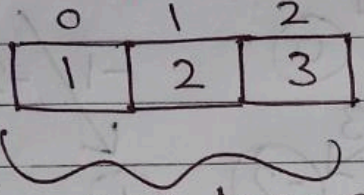
15-03-2023

Recursion (class 4)

Date.....

Ques: Given an array of size 'n' & has n distinct elements, also given a target. we have to give the mini-mum number of elements required to reach target sum. (V. Imp).



[Target = 5]

we have infinite (∞) suppy of these numbers.

this problem is also called COIN CHANGE PROBLEM

To make target, i.e., 5 from 1, we can add 1 to 5 times, i.e., 1+1+1+1+1, i.e., 5.
Similarly, we can make 5 by adding, 1+2+2 = 5 and 2+3 = 5

1+1+1+1+1 = 5 } 5 no. make target.

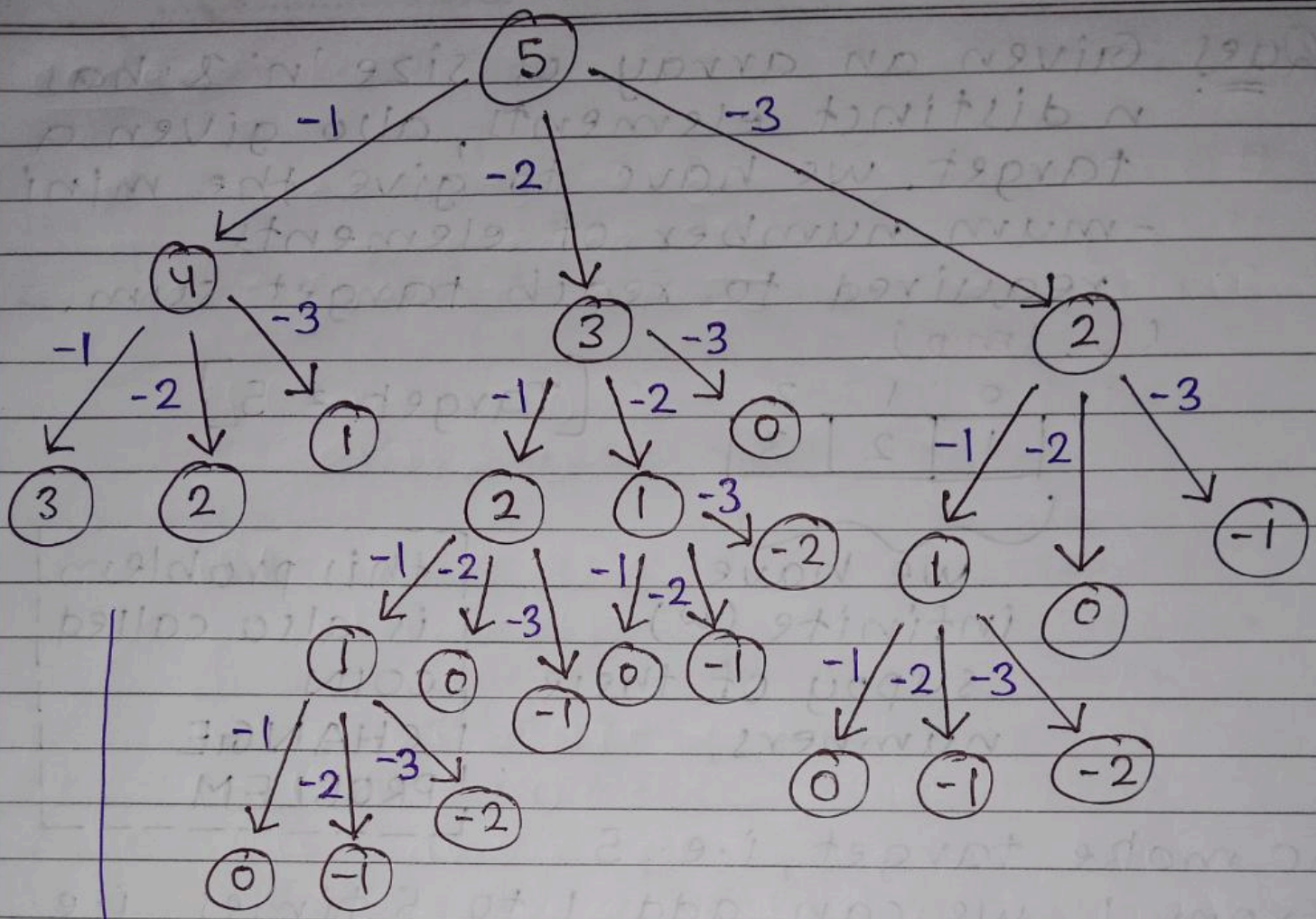
1+2+2 = 5 } 3 no. make target.

