

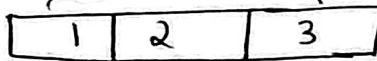
15/mar/23

infinite

Recursion - 04

Q1 →

i/p →



, size = n ,

n distinct (unique) elements.

target = 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}

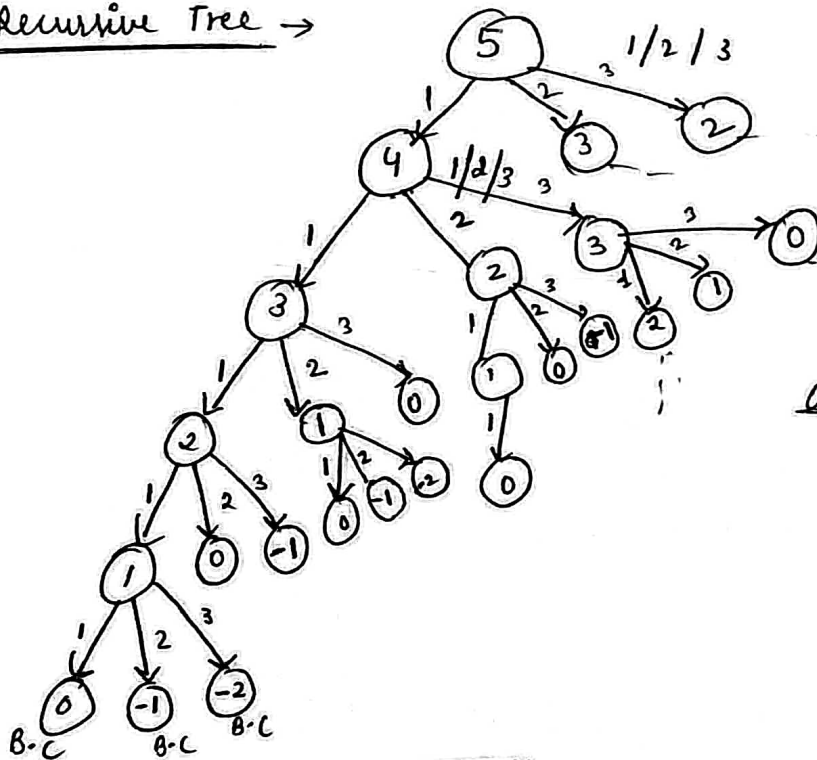
→ {1, 2, 2}

→ {1, 1, 3}

→ {2, 3}

min

Recursive Tree →



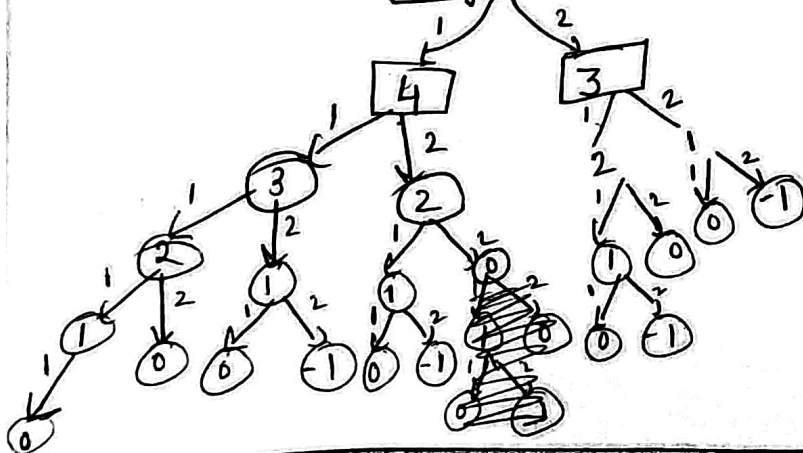
and so on.

Two Base Cases (= 0, < 0)

0 ← target reached

→

target = 5



ans [1 | 2] , target = 5

{1, 1, 1, 1, 1}

{1, 1, 1, 2}

{1, 2, 2} ✓ min.

Base cases - $mini = INT_MAX$
 $(sum < 0 \rightarrow \text{Invalid value } (-ve \text{ value nhi bna skte}).$
 \rightarrow means $mini$ update nhi \neq honi chahiye.

