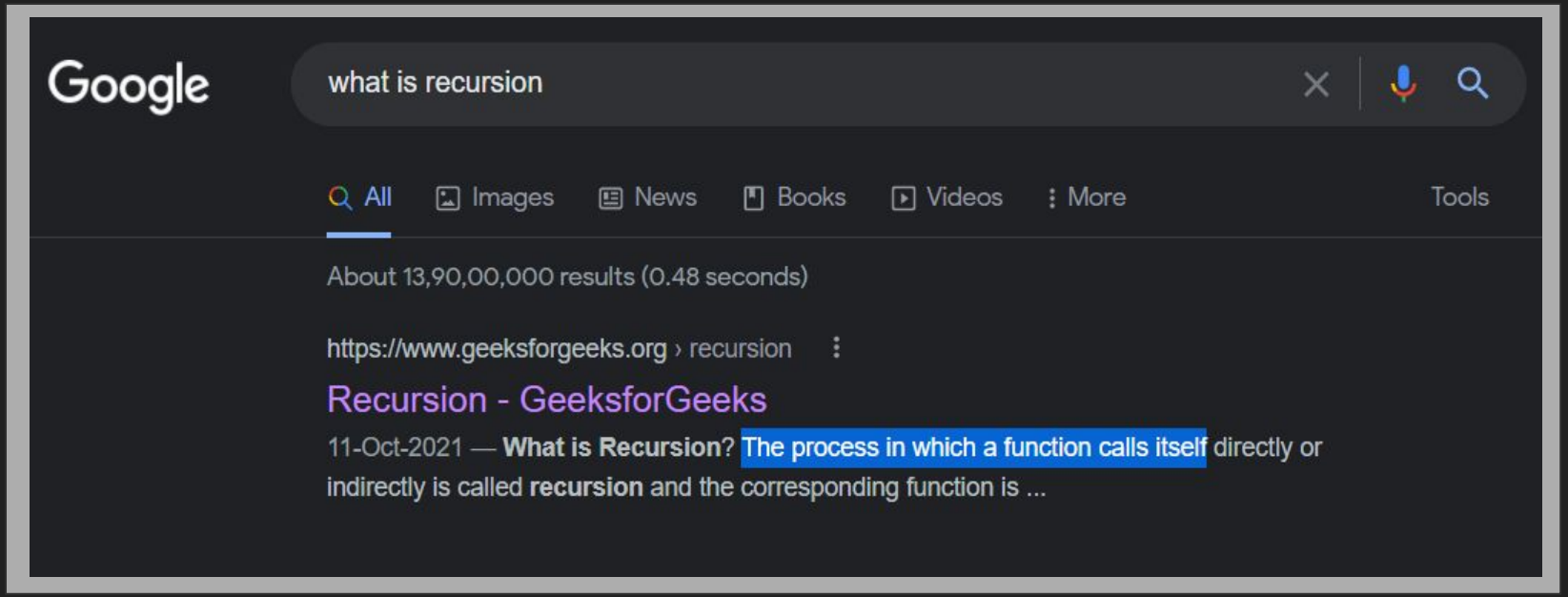
LEARN THIS LECTURE FROM MY VIDEO INSTEAD

Quickest way to THOROUGHLY understand recursion for DSA/CP

Recursion



About Me

Not Bragging, just telling it to learners so that they learn confidently with faith in the teacher.

Hi, I am Utkarsh Gupta.

Upcoming Google Software Engineer. (Offcampus, Google contacted me)

I am one of the best Competitive Programmers in India.

Subscribe to my YT Channel for more content

Achievements:

India Ranks 2, 2, 2, 3 in Google Kickstart Round A,B,C,D respectively.

Grandmaster on Codeforces (India Rank 2)

7 star coder on Codechef

Watch me do Leetcode Weekly Contest in less than half time

BENEFITS OF THIS SLIDE DECK

- No prerequisites: just basic if-else, loop, function etc knowledge needed.
- Recursion is a very important concept, used in the harder DSA concepts like Trees, Graphs, DP, etc also
- Both C++ and Java users can understand (just try to read the codes)
- A 2* coder and a 7* coder both know recursion, but the difference is in the way of thinking! This lecture is different from most other recursion lectures because I'll teach how to THINK RECURSIVELY like a grandmaster.
- Koi pooche ki itna accha recursion kaise aata hai? Bolna ki "Utkarsh bhaiyya se seekha hai."

