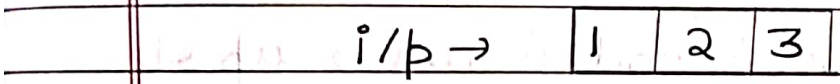
15/03/2023

Q1 There is array of size n & has n distinct elements. We have been given some target. We have to tell the minimum no. of elements required to reach target sum. (Coin change problem)



target = 5

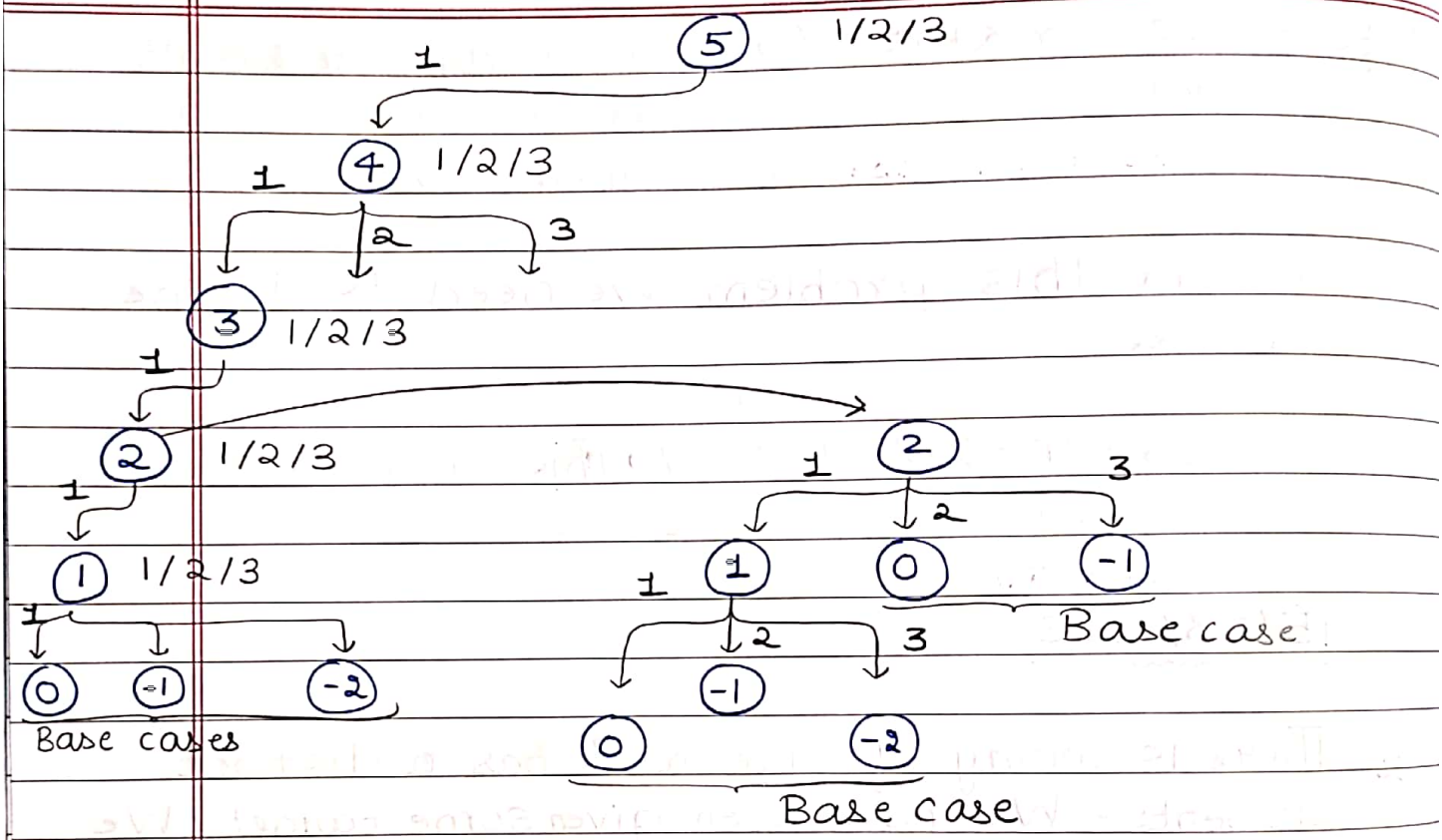
o/p → 2

{1, 1, 1, 1, 1}
{1, 1, 1, 2}
{2, 2, 1}
{1, 3, 1}
{3, 2}

} Some w

Making the recursive tree

The recursive tree is on the other page. If we are able to make the recursive tree, then we can easily code it.



