15/mar/23

Recursion - 04

Infinite

Q1 → i/p → | 1 | 2 | 3 | , size = n ,

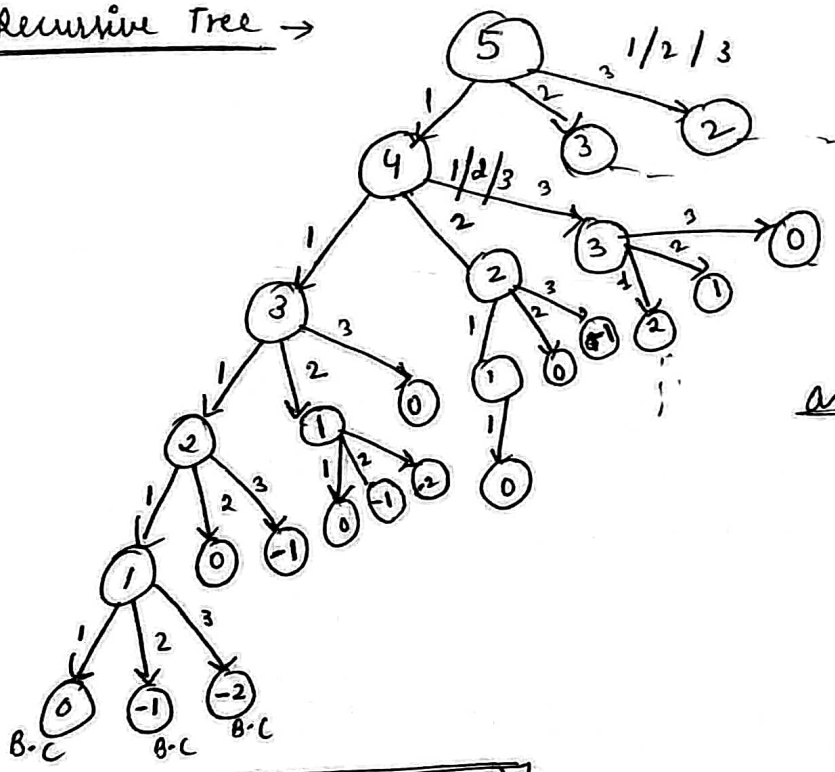
n distinct (unique) elements.

target sum = 5

you have to tell the minimum number of elements required to reach sum. (same as coin change)

5 → {1, 1, 1, 1, 1}
→ {1, 1, 1, 2}
→ {1, 2, 2}
→ {1, 1, 3}
→ {2, 3} ← min

Recursive Tree →

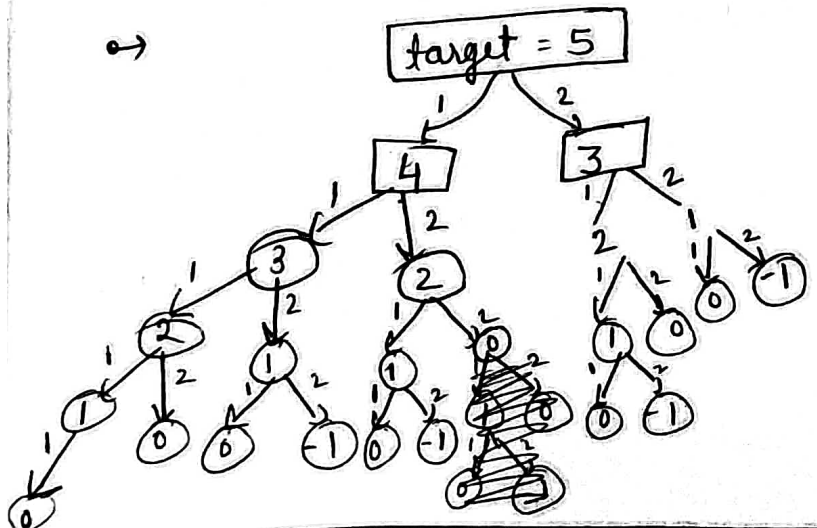


and so on.

Two Base Cases (=0, <0)

0 ← target reached

→



arr | 1 | 2 | , target = 5

{1, 1, 1, 1, 1}
{1, 1, 1, 2}
{1, 2, 2} ✓ min.

Base cases :- mini = INT_MAX

Sum (target) < 0 → Invalid ~~valid~~ value (-ve value nhi bna skte).

↳ means mini update nhi ~~s~~ honi chahiye.

sum == 0 or target → no. Ban gya.