$sum == 0 \rightarrow$ no. Ban gya.
 \rightarrow ~~update mini~~ return 0

code -

```
int solve (vector<int> &arr, int target){
```

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

0 banane ke liye, ek bhi element nhi lagega.

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

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

```
    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);
```

now this line is wrong.

/* we just include a element (means we took one element to make the target), so we will add 1 in ans.

```
        mini = min(mini, ans);
```

now in this if ans is INT_MAX then we are adding 1 in this so this is wrong.

so we will only update mini when ans is not INT_MAX. */

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

```
    }
    return ans;
```

```
}
```

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

eg \rightarrow target = 5

arr

1	2
---	---

target = 5, 0 ← output

target = 5, 1 \leftarrow output

target = 5, 2

target = 5, 3

target = 5, 4

target = 5, 5

target = 5, 6

t = 5, 2

t = 5, 3

t = 5, 4

t = 5, 5

t = 5, 6

B.C (target == output)

B.C (target < output)

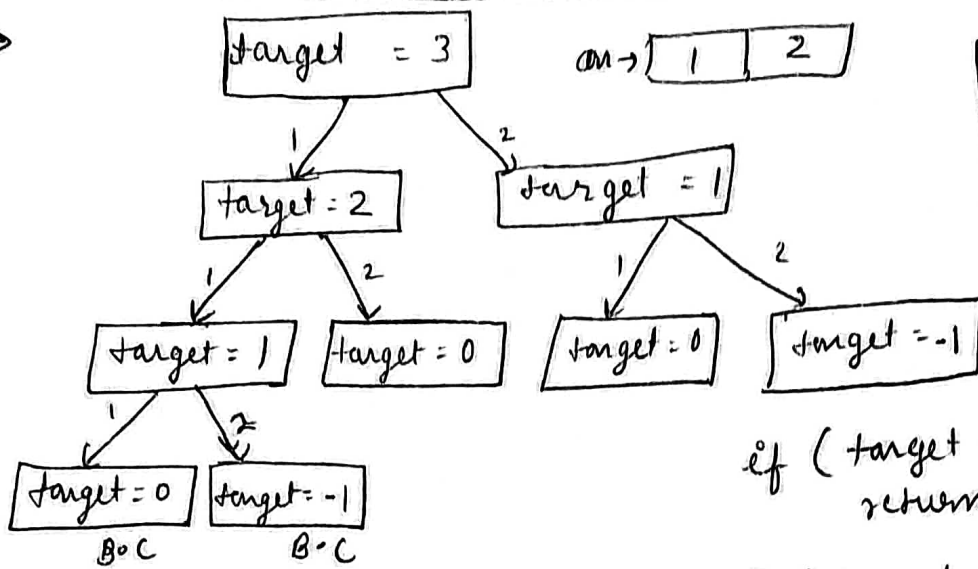
target = 3, output = 0

[illegible]

def (target == output)
return 0.

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

Task \rightarrow Code this method.



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

if (target == 0)
return 0;

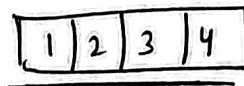
if (target < 0)
return INT_MAX;

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 ~~it~~ 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 (an, target - an[i]);

valid

Invalid

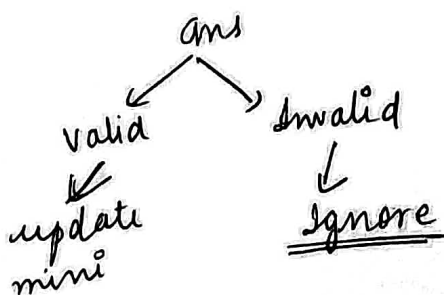
when ans = INT_MAX

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 ~~get~~ make target.



