10/10/2023
Tuesday

* **LEETCODE - 70 (climbing stairs)**

You are climbing a staircase. It takes n steps to reach the top. Each time you can climb 1 or 2 steps. Distinct ways to climb the top.

I/P = $n = 5$, $1 \leq n \leq 45$

O/P = 8

Explanation → There are eight ways to climb to the top.

1. 1 + 1 + 1 + 1 + 1
2. 1 + 1 + 1 + 2
3. 1 + 1 + 2 + 1
4. 1 + 2 + 1 + 1
5. 2 + 1 + 1 + 1
6. 2 + 2 + 1
7. 2 + 1 + 2
8. 1 + 2 + 2

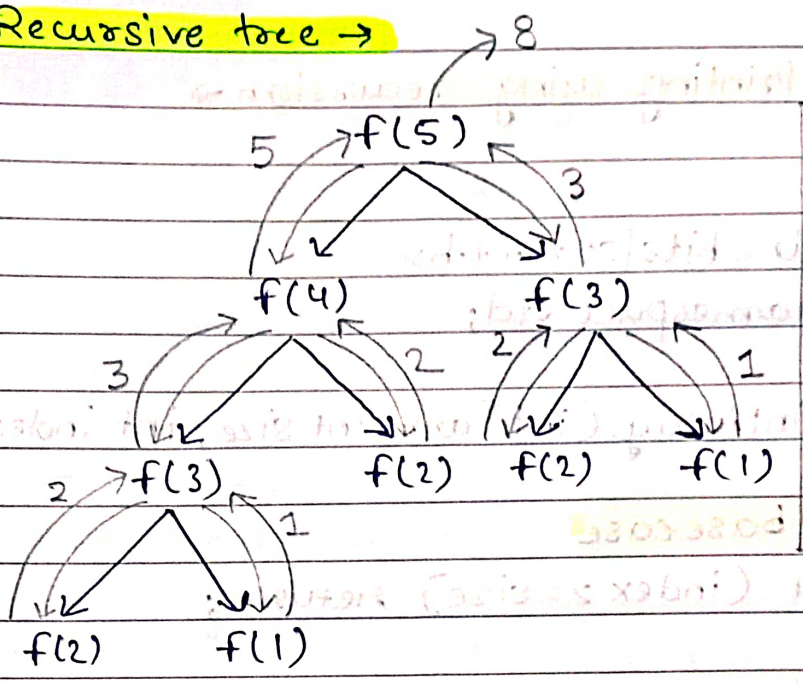
So, n^{th} step par jane k 2 ways hai. n^{th} step par pahuchne se pehle hum ya to $(n-1)^{th}$ step par honge ya fir $(n-2)^{th}$ step par honge because we can climb 1 or 2 steps at a time.

Recursive relation →

$$f(n) = f(n-1) + f(n-2)$$

Base case → when $n=1$, no. of ways = 1
when $n=2$, no. of ways = 2 i.e 1+1 & 2.

Recursive tree →




```
// base case
if(n==0||n==1)
{
    return n;
}
// recursive call
f(n)=f(n-1)+f(n-2)
```

Code →

```
#include <bits/stdc++.h>
using namespace std;

int climbStairs(int n){
    //base case
    if(n==1 || n==2) return n;

    // recursive call
    int ans= climbStairs(n-1)+climbStairs(n-2);
    return ans;
}

int main(){
    cout<<climbStairs(5)<<endl;
}
```

__/__/____

* Array Printing using recursion →

Code →
```
#include<bits/stdc++.h>
using namespace std;

void printArray(int* arr, int size, int index){
    //base case
    if(index >= size) return;

    //processing
    cout<<*(arr+index)<<" ";

    //recursive call
    printArray(arr, size, index+1);
}

int main(){
    int arr[] = {10,20,30,40,50};
    int size = 5;
    int index = 0;
    printArray(arr, size, index);
}
```

O/P → 10 20 30 40 50

* Search target in Array →

I/P → 10 20 30 40 50
target = 40

O/P → target found or not: 1

Code →
```
#include <bits/stdc++.h>
using namespace std;

bool findTarget (int arr[], int size, int target, int index)
{
    // base case
    if (index >= size) return 0;
    // agr target 1st index par mil jaye to usko base
    // case mai le liya kyuki jha par target mil jaye
    // vha par ek type se rukna hai
    if (arr[index] == target) return 1;

    // recursive call, rest array traversal recursion kardega
bool aagekaAns = findTarget (arr, size, target, index+1);
return aagekaAns;
}

int main(){
    int arr[] = {10, 20, 30, 40, 50};
    int target = 40;
    int index = 0;
    int size = 5;
    cout << "target found or not: " << findTarget
                                (arr, size, target, index);
}
```