↳ ~~update mini~~ return 0

Code -

```
int solve (vector<int>& arr, int target){
    if (target == 0)
        return 0;
    if (target < 0)
        return INT_MAX;
    int mini = INT_MAX;
    har ek array ke liye call krenge.
    for (int i = 0; i < arr.size(); i++){
        int ans = solve (arr, target - arr[i]);
        mini = min (mini, ans);
    }
    /* we just ~~p~~ include a element
    (means we took one element to make
    the target), so we will add 1 in ans.
        mini = min (mini, ans+1);
    now in this if ans is INT_MAX then we are
    adding 1 in this so this is wrong.
    & we will only update mini when ans is not
    INT_MAX. */
    ⇒   if (ans != INT_MAX)
            mini = min (mini, ans+1);
    }
    return ans;
}
```

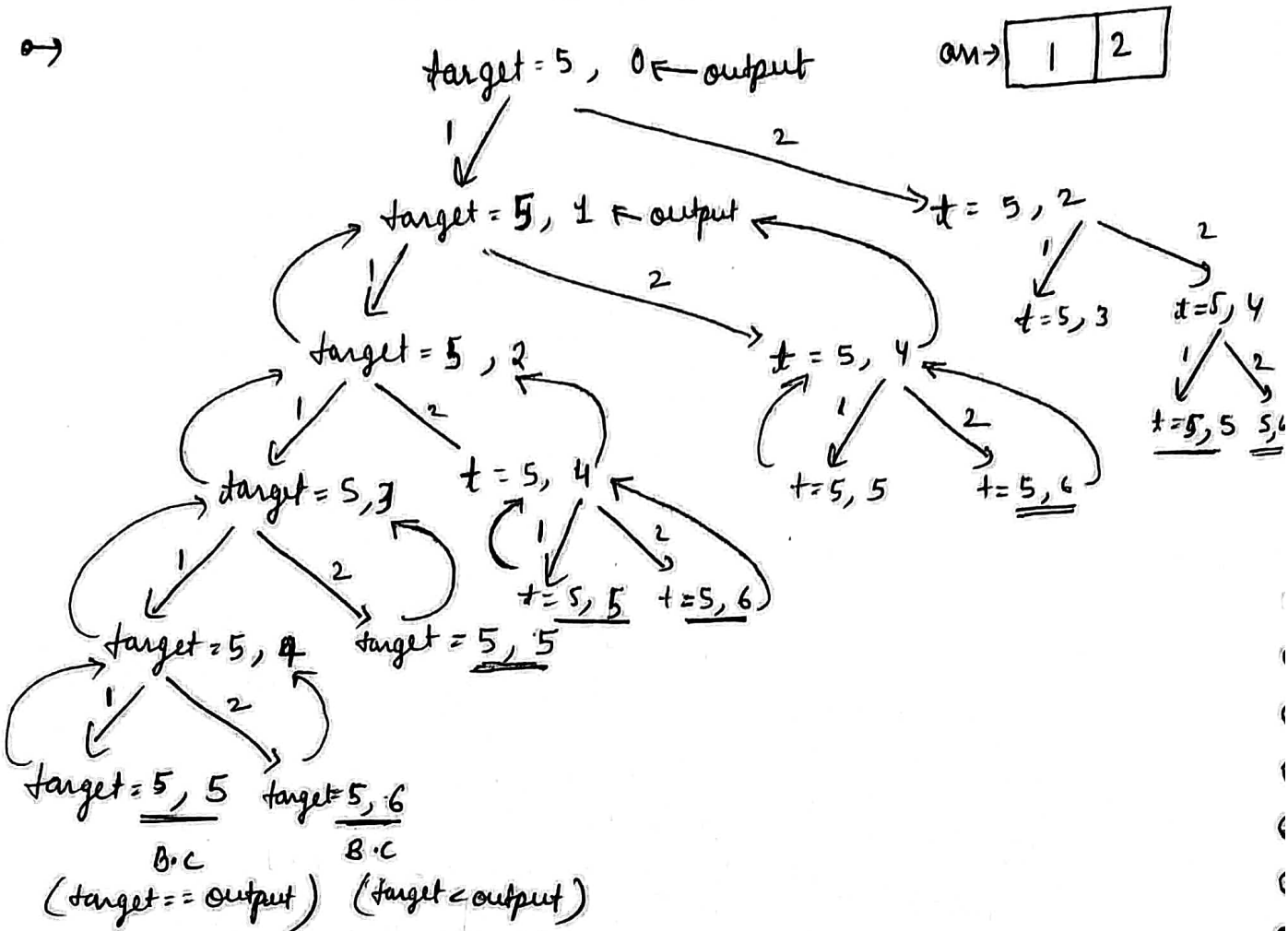
0 bnane ke liye, ek bhi element nhi lagega.