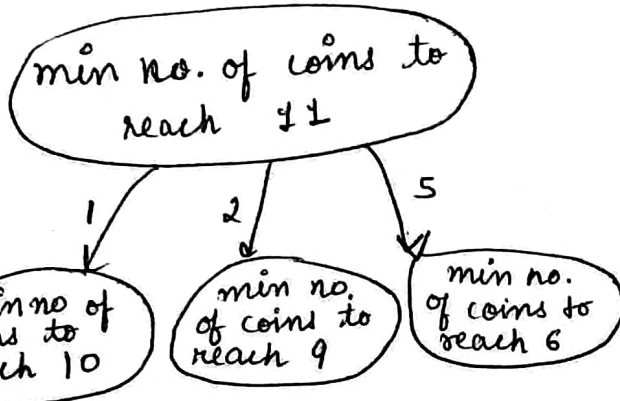
let's understand $ans + 1 \rightarrow$

target = 11

i/o \rightarrow

1	2	5
---	---	---

infinite supply of coins.

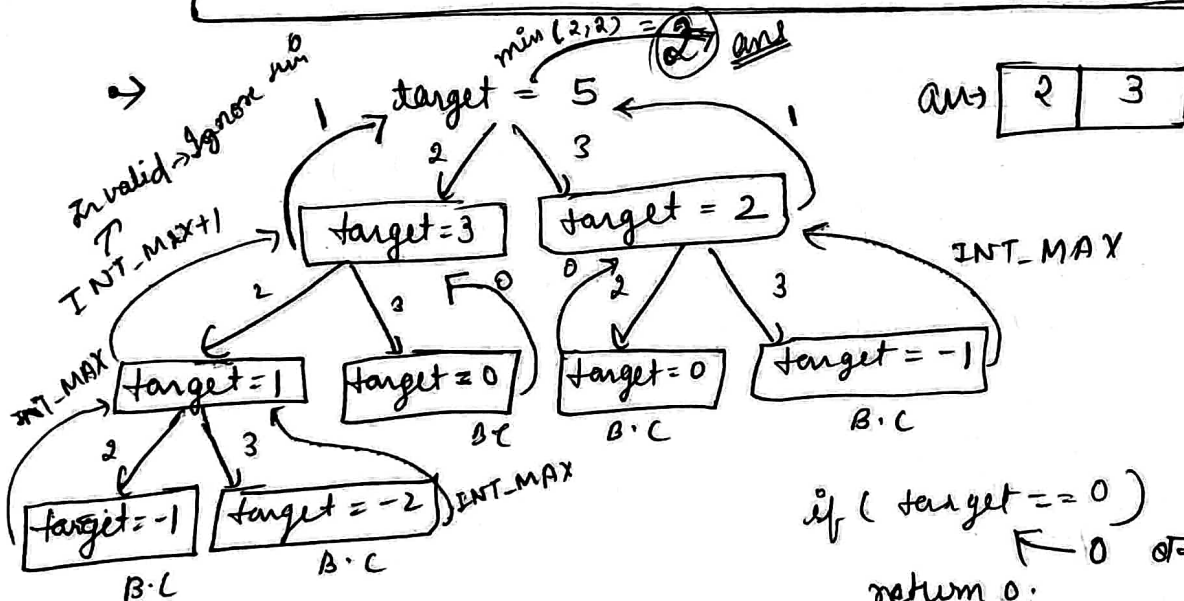


ans

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

\Rightarrow min no. of coins to reach 11 \Rightarrow ans + 1 \leftarrow because 1 coin we already choosed.

OR min no. of coins to reach 11 \Rightarrow
 \Rightarrow min $\left\{ \begin{array}{l} 1 \text{ coin of } 1 \text{ Rs} + \text{min coin to } 10 \text{ Rs} \\ 1 \text{ coin of } 2 \text{ Rs} + \text{min coin to } 9 \text{ Rs} \\ 1 \text{ coin of } 5 \text{ Rs} + \text{min coin to } 6 \text{ Rs} \end{array} \right.$

