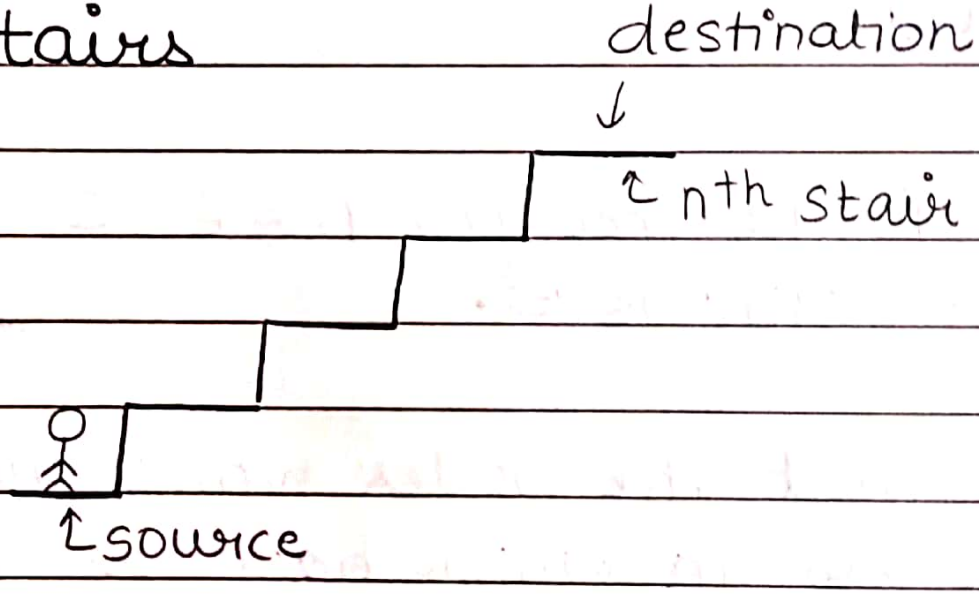
11/03/2023

Questions on Recursion

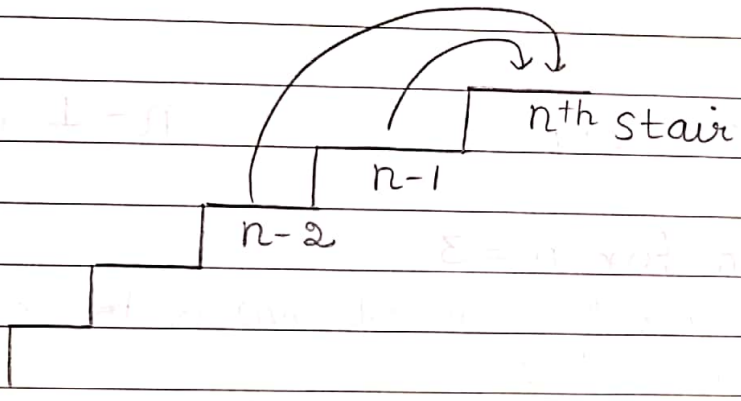
Q1 Climb Stairs



Steps allowed ⇒ move 1 stair at a time
move 2 stair at a time

Find the minimum number of steps to reach nth stair.

We are currently on the 0th stair. We just have to solve one case & rest recursion will handle.



To reach the n^{th} stair, we can reach from $(n-1)^{th}$ stair or $(n-2)^{th}$ stair.

n^{th} stair = $(n-1)^{th}$ stair + $(n-2)^{th}$ stair

$f(n) = f(n-1) + f(n-2)$

↳ As we have to find total ways

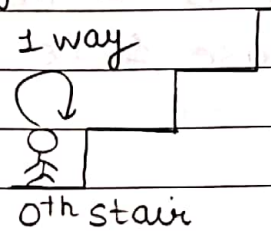
The recursive relation is same as that of the fibonacci series.

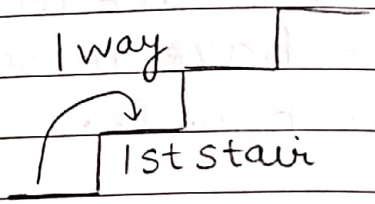
Code

```
int climbStairs (int n) {
        // Base Case
        if (n == 0 || n == 1)
                return 1;
        // Recursive relation
        return climbStairs (n-1) + climbStairs
                                        (n-2);
}
```

Understanding base case

(only for n = 0 case)