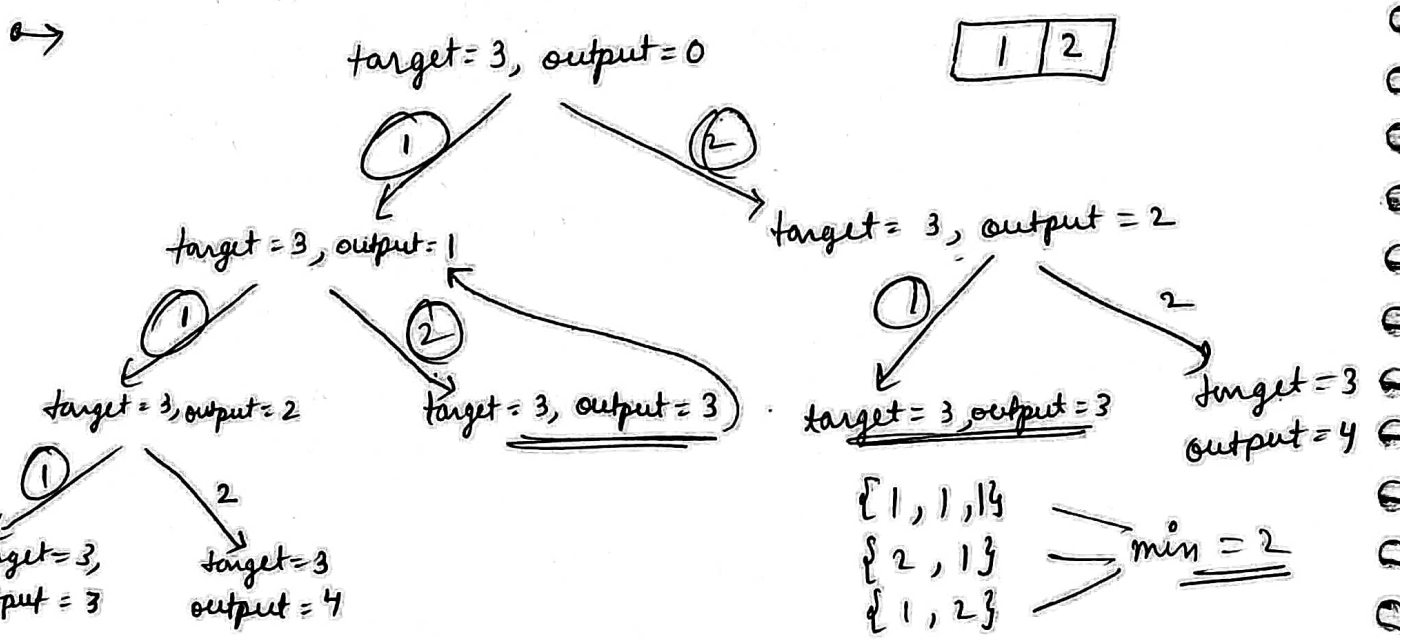
-ve hum bna nhi skte isiliye kuch aisa return krenge ki mini update hi na ho.

~~mini = min (mini, ans);~~ now this line is wrong.

→ another way to solve this problem is to start from 0. (with the help of an additional variable).