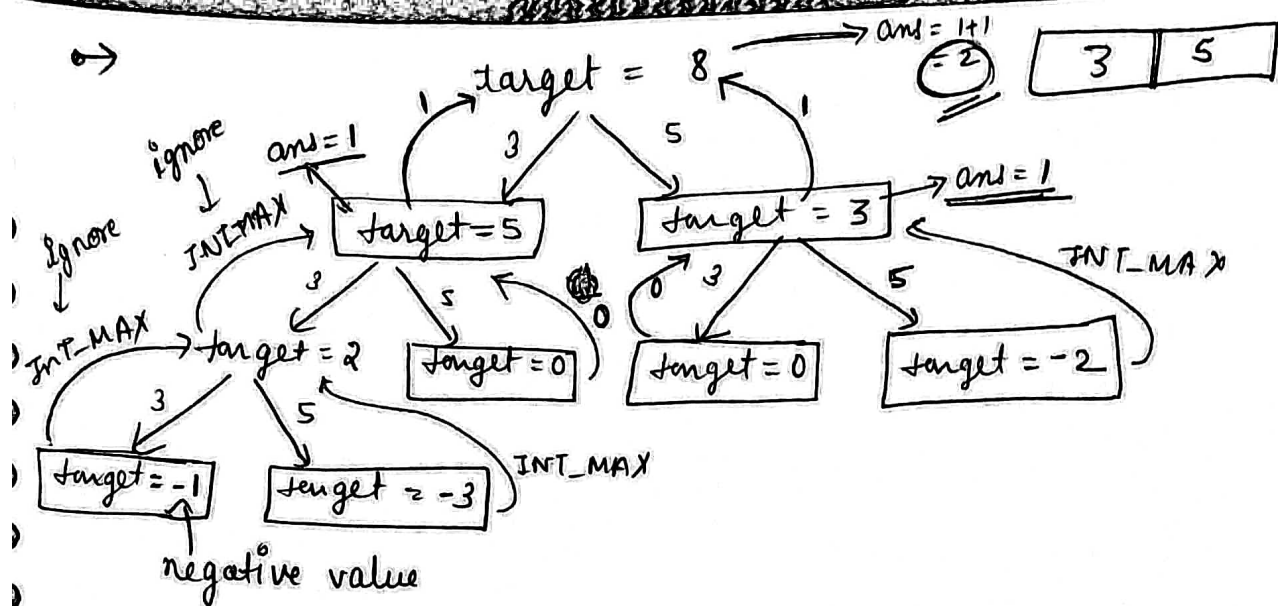


if (target == 0)
 \leftarrow 0 बनाने में 0 coin लगे,
 return 0;

if (target < 0)
 \leftarrow kitni भी coin लगा ली हम नहीं बना सकते,
 return INT_MAX;

min no. of coins to create $0 \Rightarrow 0$

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



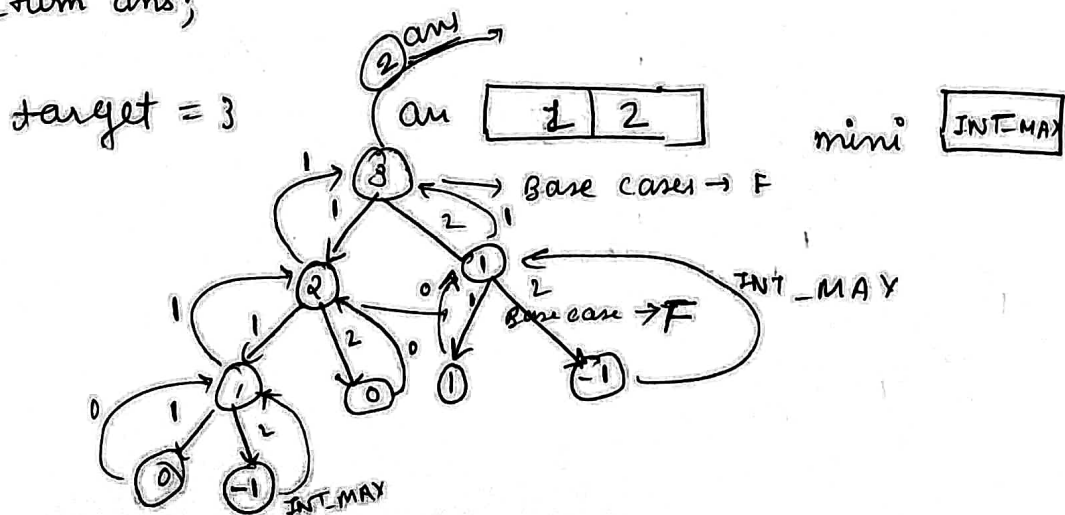
$$\text{target } 5 = 1 \text{ 5Rs coin} + \text{solve}(\text{target} = 0) \\ = \underline{1 + 0} = 1$$

$$\text{target } 3 = 1 \text{ 3Rs coin} + \text{solve}(\text{target} = 0) \\ = 1 + 0 = 1$$

```

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 mini;
}

