Note→ If I got stuck in any problem of recursion, I just have to do is, to use box method for smaller input or make recursive tree for it.

13/10/2023
Friday

* Check array is sorted or not →

I/P→ 10 20 30 60 40 50

O/P→ array is not sorted.

Code →

```
#include <bits/stdc++.h>
using namespace std;
bool checkSorted (int *arr, int size, int index){
        //base case, if true, means array is sorted
        if (index >= size) return 1;

if (arr[index] > arr[index-1]){
        //sorted hai
        //recursive call.
  bool aagekaAns = checkSorted(arr, size, index+1);
  return aagekaAns;
 }
else {
    return 0; //sorted nahi hai
 } }
int main(){
   int arr[] = {10, 20, 30, 60, 40, 50};
   int size = 6;
   int index = 1;
   bool isSorted = checkSorted (arr, size, index);
   if (isSorted){
        cout << "array is sorted" << endl;
      }
   else{
      cout << " array is not sorted. " << endl;
   } }
```

* Binary Search for finding target →

I/P → 10 20 30 40 50 60 , target = 40

O/P → target found at index : 3

Code →

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

int binarySearch(vector<int>v,int s, int e, int target){
        //base case
        if (s>e){
            return -1;
        }
//processing →that 1 case jo khud se solve krna hai
        int mid= s+(e-s)/2;
        if(v[mid]==target){
            return mid;
        }

// rest recursion will handle
    if (v[mid]>target){
     //left m jane k liye recursive call
    return binarySearch(v, s, mid-1, target);
}
    if (v[mid]<target){
     // right m jane k liye recursive call
    return binarySearch(v, mid+1, e, target);
}
}
```

```
int main(){
    vector<int>v{10, 20, 30, 40, 50, 60};
    int s=0;
    int e=v.size()-1;
    int target=40;
    int index = binarySearch (v, s, e, target);
    if (index == -1) {
        cout<<"target not found"<<endl;
    }
    else {
        cout <<"target found at index: "<<index<<endl;
    }
}
```

* Include and exclude pattern → Pattern 01

≔ Subsequences of string →

I/P → "dsa"

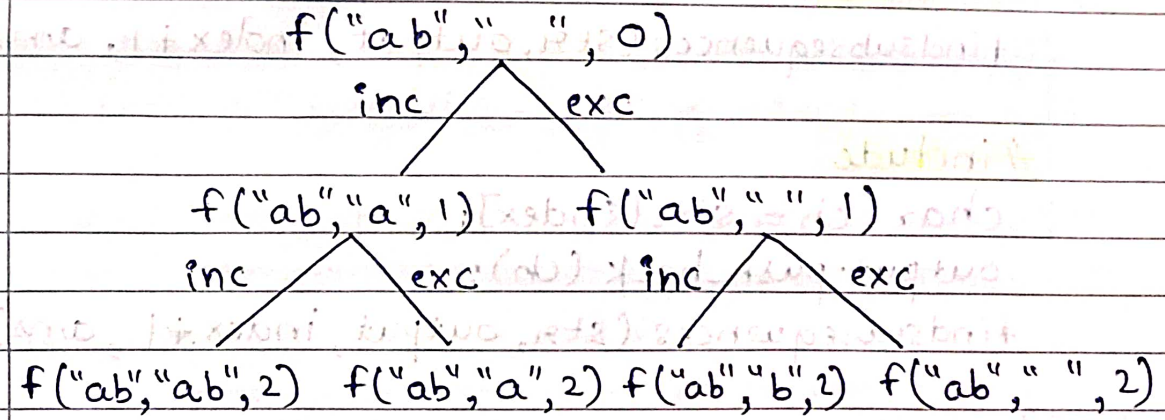
O/P → →
→ a
→ s
→ sa
→ d
→ da
→ ds
→ dsa

Condition → Order must remains the same as in input string.

Dry Run →

let string str = "ab"

a	b
0	1



Output string = "ab", "a", "b", " "

→ So, if input string has n characters, number of sub-sequences will be 2^n.

Code →

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

void findsubsequences (string str, string output,
                       int index, vector<string>&ans)
  {
    // base case
    if (index >= str.length()){
        ans.push_back(output);
        return;
    }
```

```
// recursive call

// exclude
findsubsequences(str, output, index+1, ans);

// include
char ch = str[index];
output.push_back(ch);
findsubsequences(str, output, index+1, ans);
}

int main(){
    string str = "dsa";
    string output = "";
    int index = 0;
    vector<string> ans;
    findsubsequences(str, output, index, ans);
    for(auto s: ans){
        cout << "-> " << s << endl;
    }
}
```

Pattern → Explore all possible ways

__/__/____

* Maximize the cut segments → gfg

Given an integer n denoting the length of a line segment. You need to cut the line segment in such a way that the cut length of a line segment each time is either x, y or z. After performing all the cut operations, your total number of cut segments must be maximum.
If no segment can be cut, then return 0.

