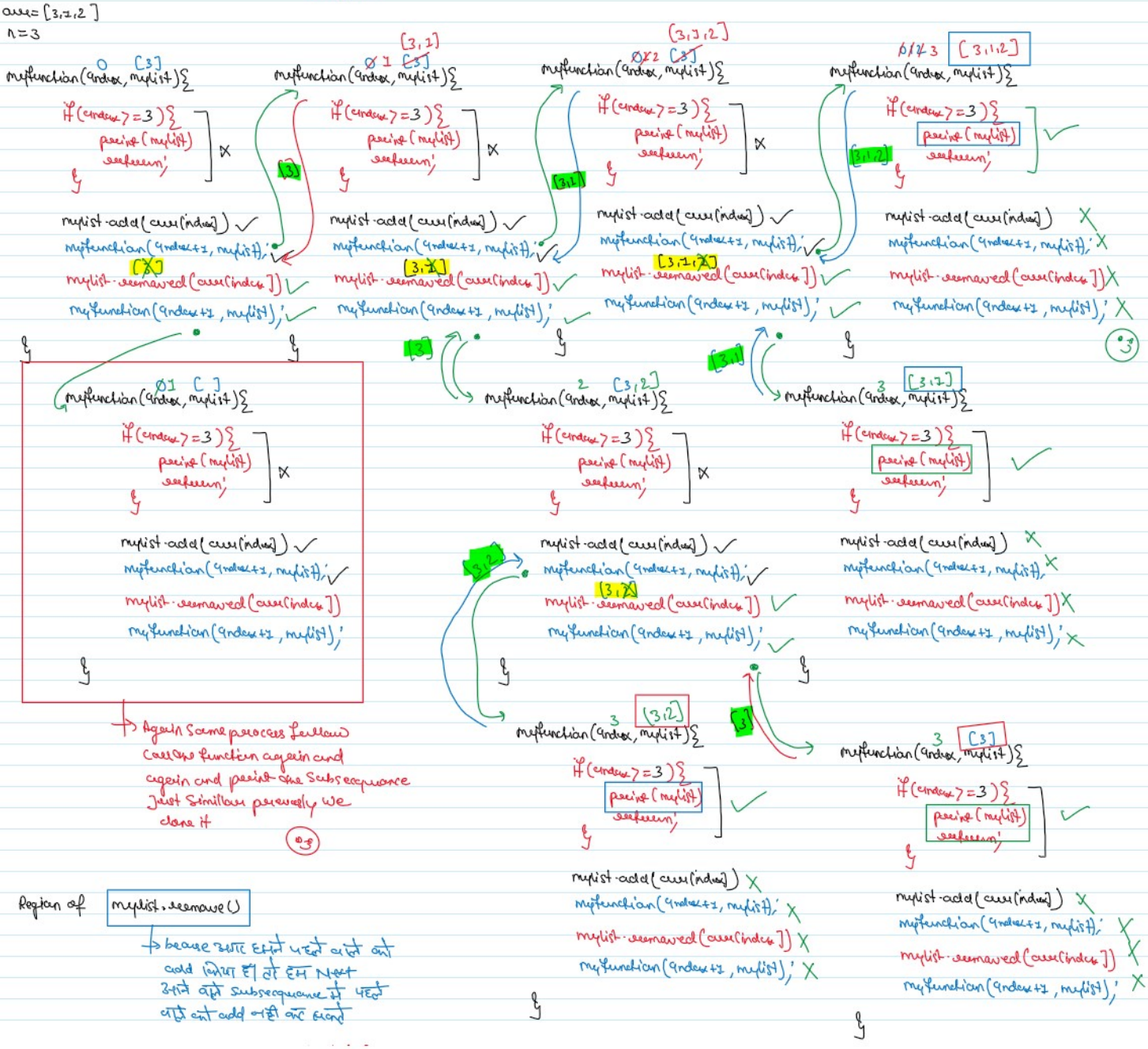
Add the element / Subsequence into the mylist