eg → target = 5 arr | 1 | 2 |

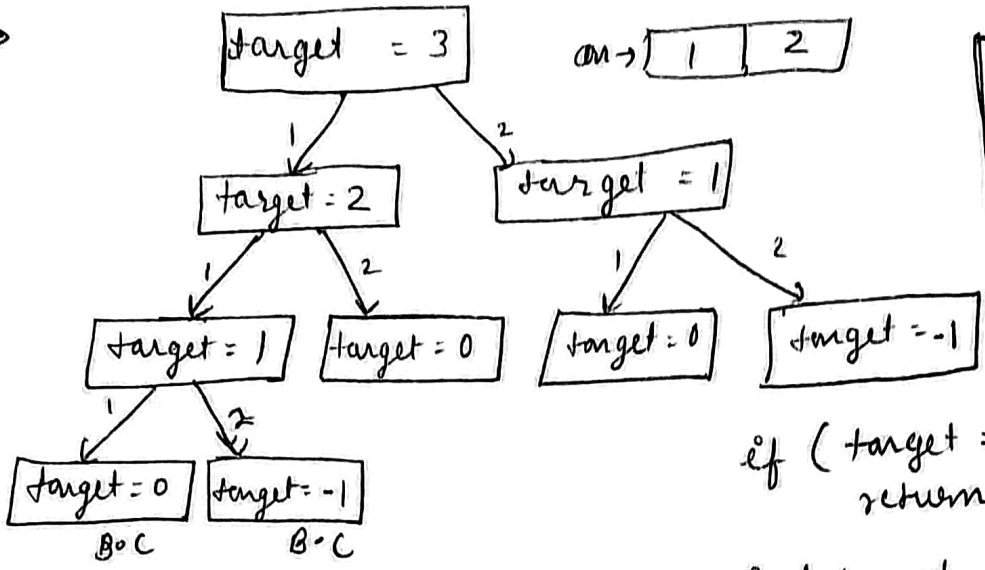




Base Cases -

```
if (target == output)
        return 0.

if (target < output)
        return INT_MAX;
```

Task → Code this method.



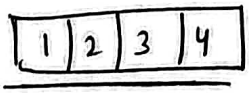
Solve func tells minimum no. of elements to make target.

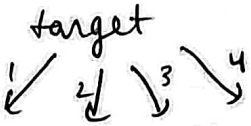
```
if (target == 0)
    return 0;

if (target < 0)
    return INT_MAX;
```

target = 0 , we have target as 0 , so to make 0 , we don't have to add / subtract anything. That's why return 0.

And when target < 0 , that means it is not even possible to make that target , ~~Also~~ it's not possible, so that's why we are returning INT_MAX. so that ~~if~~ it get never included in answer.

Now why for loop?





For Each target value, recursion call is called for each element of the array.

```
for (int i = 0; i < n; i++){
    int ans = solve (arr, target - arr[i]);
```

ans → valid / Invalid (Invalid when ans = INT_MAX)

```
    if (ans != INT_MAX)
        mini = min (mini, ans + 1);
```

+1 because we just included the current no. (element), And we included current no, that means it has been used to make target.

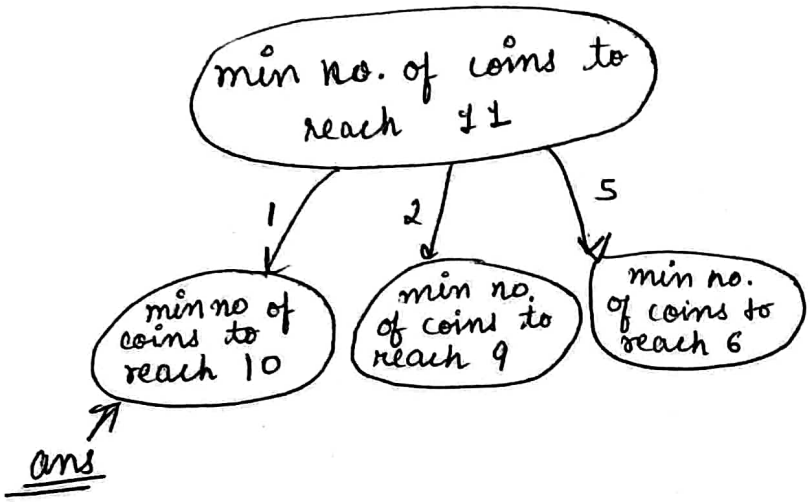
ans → valid → update mini
ans → Invalid → Ignore

let's understand ans + 1 →

target = 11

i/p → arr → | 1 | 2 | 5 |

infinite supply of coins.