2+3 = 5 } 2 no. make target.

In this, we recursively take all the ele-ments & subtract them from our target, until we found 0 and/or we found a -ve number.

→ target

Date.....

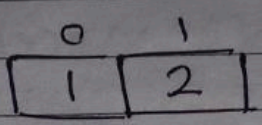


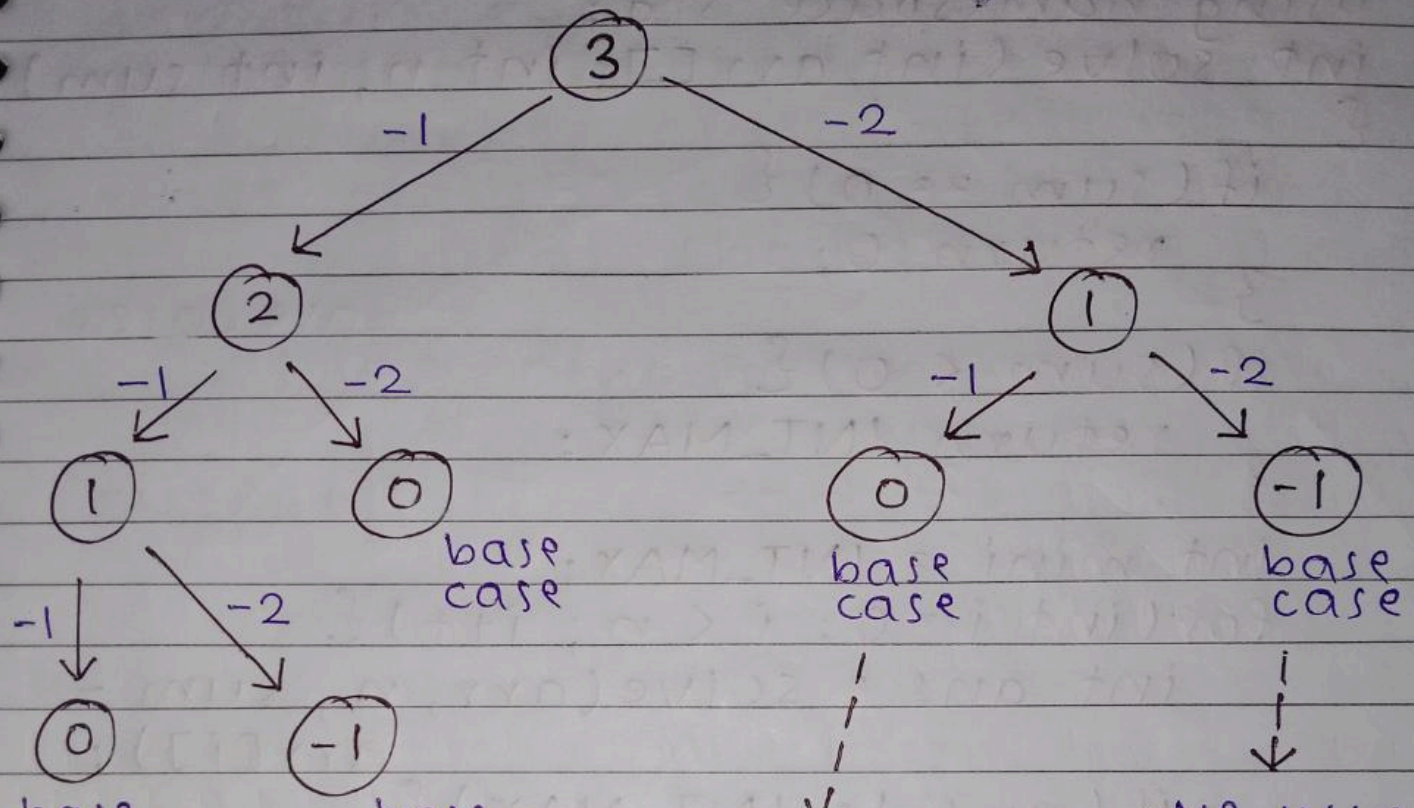
we'll find this one also like same way we find other nodes.