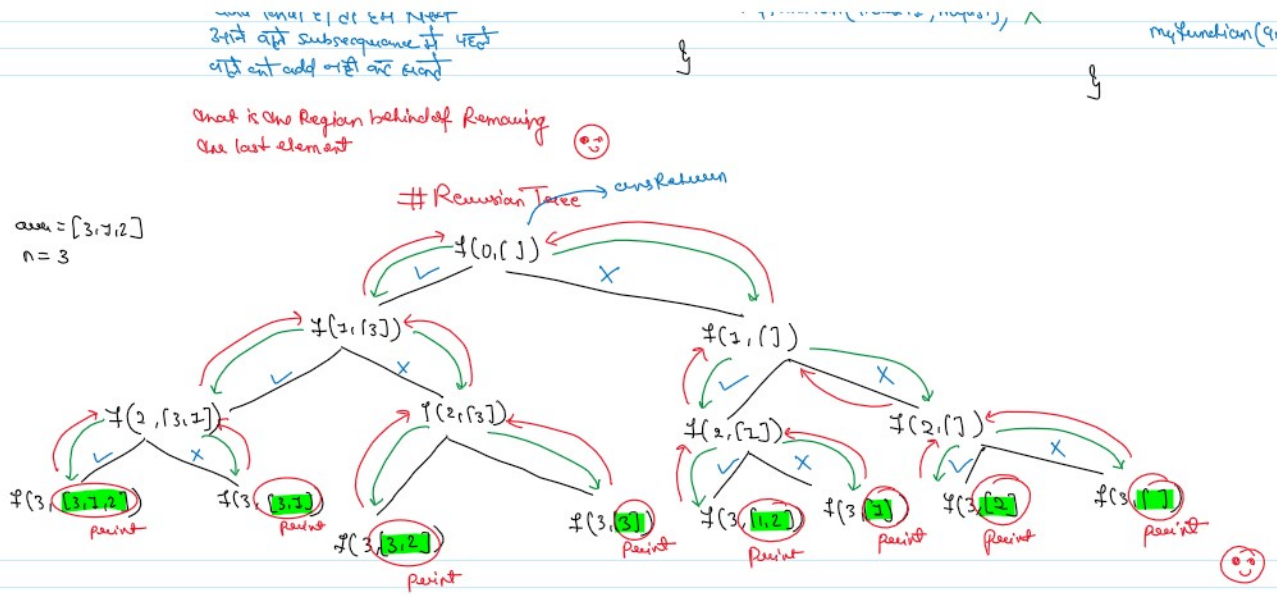
taking the case

Not taking case

#Let's understand the Sudo Code and Dry Run

arr = [3,1,2]
n = 3





Time Complexity = ?

→ for every recursion we have tuple of option take or not take = 2^n (Exponential)

$O(2^n)$

Space Complexity = $O(N)$

→ because we store one answer element that is any subsequence

#Implementation Python

```
class RecursionOnSubsecquence:

    def printAllSubsecquances(self,index,arr,myans):

        if index >= len(arr):
            print(myans)
            return

        myans.append(arr[index])
        self.printAllSubsecquances(index+1,arr,myans)
        myans.remove(arr[index])
        self.printAllSubsecquances(index+1,arr,myans)

    # reverse order
    def reverseOrd(self,index,arr,myans):

        if index >= len(arr):
            print(myans)
            return

        self.reverseOrd(index + 1, arr, myans)
        myans.append(arr[index])
        self.reverseOrd(index+1,arr,myans)
        myans.remove(arr[index])



ans = RecursionOnSubsecquence()
arr = [1,2,3,4,5,6,7,8,9,10]
print(ans.printAllSubsecquances(0,arr,[]))

print('Reverse')
print(ans.reverseOrd(0,arr,[]))
```

#Implementation Java

```
import java.util.*;

public class L6_Recursion_on_Subsequences {
    public static void main(String[] args) {

        int[] arr = {1,2,3};
        ArrayList<Integer> myans = new ArrayList<>();
        printAllSubsecquances(0,arr,myans);

        System.out.println("For Reverse Printing");
        reversePrint(0,arr,myans);

    }

//    for reverse printing
    private static void reversePrint(int index, int[] arr,
ArrayList<Integer> myans) {

        if(index == arr.length){
            System.out.println(myans);
            return;
        }

        reversePrint(index+1,arr,myans);
        myans.add(arr[index]);
        reversePrint(index+1,arr,myans);
        myans.remove(myans.size()-1);

    }

    private static void printAllSubsecquances(int index, int[]
arr, ArrayList<Integer> myans) {

        if(index == arr.length){
            System.out.println(myans);
            return;
        }

        myans.add(arr[index]);
        printAllSubsecquances(index+1,arr,myans);
        myans.remove(myans.size()-1);
        printAllSubsecquances(index+1,arr,myans);

    }
```

```
        printAllSubsecquances(index+1,arr,myans);

    }


}
```