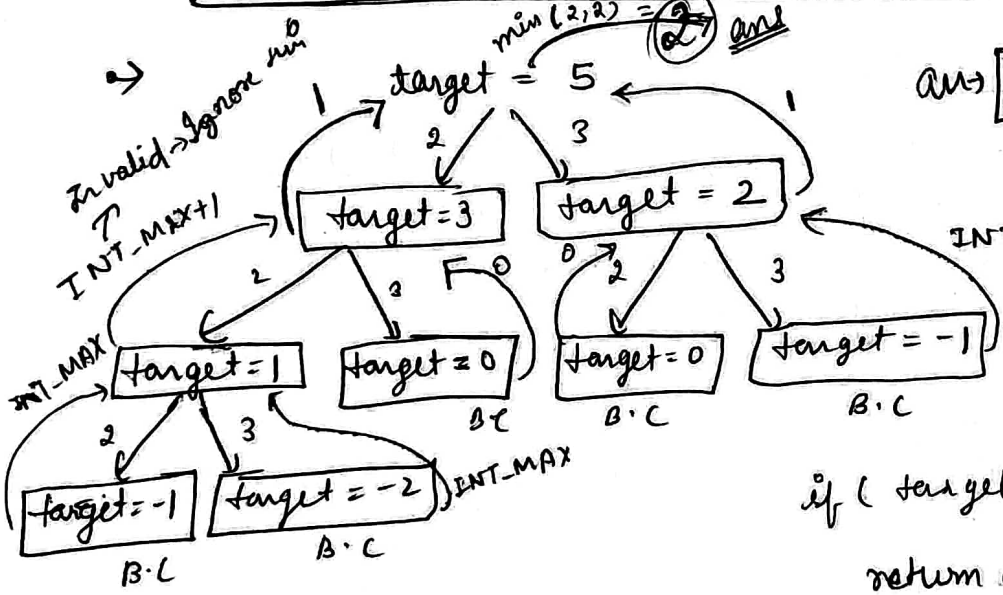


Why we choose INT_MAX?
→ because it's a safe value (finding minimum value)

⇒ min no. of coins to reach 11 ⇒ ans + 1 ← because 1 coin we already choosed.

or min no. of coins to reach 11 ⇒ min
- 1 coin of 1 Rs + min coin to 10 Rs
- 1 coin of 2 Rs + min coin to 9 Rs
- 1 coin of 5 Rs + min coin to 6 Rs.