The Base case is that, we stop when target == 0 then return 0, as we don't need any other number to get 0 as target.
And if we found target as -ve then return any random number like INT_MAX, as we can't make -ve num from any other numbers.

Date.....

Example, [Target = 3]





0 will return back 0 value

-ve num will retu -rn back INT_MAX value.

Teacher's Sign .....

code:-

```
#include<bits/stdc++.h>
using namespace std;
int solve(int arr[], int n, int sum)
{
    if(sum == 0){
        return 0;
    }
    if(sum < 0){
        return INT_MAX;
    }
    int mini = INT_MAX;
    for(int i=0; i<n; i++){
        int ans = solve(arr, n, sum -
                        arr[i]);
        if(ans != INT_MAX){
            mini = min(mini, ans+1);
        }
    }
    return mini;
}
int main()
{
    int arr[] = {1, 2, 5};
    int n = 3;
    int sum = 11;
    int ans = solve(arr, n, sum);
    cout << "minimum coins to make"
         << sum << "is" << ans;
    return 0;
}
```

Base cases

ans+1, means we call recursive function for second sum, but we need original sum, so we add one more coin to that sum.

Date.....

Ques cut into segments. (V.Imp)

we have given an integer 'n' denoting the length of the rod. we need to determine the maxim -um number of segments we can make of this rod provided that each segment should be of length x, y and z.

Example :- Input, n = 11
x = 2
y = 3
z = 5

→ length of rod.

output, 5

↓

segment of {2, 2, 2, 2, 3} makes 11

Given lengths, from these lengths we have to make the length of the rod, & we have to return the maxi -mum no. of segments.