```



① We learned two patterns →

① Include - Exclude

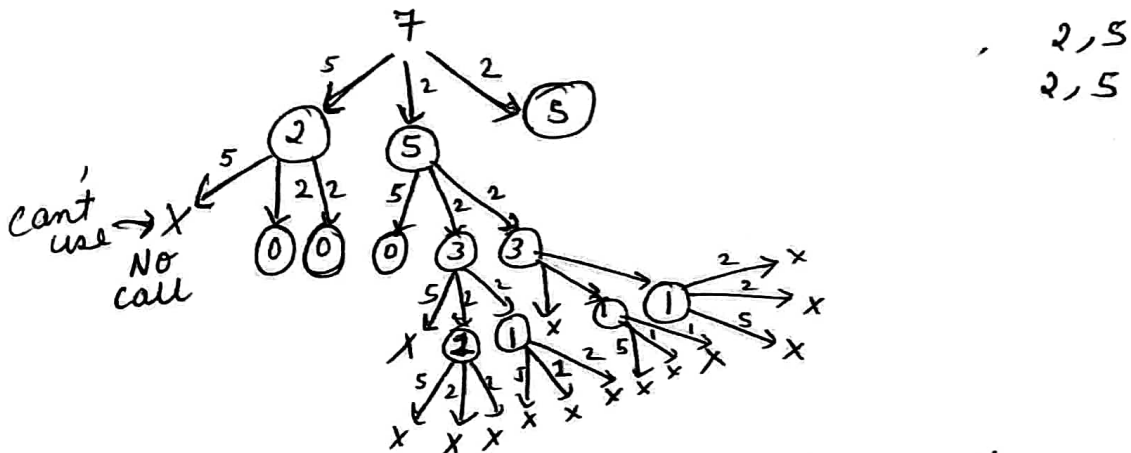
② for loop \rightarrow Iterate on each element

Ques 2- Cut into segments \rightarrow

$$i/p \rightarrow N \leftarrow \text{rod length}$$

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

eg $\rightarrow N = 7, \quad x = 5, y = 2, z = 2,$



// solve function returns max. no. of segments \rightarrow

```
int solve(int n, int x, int y, int z){
```

// base case

if ($n \geq 0$)

```
return 0;
```

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

```
int a = INT_MIN;
```

if $(n-x \geq 0)$ {

ans = solve (n-x, x, y, z) + 1;