I/P → n = 5, x = 5, y = 3, z = 2

O/P → 2

Explanation → We can maximize the cut segments by making two segments of length 3 and 2.

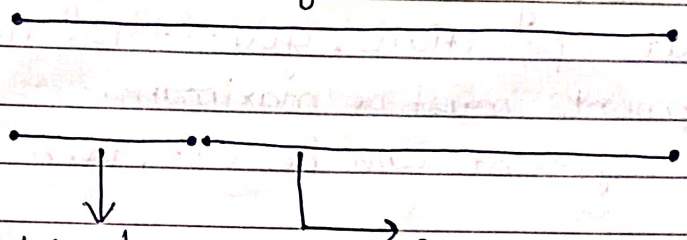
Logic →

```
//base case
if (n==0) return 0;
if (n<0) return INT_MIN;

//recursive call → jb cut length x hai
int optionX = 1+ maximizeTheCuts (n-x, x, y, z);

//recursive call → jb cut length y hai
int optionY = 1+ maximizeTheCuts (n-y, x, y, z);

//recursive call → jb cut length z hai
int optionZ = 1+ maximizeTheCuts (n-z, x, y, z);

return max (optionX, max (optionY, optionZ));
```

⇒ Jb n=0 means length of segment 0 ho gyi. That's cohy return 0 kiya.

⇒ Jb n<0 means length of segment negative ho gyi which is not possible. That's cohy invalid value return kr di INT_MIN.

Assume length of segment = n.

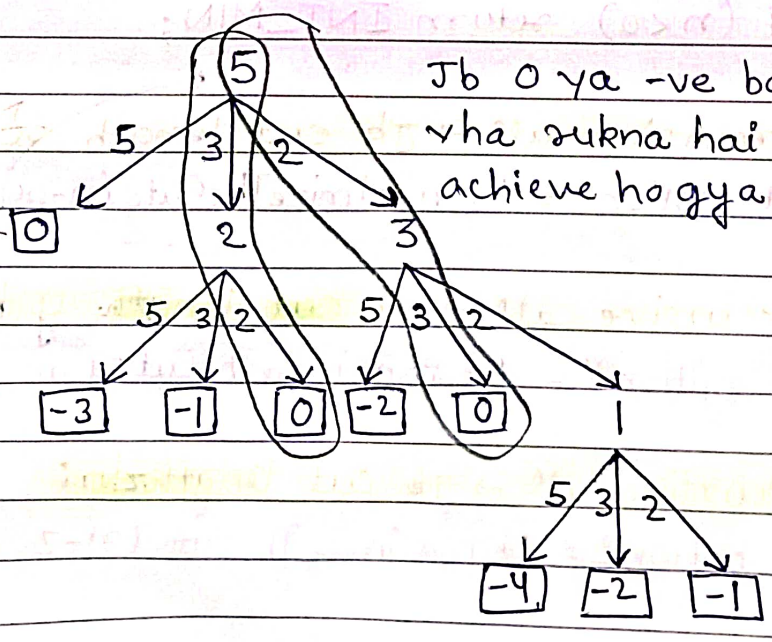


1 segment ye ho gya. Iski length x, y or z ho sakti hai. ye vo one case hai Jo hume khud solve karna hai

Rest of segments kitne banenge iska ans recursive call lake degi. Ye rest cases hai Jo recursive function call solve karegi.