Example :- Input : n = 7 → length of rod.
x = 2
y = 5
z = 5

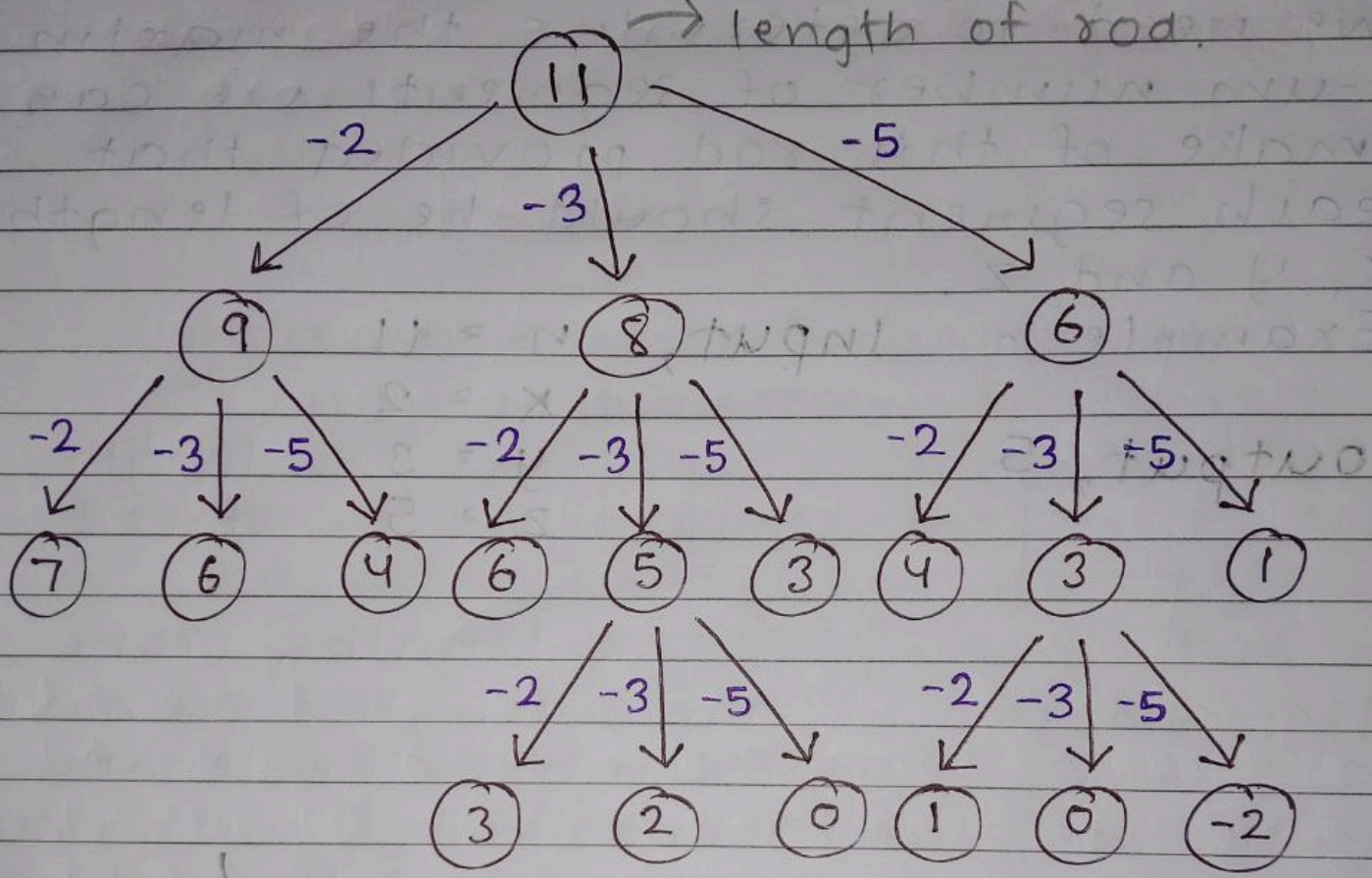
output :- 2

↓

segments of {2, 5} makes 7.

Date.....

Like we apply the pattern Queition, the same way we apply same pattern in this Queition also.



Like same way we make call everytime.
& base condition is that if (n >= 0) then return 0.

code:-

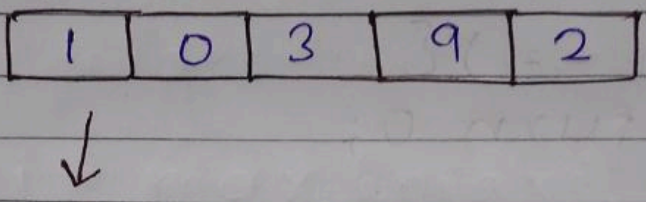
```
#include<bits/stdc++.h>
using namespace std;
int solve(int n, int x, int y,
                          int z)
{
    if(n == 0){
      return 0;
    }
    if(n < 0){
      return INT_MIN;
    }
    int ans1 = solve(n-x, x, y, z) + 1;
    int ans2 = solve(n-y, x, y, z) + 1;
    int ans3 = solve(n-z, x, y, z) + 1;
    int ans = max(ans1, max(ans2,
                          ans3));
    return ans;
}
int main()
{
    int n = 11, x = 2, y = 3, z = 5;
    int ans = solve(n, x, y, z);
    if(ans < 0){
      ans = 0;
    }
    cout << "maximum segments in
      which we cut the rod is " <<
        ans << endl;
    return 0;
}
```

Date.....

Ques. Maximum sum of non adjacent elements. (V. Imp).

We've given an array of integers, we have to find the maximum sum of non-adjacent elements.