3

```
int b = 0; INT_MIN;
```

$$\text{if } (n - y > 0) \{$$
$$b = \text{solve}(n-y, x, y, z) + 1;$$

3

```
int c = 0 INT_MIN;
```

$$if(n-z>0) \{$$
$$c = \text{solve}(n-2, x, y, z) + 1;$$

3

```
Int ans = max(a, max(b, c));
```

```
return ans;
```

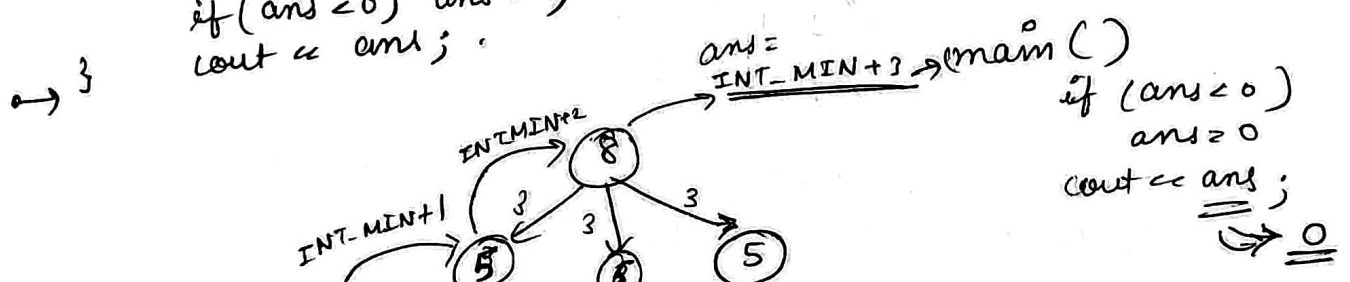

let's make a simple code →

```

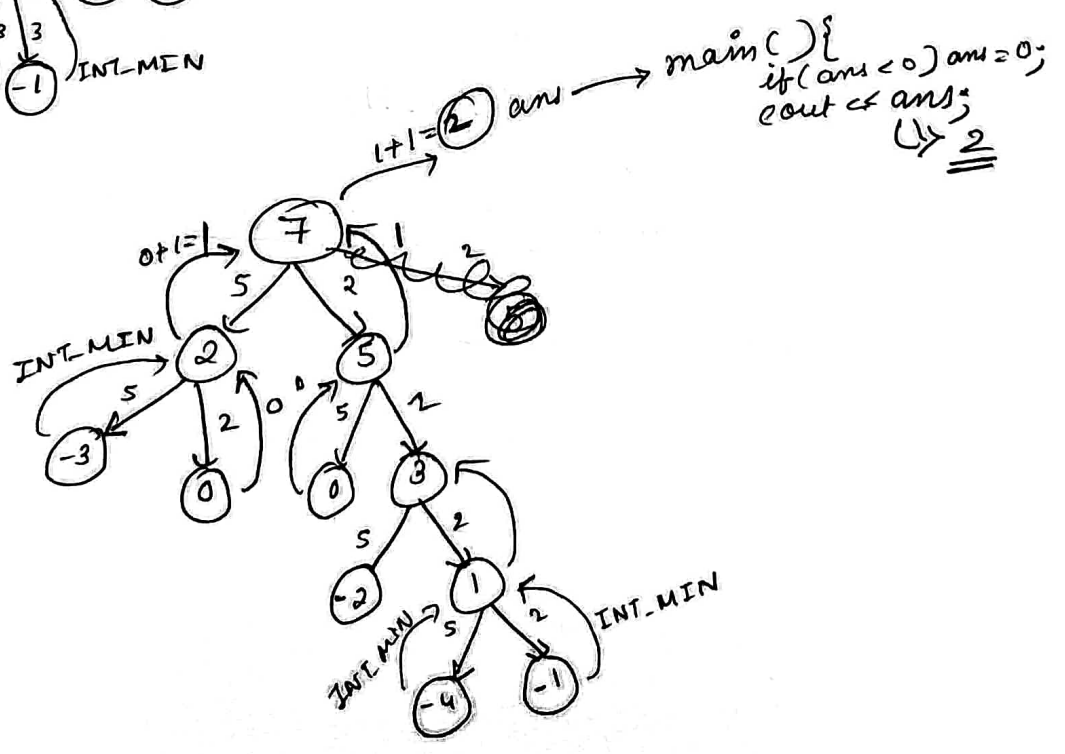
int solve(int n, int x, int y, int z) {
    if (n == 0) {
        return 0;
    }
    if (n < 0) {
        return INT_MIN;
    }
    int ans1 = solve(n-x, x, y, z) + 1;
    int ans2 = solve(n-y, x, y, z) + 1;
    int ans3 = solve(n-z, x, y, z) + 1;
    int ans = max(ans1, max(ans2, ans3));
    return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

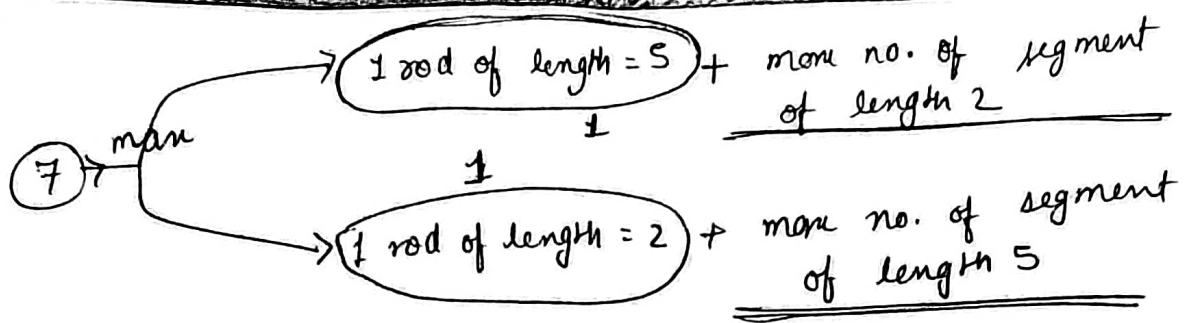
int main() {
    int n, x, y, z;
    cin >> n >> x >> y >> z;
    int ans = solve(n, x, y, z);
    if (ans < 0) ans = 0;
    cout << ans;
}