So, total segments = 1 segment + rest segments

Dry run →



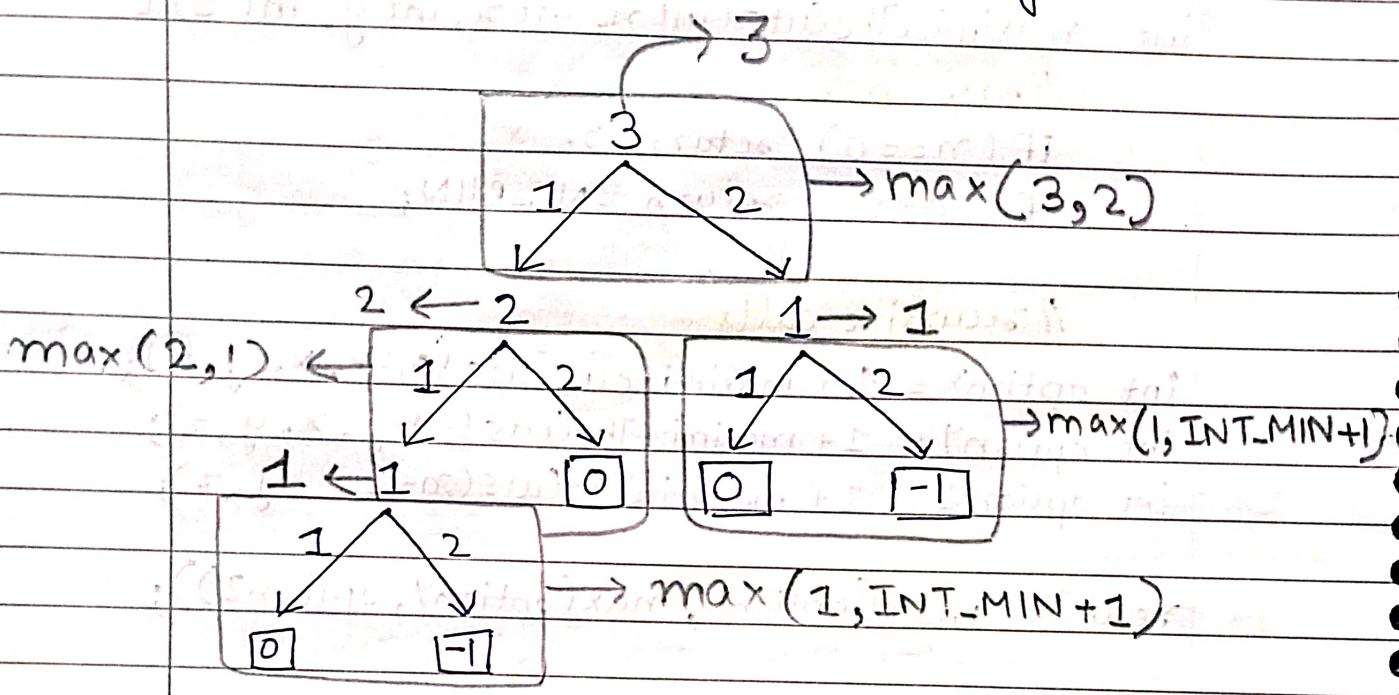
Jb 0 ya -ve ban gya to vha rukna hai base case achieve ho gya.

Code →

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

int maximizeTheCuts(int n, int x, int y, int z){
    //base case
    if(n==0) return 0;
    if(n<0) return INT_MIN;

    //recursive calls
int optionX = 1 + maximizeTheCuts(n-x, x, y, z);
int optionY = 1 + maximizeTheCuts(n-y, x, y, z);
int optionZ = 1 + maximizeTheCuts(n-z, x, y, z);

return max(optionX, max(optionY, optionZ));