For example :-



non adjacent elements are :-

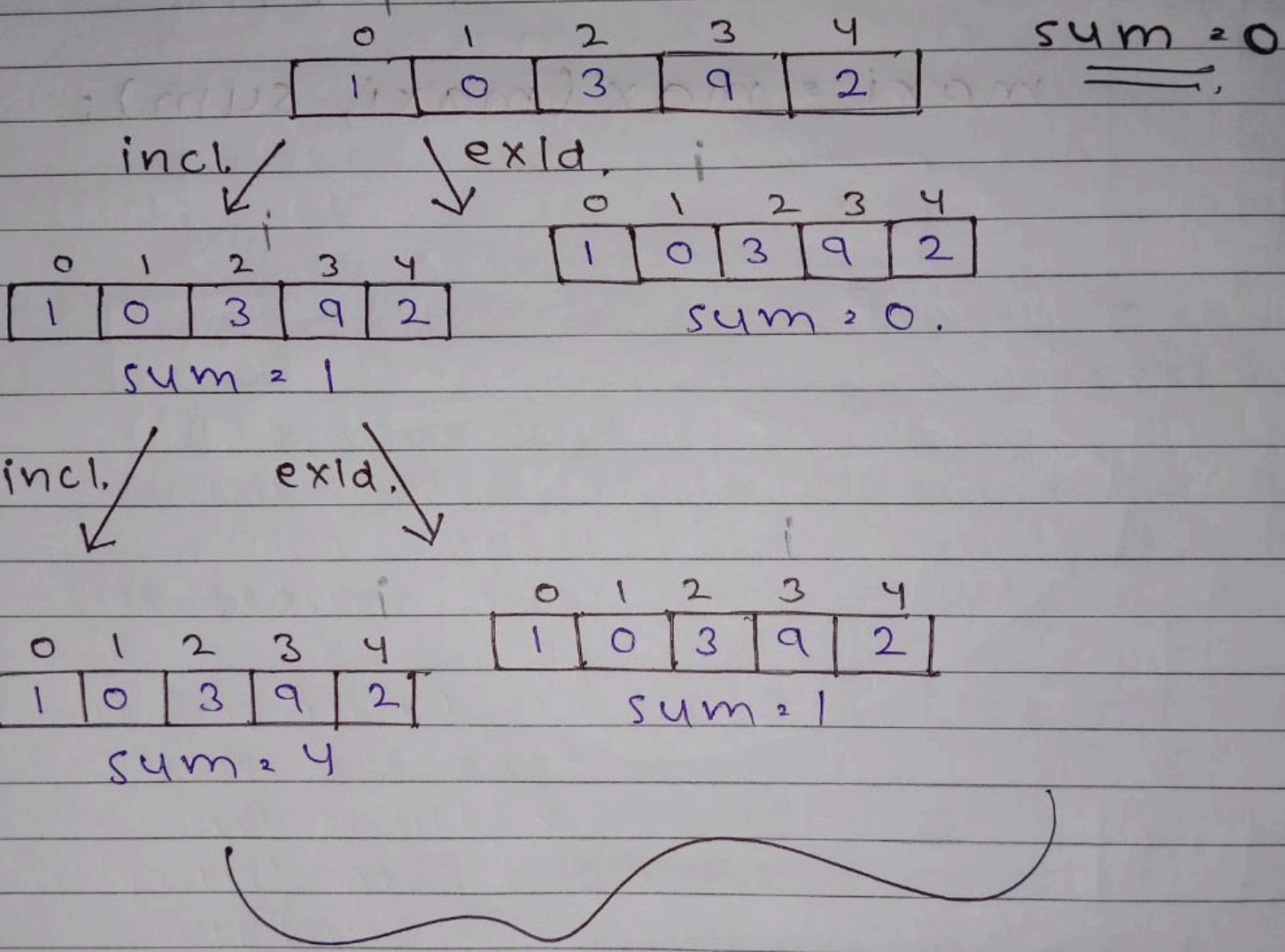
(1, 3), (1, 3, 2), (1, 9)
(1, 2), (0, 9), (0, 2),
(3, 2)

$1 + 3 = 4$
$1 + 3 + 2 = 6$
$1 + 9 = 10$
$1 + 2 = 3$
$0 + 9 = 9$
$0 + 2 = 2$
$3 + 2 = 5$

maximum sum is : 1 + 9, i.e., 10.

Or we can say, we have to return the maximum sum of subsequences in which no two elements are adjacent.

In this question, we have to return the maximum sum of all the subsequences. so it means, the subsequences pattern is also used in this problem also.
The subsequence pattern is Exclude & Include pattern.



like this way, we follow the Include Exclude pattern.

* If we include an element then we can't include just the next element because of non adjacent property. so, 'i' will be incremented by 2.

Date.....

* If we exclude an element then we can consider the next eleme -nt. so, in exclude case, 'i' will increment by 1.

* If 'i' >= n, then only we update the maximum variable which holds the max sum, i.e.,

```
maxi = max(maxi, sum);
```

code:-

```
#include<bits/stdc++.h>
using namespace std;
void solve(int arr[], int n, int i,
                    int sum, int &maxi)
{
    //base case
    if(i >= n){
        maxi = max(maxi, sum);
        return;
    }
    //include case
    solve(arr, n, i+2, sum + arr[i]
                        ,maxi);
    //exclude case
    solve(arr, n, i+1, sum, maxi);
}
int main()
{
    int arr[] = {1, 0, 3, 9, 2};
    int n = 5;
    int sum = 0;
    int maxi = INT_MIN;
    int i = 0;
    solve(arr, n, i, sum, maxi);
    cout << "\n Maximum sum in
      non adjacent elements are: "
      << maxi;
    return 0;
}
```