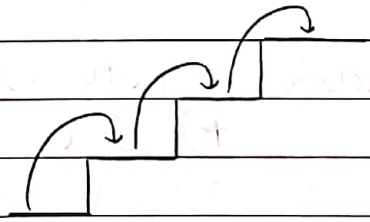
n = 0, 1 way

n = 1, 1 way

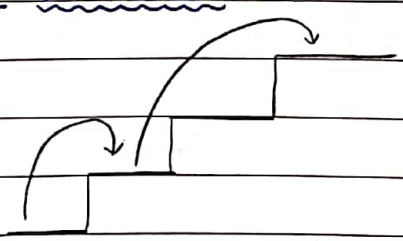
Dry run for n = 3

The answer for no of ways to reach the 3rd stair is 3

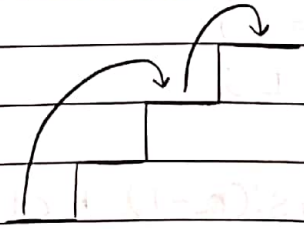
1st solution



2nd solution



3rd solution



The most confusing thing in the question is the base case for 0th stair, that is only 1 way is there to reach 0th stair.

↳ Time Limit Exceeded

Note→ There will many TLEs in case of recursion which will further be optimized.

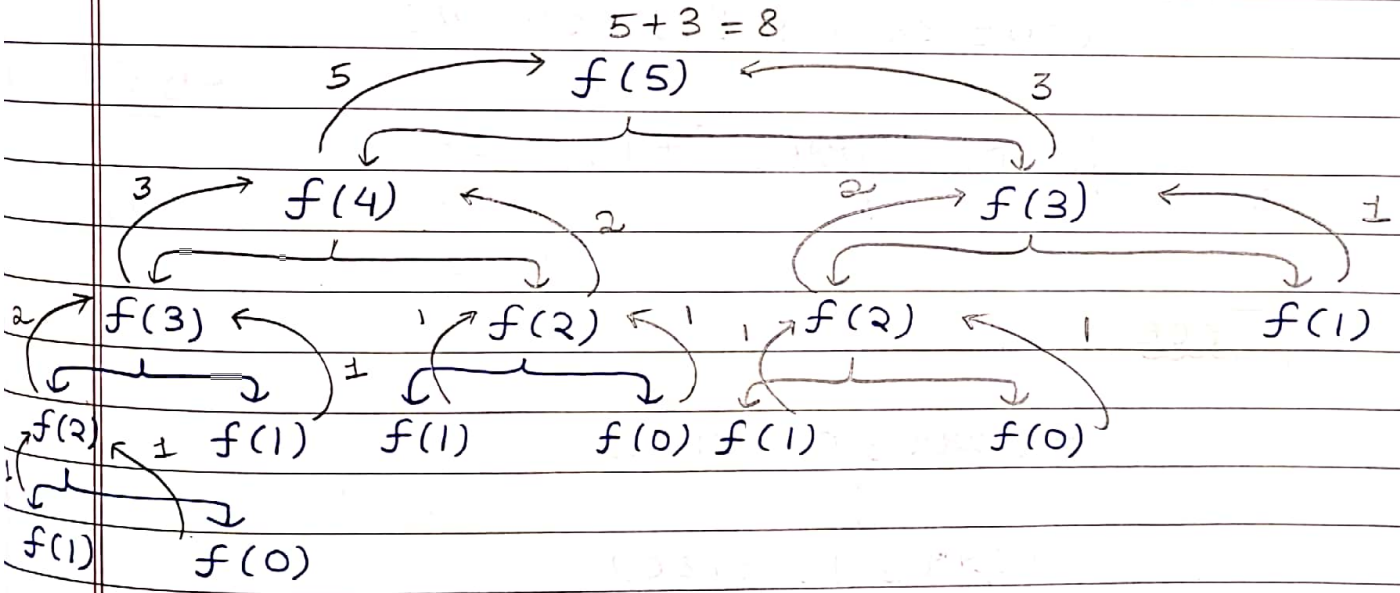
$f(5) = f(4) + f(3)$

↳ $f(3) + f(2)$

↳ $f(2) + f(1)$

↳ $f(1) + f(0)$

Similary for $f(3)$ in 1st line, calls will be sent. See the recursion tree below.



Hence to reach the 5th stair, there are 8 possible ways.

Note → $f(0) = 0$ is not selected as it means that we can't reach the 0th step/stair but we are already on 0th stair & hence we take it as 1.

Q2 Print the elements of the array.

We will simply take the base case as if index becomes equal to size of array, we need to return. After that simply print

the element & rest recursion will handle.