}
int main(){
    int n= 5;
    int x = 5;
    int y = 3;
    int z = 2;
    int final_ans= maximizeTheCuts(n,x,y,z);
    if(final_ans<0){
        final_ans = 0;
    }
    cout<<final_ans<<endl;
}
```

Let for another test case, where →

$n = 3$ and $x = 1, y = 2$



For $n = 3$, $x = 1$ and $y = 2$,

Output = 3

Explanation → If we want to maximize the cut segments, we can do it by cutting the segment into three segment of length x i.e. 1 means 1+1+1.

Note → Har baar ans ko +1 krke isliye bhej rhe h kyki current target par pahuchne ke liye humne 1 segment x, y or z lengthkacut kiya hoga, bs usi 1 segment ko final ans me add krke return kr rahe hai.

= Explore all possible ways

* Coin Change (leetcode → 322)

You are given an integer array of coins representing coins of different denominations and an integer amount representing a total amount of money.
Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.
You may assume that you have infinite number of each kind of coin.

I/P : coins = [1, 2, 5], amount = 11

O/P : 3

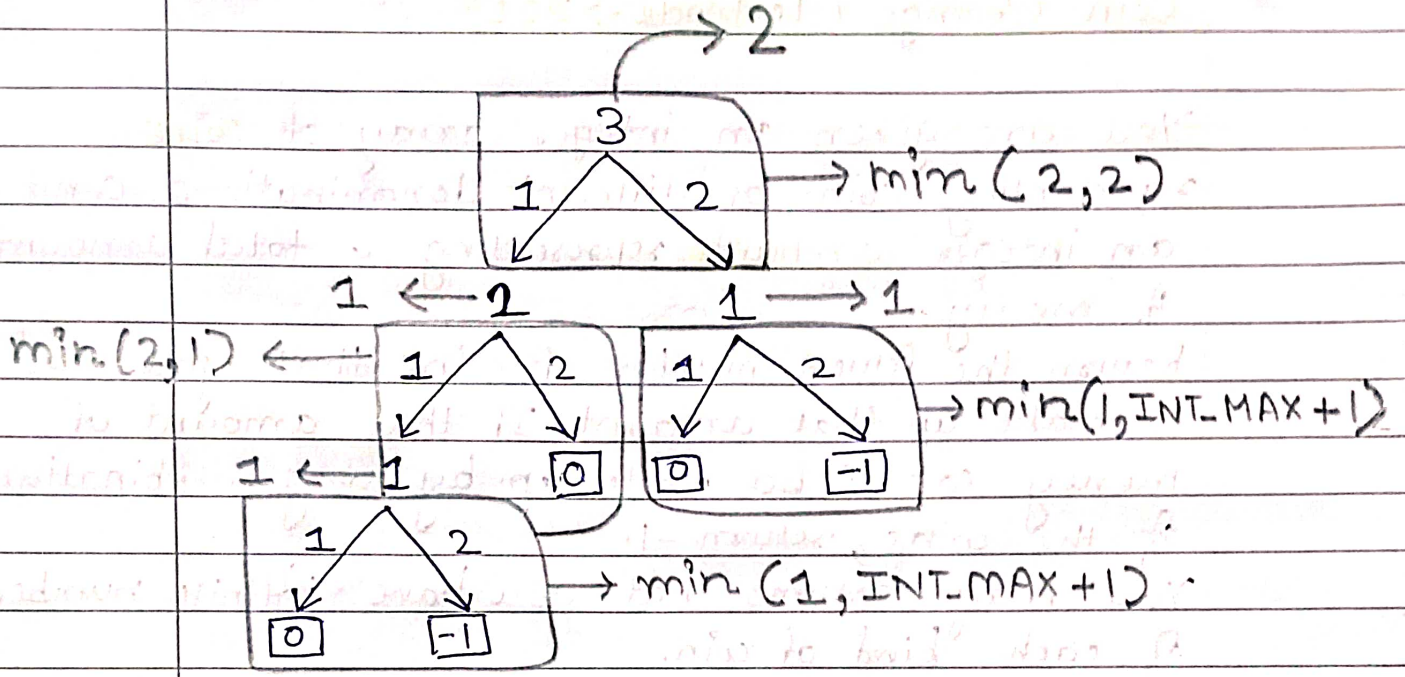
Explanation: minimum possible ways = 5+5+1 i.e. 3

Logic → • Jb amount = 0 ho jaye means target amount achieved. In that case, return 0.

- Jb amount < 0 i.e. negative ho jaye which is invalid. So, in that case, invalid answer return krdo INT_MAX.

- Ek loop chalado jo coins array par traverse karegi. Ek case solve krdo, rest recursion will handle out.

Dry Run: let coins = [1, 2] and amount = 3



for coins = [1, 2] and amount = 3,

Output = 2

Explanation → 3 = 1 + 2 i.e. coins used = 2

Note → Har baar ans mai +1 krke isliye bhej rahe hai kyuki current target amount ko achieve karne ke liye humne coins array mai se 1 coin use kiya hoga chahe ver of denomination it may be, bs usi 1 coin ko final ans mai add krke return kar rahe hai.

Code →

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

int coinChange(vector<int>&coins, int amount){

        //base case
        if(amount==0) return 0;
        if(amount<0) return INT_MAX;

    int mini = INT_MAX;
    for(auto currCoin : coins){      //recursive call →
        int ans = coinChange(coins, amount-currCoin);
        if(ans!=INT_MAX)
        mini= min(mini, ans+1);
    }
    return mini;
}

int main(){
    vector<int> coins{1,2,5};
    int amount=11;
    int finalAns = coinChange(coins, amount);
    if(finalAns==INT_MAX){
            finalAns = -1;
        }
    cout<<finalAns<<endl;
}
```

__/__/____

* **Maximum Sum of non-adjacent elements or House Robber (leetcode-198)**

You are a professional robber planning to rob houses along a street. Each house has a certain amount stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically connect the police if two adjacent houses were broken into on the same night.
Given an integer array **nums** representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.

I/P : nums = [2, 9, 7, 4]

O/P : 13

Explanation: Rob house 1 (money = 2)
Rob house 3 (money = 7)
Total amount you can rob = 2 + 7 = 9

Rob house 2 (money = 9)
Rob house 4 (money = 4)
Total amount you can rob = 9 + 4 = 13

So, maximum amount you can rob = 13

Code→

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

int rob(vector<int> &nums, int size, int index){

    //base case
    if (index >= size) return 0;

// recursive calls →
// chori karlo → ith index par
int option1 = nums[index] + rob(nums, size, index+2);
//index+2 isliye kiya kyuki currently, jis index par hai
// i.e. ith index uske adjacent index par chori nahi
// karni hai i.e. (i+1)th index par.

// chori mat karo → ith index par
int option2 = 0 + rob(nums, size, index+1);
// index+1 because ith index par chori nahi ki to
// ab (i+1)th index par chori kar sakte hai

return max(option1, option2);
}
int main(){
    vector<int> nums{2,9,7,4};
    int size = nums.size();
    int index = 0;
    int ans= rob(nums, size, index);
    cout << ans << endl;
}
```