→



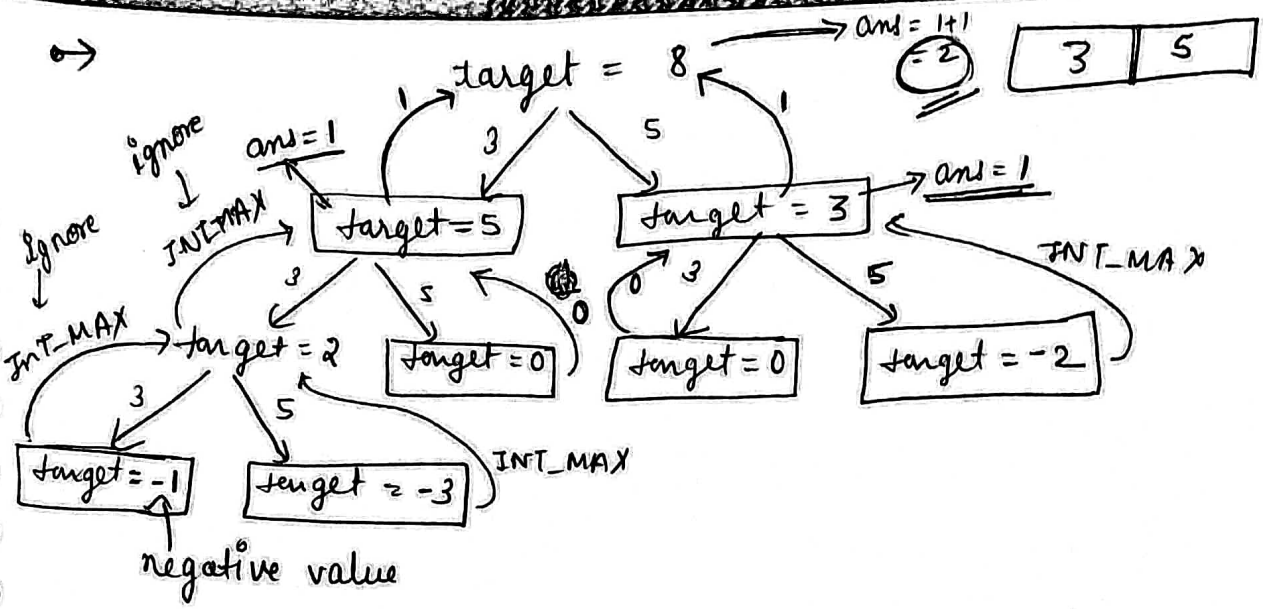
ans → | 2 | 3 |

if (target == 0)
return 0;
← 0 बनाने में 0 coin लगेंगे।

else if (target < 0)
return INT_MAX;
← कितने भी coin लगा लो हम नहीं बना सकते।

min no. to coins to create 0 ⇒ 0

min no. of coins to reach 3 = 0 + 1 = 1



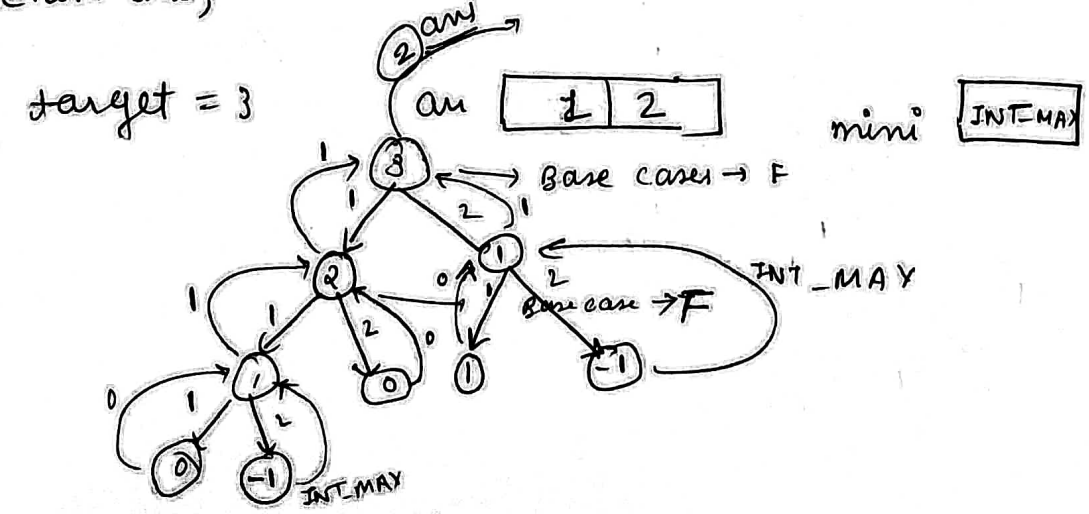
target 5 = 1 5Rs coin + solve (target = 0)

= 1 + 0 = 1

target 3 = 1 3Rs coin + solve (target = 0)

= 1 + 0 = 1

```
int solve (arr, target)
{
    if (target == 0)
        return 0;
    if (target < 0)
        return INT_MAX;
    int mini = INT_MAX;
    for (int i = 0; i < n; i++){
        int ans = solve (arr, target - arr[i]);
        if (ans != INT_MAX) {
            mini = min(mini, ans+1);
        }
    }
    return ans;
}
```



⊙ We learned two Patterns →

① Include - Exclude

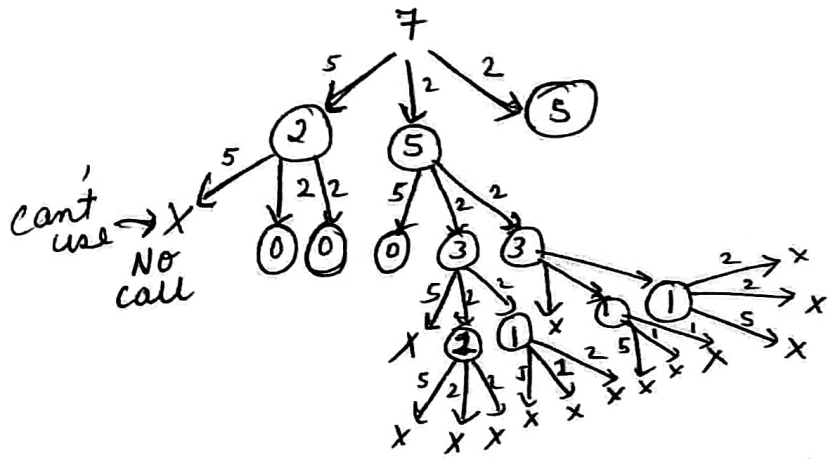
② for loop → Iterate on each element

Ques2- Cut into segments →

i/p → N ← rod length

find max no of segments you can make of this rod provided x, y and z.

eg → N = 7, x = 5, y = 2, z = 2,



2, 5
2, 5

// solve function returns max. no. of segments →

```
int solve (int n, int x, int y, int z){
    // base case
    if (n == 0)
        return 0;
```

→ If the length of a rod is 0, then we can't even cut it.

```
    int a = INT_MIN;
    if (n-x >=0){
        ans = solve (n-x, x, y, z) + 1;
    }
```