```



→ n = 7
x = 5
y = 2
z = 2





Ques 3 → Max Sum of non-adjacent elements →

i/p →

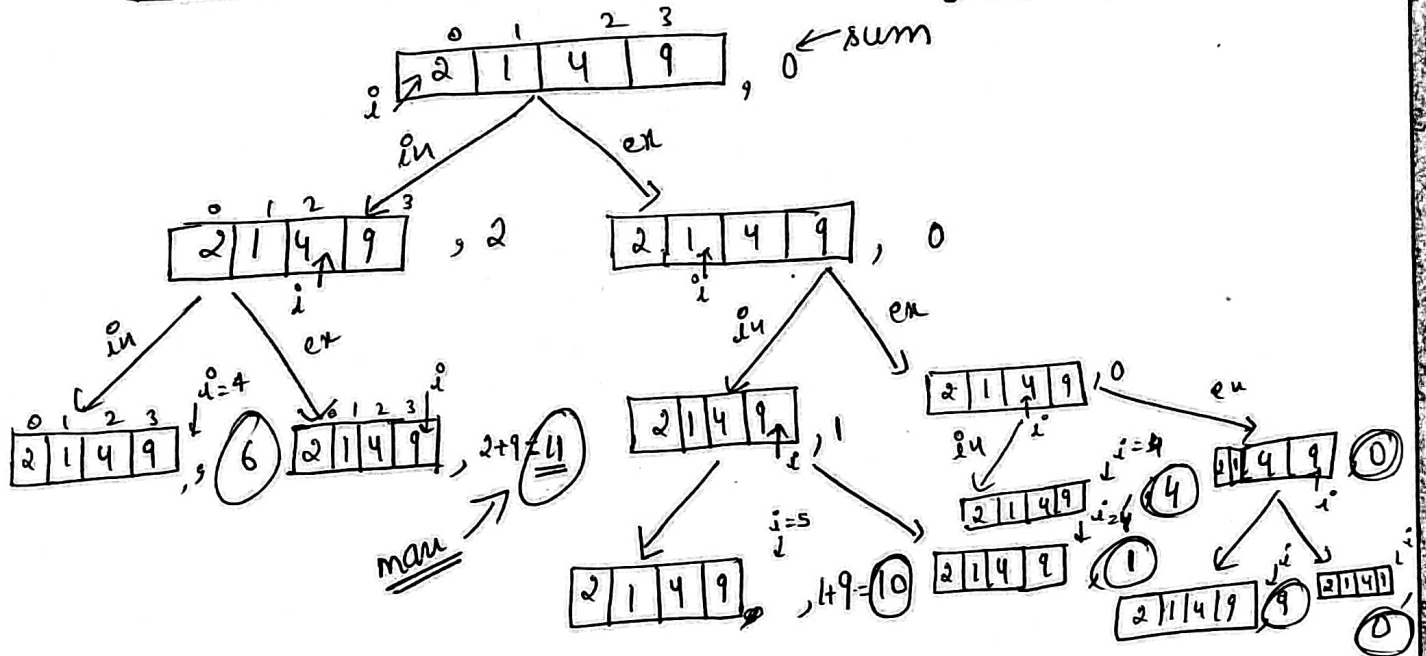
2	1	4	9
---	---	---	---

, Return the maximum sum of subsequence in which no. two elements are adjacent.

o/p → 11 (2+9)

(Same as House Robber-1)

Subsequence → ^{some} ~~may~~ elements is present some are not. Include/Exclude



Code →

```
void solve(int vector<int> &arr, int i, int sum 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; arr = arr.size();
```

```
    solve(arr, i, sum, maxi);
```

```
    cout << maxi;
```

```
}
```