Similarly the tree can be made for other elements also.
The 0 on the leftmost end is made up of {1, 1, 1, 1, 1} but we have to find the minimum no. of elements required.

Note → Initially mini is INT_MAX and whenever any -ve value is seen, then simply return INT_MAX as this won't change value of mini as we don't have to change mini in the invalid case.

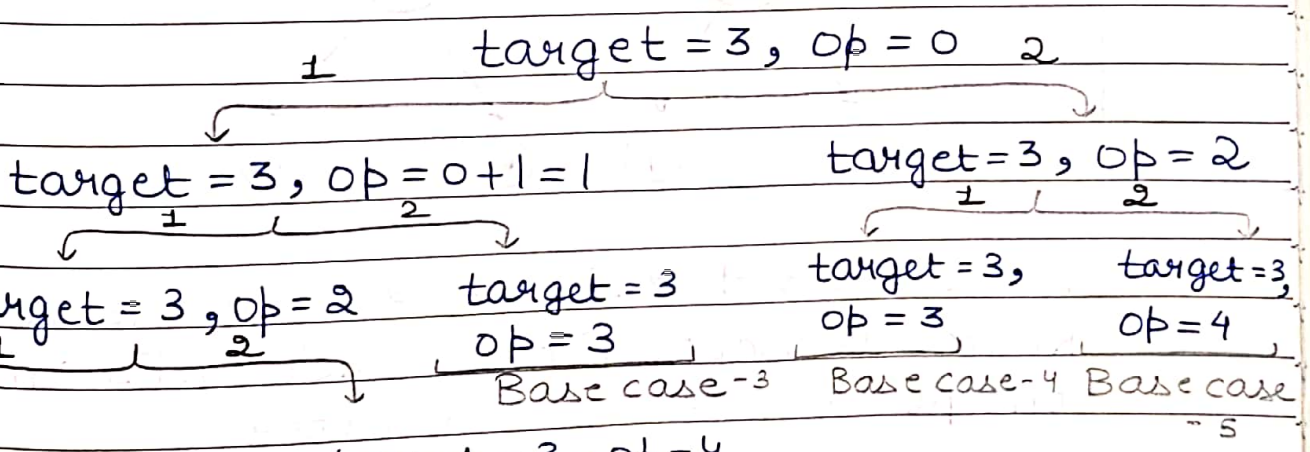
Code {target to 0}

```
int solve (vector <int> & arr , int target) {
        // Base case
        if (target == 0)
            return 0;
```

```
//Invalid case → mini should not get updated
    if (target < 0)
        return INT_MAX;
    //Solve 1 case
    int mini = INT_MAX;
    for (int i=0; i< arr.size(); i++) {
        // Include the coin & solve further
        int ans = solve (arr, target - arr[i]);
        //ans+1 is not out of range
        if (ans != INT_MAX)
//Find min. value → mini = min (mini, ans+1);
    }
    return mini;
}
```

Alternative approach {0 to target}

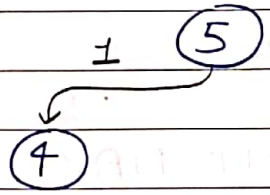
arr → {1, 2}
target = 3



Base case 1 → {1, 1, 1} 3
Base case 2 → Here op > target
Base case 3 → {1, 2} 2
Base case 4 → {2, 1} 2