Code

```
void print (int arr [], int i, int size){
        // Base Case
        if (i == size)
                return;
        // Processing
        cout << arr [i] << " ";
        // Recursive Relation
        print (arr, i+1, size);
                        ↳ move to next
                          index
}
```

Tree

```
f ( arr , 0 , size)
        ↓ cout << arr [0];
f (arr , 1 , size)
        ↓ cout << arr [1];
f (arr , 2 , size)
        ↓ cout << arr [2];
        ⋮ } goes on until base
          condition is achieved
```

Alternative code

```
void print (int arr [], int size) {
        // Base Case
        if (size == 0)
                return;
```

```
// Processing
    cout << arr[0] << " ";
    print(arr+1, n-1); //Recursive Relation
}
```

Q3 Find maximum element in the array.
First create a variable maxi & initialize it with INT_MIN. Now as we start traversing the array, we will check that if the maxi is less than arr[i], if true then update maxi.

Code

```
void findMax(int arr[], int i, int size,
             int & maxi) {
    // Base Case          ↳ initialized to INT_MIN
    if (i == size)
        return;
    // Processing
    if (arr[i] > maxi)
        maxi = arr[i];
    // Recursive Relation
    findMax(arr, i+1, size, maxi);
}
```

The dry run of the above code is similar but instead of printing we will be comparing the maxi & arr[i].

Q4 Find minimum element in the array.

```
void findMin(int arr[], int i, int size,
             int & mini) {
                    ↳ initialized to INT_MAX
    // Base case
    if (i == size)
```

```
    return ;
    // Processing
    if (mini > arr[i])
        mini = arr[i];
    // Recursive Relation
    findMin(arr, i+1, size, mini);
}
```

The approach is same as that of Q3 but the if condition has changed.

Q5 Find whether key is present in string or not.

i/p → lovebabbar , key = 'r'
o/p → Yes

Will be done by recursion
(l) o v e b a b b a r
↳ Checking this only (1st case solve)

Base condition ⇒ If we have traversed whole string, then simply return false. If while traversing, we found the key then simply return true otherwise we move on to the next index.

Code

```
bool checkkey(string str, int i, int n,
              int key) {
    // Base condition
    if (i == n)
        return false;
```

```
// 1 case solve
if (str[i] == key)
        return true;
// Recursive relation
return checkkey (str, i+1, n, key);
}   not mandatory          ↳ move to next index.
```

Note → By doing &, TLE can get eliminated. So here string can be passed as reference.

Here we can return index of the key in string. Just change bool type to int, false to -1 and true to i. If there are multiple occurences of the key in the string, then we can store those indexes in vector.

Q6 Find the digits of the number.

i/p → 647
o/p → digits
6
4
7

647 % 10 = 7 } 1st digit
647 / 10 = 64

64 % 10 = 4 } 2nd digit
64 / 10 = 6

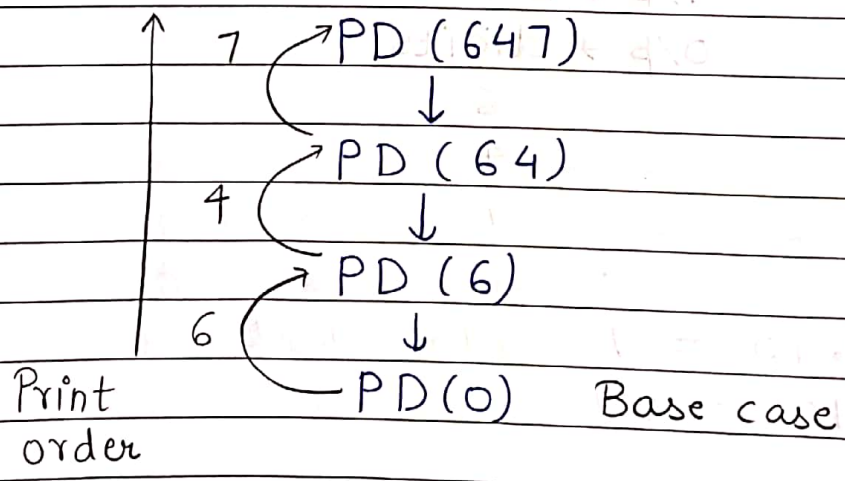
6 % 10 = 6 } 3rd digit
6 / 10 = 0 → stop

Base condition ⇒ When n becomes 0, we need to stop.

Code

```
void printDigits(int n){
    //Base case
    if(n==0)
        return;
    //Recursive Relation
    printDigits(n/10);
    //Processing
    int digit = n%10;
    cout << digit << " ";
}
```

Tree



6 4 7 will be printed which are the digits of number 647.