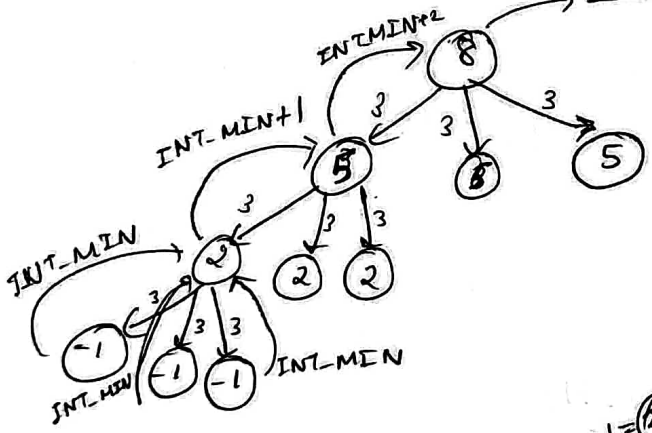
← This solve func is giving ans of n-x so if we have to find ans of n then we will add 1.

```
    int b = INT_MIN;
    if (n-y >=0){
        b = solve (n-y, x, y, z) + 1;
    }
    int c = INT_MIN;
    if (n-z >=0){
        c = solve (n-z, x, y, z) + 1;
    }
    int ans = max (a, max (b, c));
    return ans;
}
```

let's make a simple code →

```
int solve(int n, int x, int y, int z){
    if(n==0){
        return 0;
    }
    if(n<0){
        return INT_MIN;
    }
    int ans1 = solve(n-x, x, y, z) + 1;
    int ans2 = solve(n-y, x, y, z) + 1;
    int ans3 = solve(n-z, x, y, z) + 1;
    int ans = max(ans1 + max(ans2 + ans3));
    return ans;
}
int main(){
    int n,x,y,z;
    cin>>n>>x>>y>>z;
    int ans = solve(n, x, y, z);
    if(ans<0) ans = 0;
    cout << ans;
}
```