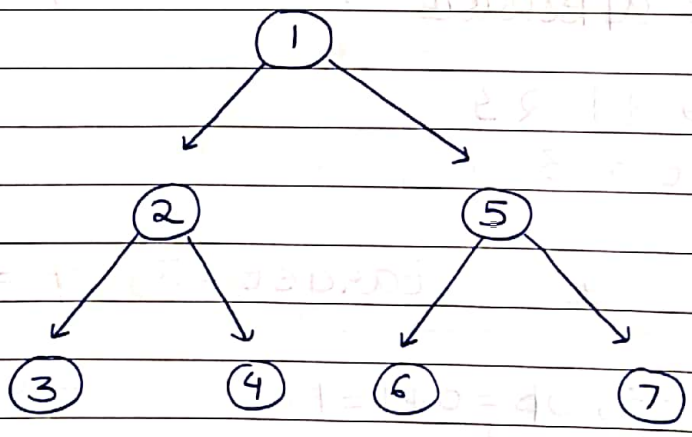
Base case 5 → ob > target
↳ elements
Minimum coins hence required = 2

Note → Why 1+ans used in min function?



solve was called for 4 but we have already include single coin so to consider that coin we have added 1 to the ans.

* Sequence of recursive calls



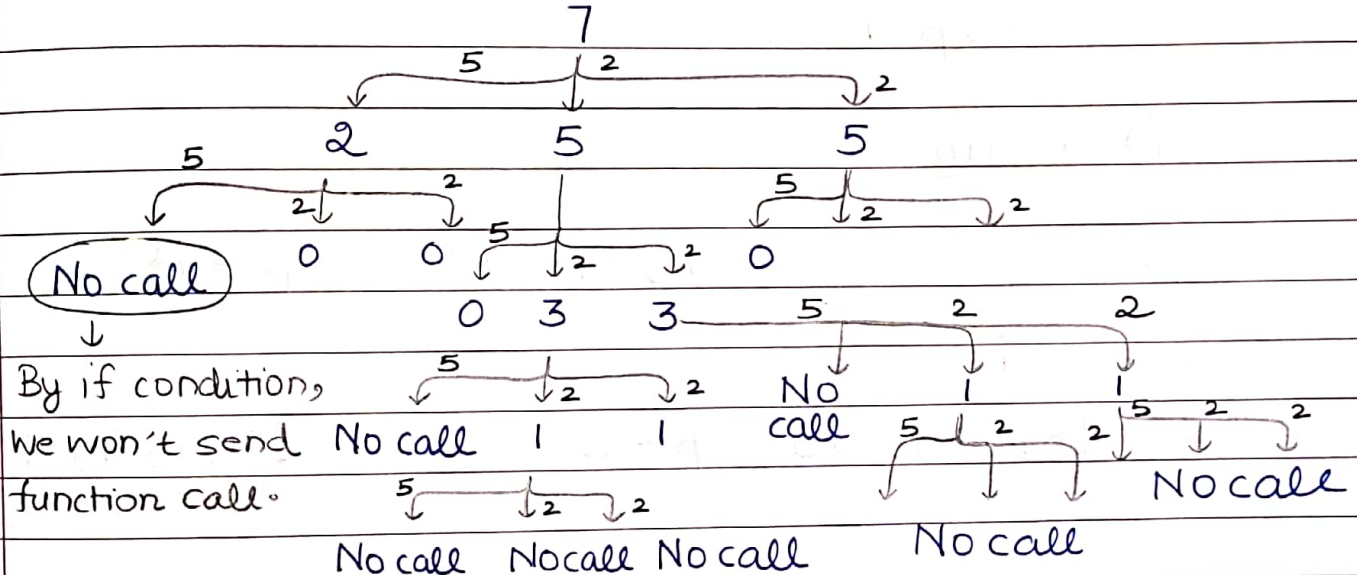
1 → 2 → 3 → 4 → 5 → 6 → 7. This is also known as Depth First Search (DFS). This concept will be studied later.