__/__/____

* Find minimum in Array →

I/P → 40 30 10 50 20

O/P → 10

Code →

```
#include<bits/stdc++.h>
using namespace std;

void minimum(int* arr, int size, int index, int &mini){
    //base case
    if(index>=size) return;

    //processings
    mini = min(mini, arr[index]);

    //recursive call
    minimum(arr, size, ++index, mini);

}

int main(){
    int arr[] = {40, 30, 10, 50, 20};
    int size = 5;
    int index = 0;
    int mini = INT_MAX;
    minimum(arr, size, index, mini);
    cout << mini << endl;
}
```

__/__/____

* Find maximum in array →

I/P → 10 40 30 50 20

O/P → 50

Code →

```
#include<bits/stdc++.h>
using namespace std;

void maximum(int *arr, int size, int index, int &maxi){
    //base case
    if(index >= size) return;

    //processings
    maxi= max(maxi, arr[index]);

    //recursive call
    maximum(arr, size, ++index, maxi);
}

int main(){
    int arr[]={10, 40, 30, 50, 20};
    int size = 5;
    int index = 0;
    int maxi = INT_MIN;
    maximum(arr, size, index, maxi);
    cout << maxi <<endl;
}
```

__/__/____

* Find even in array →

I/P → 1 2 3 4 5 6

O/P → 2 4 6

Code →

```
#include<bits/stdc++.h>
using namespace std;

void findEven(int *arr, int size, int index, vector<int> &ans)
{
    //base case
    if(index >= size) return;

    //processing
    if((arr[index] & 1)==0)
        ans.push_back(arr[index]);

    //recursive call
    findEven(arr, size, index+1, ans);
}

int main(){
    int arr[]={1,2,3,4,5,6};
    int size=6;
    int index=0;
    vector<int> ans;
    findEven(arr, size, index, ans);
    for(auto num: ans){
        cout<<num<<" ";
    }
}
```

__/__/__

* Double the array →

I/P → 10 20 30 40 50

O/P → 20 40 60 80 100

Code →

```
#include <bits/stdc++.h>
using namespace std;

void doubleTheArray(int arr[], int size, int index){
    // base case
    if (index >= size) return;
    // processing
    arr[index] *= 2;
    // recursive call
    doubleTheArray(arr, size, index+1);
}

int main(){

    int arr[] = {10, 20, 30, 40, 50};
    int size = 5;
    int index = 0;
    doubleTheArray(arr, size, index);
    for(auto num: arr){
        cout << num << " ";
    }
}
```

__/__/____

* Find all occurence of target →

I/P → 40 30 10 20 10 10 , target = 10

O/P → indices where target found: 2 4 5

Code →

```
#include<bits/stdc++.h>
using namespace std;

void searchTarget(int *arr, int size, int index,
                  int target, vector<int> &ans){
    //base case
    if(index>=size) return;
    //processings
    if(arr[index]==target){
        ans.push_back(index);
    }
    //recursive call
    searchTarget(arr, size, index+1, target, ans);
}

int main(){
    int arr[]={40,30,10,20,10,10};
    int size=6;
    int index=0;
    int target=10;
    vector<int> ans;
    searchTarget(arr, size, index, target, ans);
    cout<<"indices where target found: ";
    for(auto num: ans){
        cout<<num<<" ";
    }
}
```

* Store digits of integer in vector →

I/P → num = 435267

O/P → digits of integer are: 4 3 5 2 6 7

Code →

```
#include <bits/stdc++.h>
using namespace std;

void digitsofInteger (int num, vector<int> &ans){

    //base case
    if (num==0) return;

    // recursive call
    digitsofInteger (num/10, ans);

    // processings
    ans.push_back (num%10);
}
int main(){
    int num= 435267;
    vector<int> ans;
    digitsofInteger (num, ans);
    cout<<" digits of integer are: ";

    for (auto num: ans){
        cout<<num<<" ";
    }
}
```

* Convert vector elements into integer →

I/P → vector<int> v {2, 9, 7, 4};

O/P → number is : 2974

Code →

```
#include<bits/stdc++.h>
using namespace std;

void printInteger(vector<int>v, int index, int &num){
        // base case.
        if(index >= v.size()) return;
        // processings
        num = num*10 + v[index];
        // recursive call
        printInteger(v, index+1, num);
}

int main(){
    vector<int>v{2, 9, 7, 4};
    int index = 0;
    int num = 0;
    printInteger(v, index, num);
    cout<<"number is : ";
    cout<<num<<endl;
}
```

* Print target character indices →

I/P → "babbabar" , target = 'b'

O/P → target indices are : 0 2 3

Code →

```
#include<bits/stdc++.h>
using namespace std;

void charIndex(string s, int index, char target,
               vector<int>&ans){
    //base case
    if(index >= s.length()) return;
    //processings
    if(s[index] == target){
        ans.push_back(index);
    //recursive call
    charIndex(s, index+1, target, ans);
}
int main(){
    string s = "babbabar";
    int index = 0;
    char target = 'b';
    vector<int> ans;
    charIndex(s, index, target, ans);
    cout<<"target indices are: ";
    for(auto num: ans){
        cout<<num<<" ";
    }
}
```