→

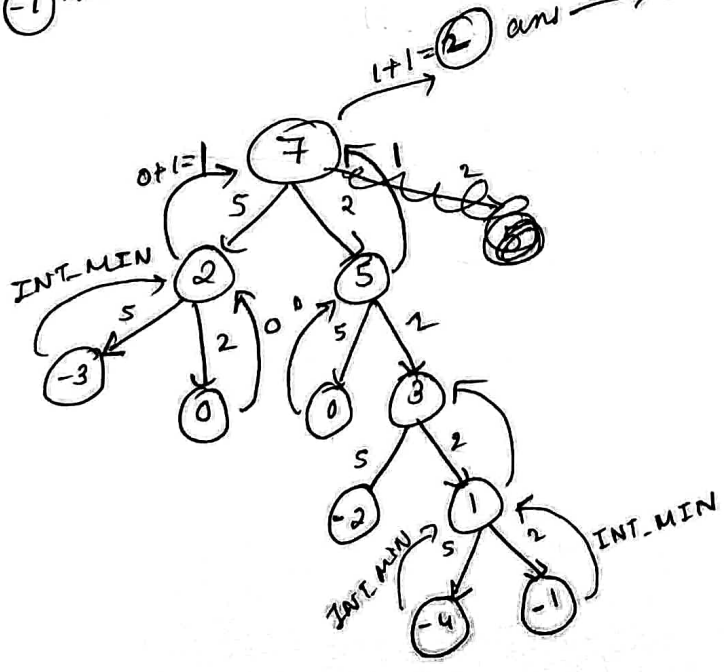


ans = INT_MIN+3 → main()

```
if (ans<0)
    ans=0
cout << ans;
```

→ 0

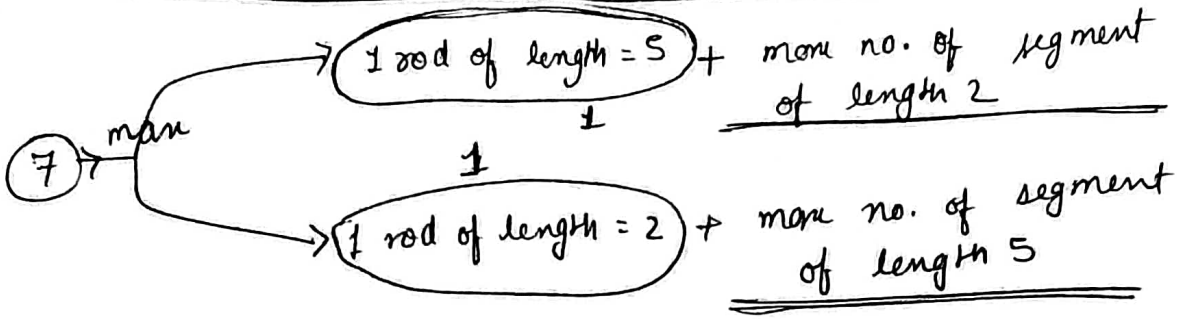
→ n = 7
x = 5
y = 2
z = 2



ans → main(){

```
if(ans<0) ans=0;
cout << ans;
```

→ 2



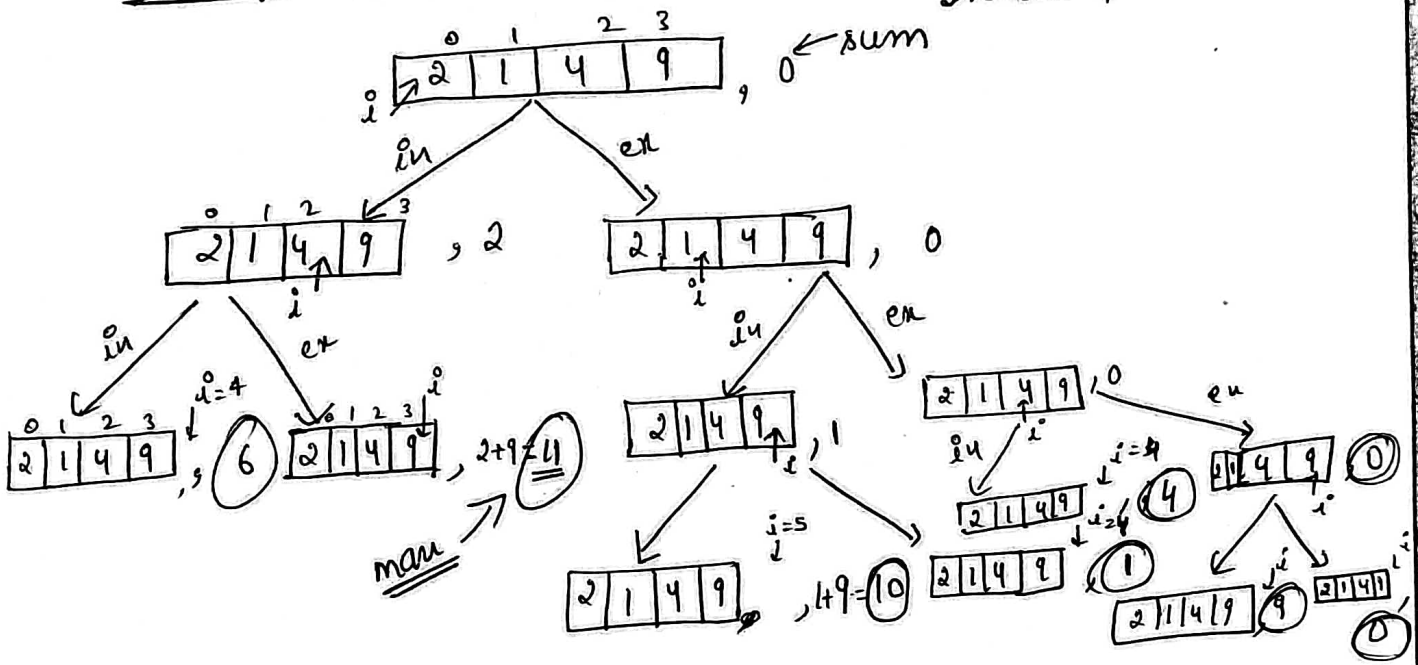
Ques 3 → Max Sum of non-adjacent elements →

i/p → | 2 | 1 | 4 | 9 | , Return the maximum sum of subsequence in which no. two elements are adjacent.

o/p → 11 (2+9) (Same as House Robber-1) LH·W

Subsequence → some elements is present some are not.

Include / Exclude



Code →

```
void solve(vector<int>& arr, int i, int sum, int& maxi){
    if (i > arr.size()){
        maxi = max(sum, maxi);
        return;
    }
    solve(arr, i+2, sum+arr[i], maxi);   ← Include
    solve(arr, i+1, sum, maxi);          ← Exclude
}
int main(){
    vector<int> arr {2,1,4,9};
    maxi = INT_MIN; i = 0; sum = 0;
    solve(arr, i, sum, maxi);
    cout << maxi;
}
```