Q2 Cut into segments. There is a rod of length N. You need to determine maximum no. of segments such that segment will be of length x, y or z.

i/p → 7

x = 5, y = 2, z = 2

o/p →



Similarly the above tree can be successfully made. (Copy subtree of 5 to right side 5)

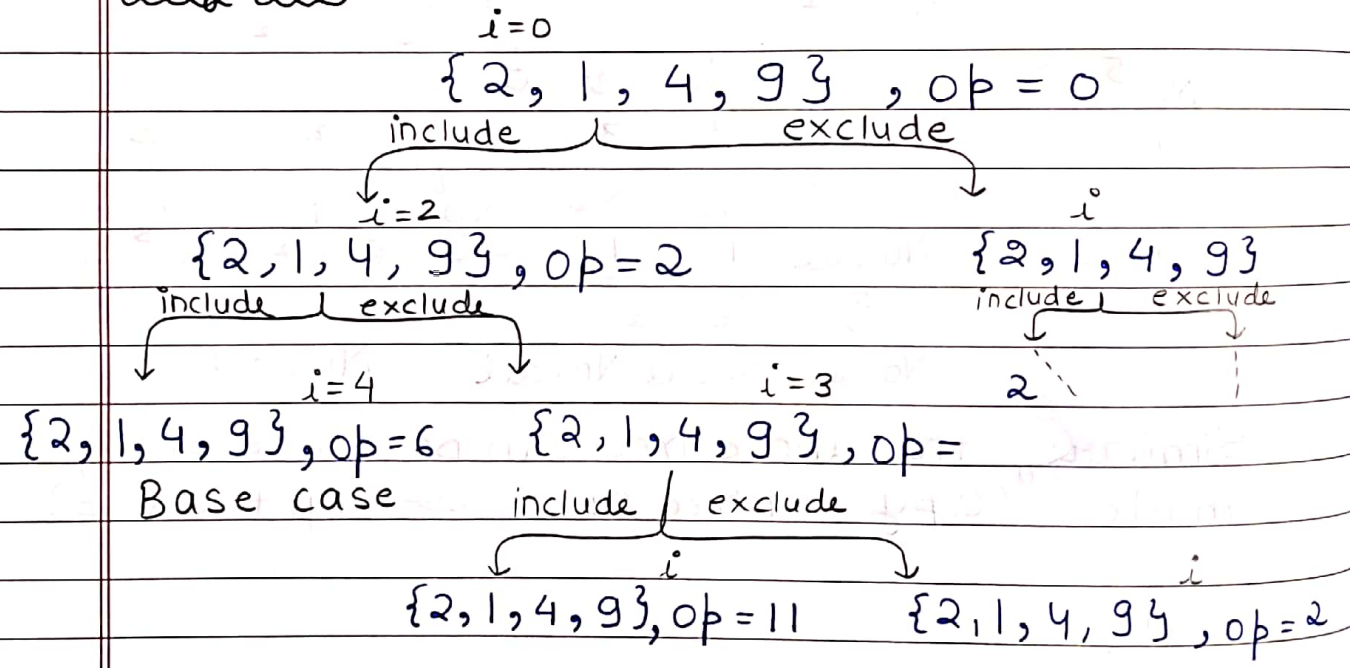
Code

```
int solve (int n, int x, int y, int z) {
    // Base case
    if (n == 0)
        return 0;
    // n-x, n-y, n-z can be negative, so handle it
    if (n < 0)
        return INT_MIN;
    // Solve I case
    int a = solve (n-x, x, y, z) + 1;
    int b = solve (n-y, x, y, z) + 1;
    int c = solve (n-z, x, y, z) + 1;
    int ans = max (a, max (b, c));
    return ans;   ↳ We need maximum segments.
}
```

Q3 Maximum sum of non-adjacent elements.

i/p → {2, 1, 4, 9}
o/p → 11 (2+9=11)

Dry run



Code

```
void solve (vector <int> & arr, int sum,
            int &maxi, int i) {
    // Base case as when index goes out of
    // range.
    if (i >= arr.size()) {
        maxi = max (sum, maxi);
        return;
    }
    //include call
    solve (arr, sum + arr[i], maxi, i+2);
```

```
//exclude call
    solve(arr, sum, maxi, i+1);

}
```

In main() function print the value of maxi as that contains the answer.

Why i+2 in case of include call?
Because we are told to consider the non-adjacent elements.