Simplest Program: Say Hello

```cpp
#include "bits/stdc++.h"
using namespace std;

int main(){
    cout << "hello\n";
}
```

OUTPUT:
hello

Same Program with Function: Say Hello

```cpp
#include "bits/stdc++.h"
using namespace std;

void say_hello(){
    cout << "hello\n";
}

int main(){
    say_hello();
}
```

THIS TRUST IS VERY VERY IMPORTANT LATER

function does its work, we TRUST it

here we don't think how the function works

OUTPUT:
hello

Google says: Recursion = function calls itself, ok let's do it

```cpp
#include "bits/stdc++.h"
using namespace std;

void say_hello(){
    cout << "hello\n";
    say_hello();
}

int main(){
    say_hello();
}
```

GOES INFINITELY:
hello
hello
hello
hello
hello
hello
hello
hello
.
.
.

LET'S TRY TO PRINT IT
FIXED NUMBER OF TIMES

```cpp
#include "bits/stdc++.h"
using namespace std;

void say_hello(int n){
    cout << "hello\n"; // 1 time done
    say_hello(n - 1); // n - 1 more times
}

int main(){
    say_hello(10);
}
```

Print n times

Print 10 times

BUT Problem still not solved????! Why???!

Let’s print value of n also to debug

```cpp
#include "bits/stdc++.h"
using namespace std;

void say_hello(int n){
    cout << "hello " << n << "\n";
    say_hello(n - 1);
}

int main(){
    say_hello(10);
}
```

Printing n for debugging

OUTPUT:

```
hello 10
hello 9
hello 8
hello 7
hello 6
hello 5
hello 4
hello 3
hello 2
hello 1
hello 0
hello -1
hello -2
hello -3
hello -4
hello -5
hello -6
```

SHOULD'VE STOPPED HERE

Let's add another
line to stop at n = 0

```cpp
#include "bits/stdc++.h"
using namespace std;

void say_hello(int n){
    if(n == 0){ // stopping case or base case
        return;
    }
    cout << "hello " << n << "\n";
    say_hello(n - 1);
}

int main(){
    say_hello(10);
}
```

NOW OUTPUT IS CORRECT

```
hello 10
hello 9
hello 8
hello 7
hello 6
hello 5
hello 4
hello 3
hello 2
hello 1
```

Base case is compulsory for stopping the recursion

We'll learn an amazing thing in the next slide

TRUST YOUR FUNCTION

In the code from last slide, say_hello(n) prints:

hello n

hello n - 1

hello n - 2

hello n - 3

..

hello 1

```
cout << "hello " << n << "\n";
say_hello(n - 1);
```

This is exactly say_hello(n - 1)

VERY VERY IMPORTANT

Remember slide 5? I talked about TRUSTING your function!
This is useful here, just trust that say_hello(n-1) will print the green box, instead of doing dry run in your head

Ok, let's move to next slide:
Try to print:
hello 1
hello 2
…
hello n

Print the numbers in increasing order instead of decreasing

hello 1

hello 2

hello 3

hello 4

.

.

hello (n-1)

hello n → cout << “hello “ << n << “\n”;

Sub-Problem, just TRUST the function to solve it
say_hello(n-1)

```
void say_hello(int n){
    if(n == 0){
        return;
    }
    say_hello(n - 1); // first n-1 lines
    cout << "hello " << n << "\n";
}
```

Why TRUSTING your function works?

> *Recursive functions require a leap of faith — you must assume that the function will work on the simpler recursive call, and if that is the case, the whole function is correctly defined.*
>
> (source)

For a more rigorous proof, think of mathematical induction, we know that the base case is correct, if we assume (or TRUST) the subproblem to be correct, and based on that if our bigger problem is correct, then it'll be always correct.

For example, 0 is correct because base case, 1 is correct if 0 is correct, 2 is correct if 1 is correct and so on, hence it is always correct.

This Leap of Faith or Trust on your function is essential for recursion, because it is impossible to dry run a recursive code while writing it.

Practice: Use Recursion to-

Solutions and approach discussed in the video

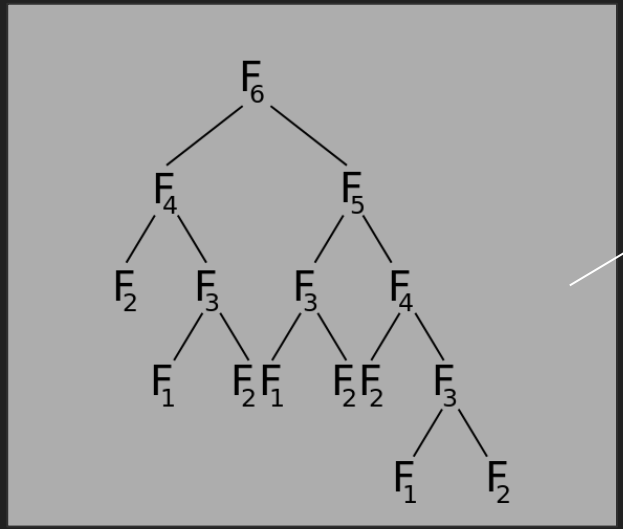
- Sum of numbers from 1 to n
- Sum of digits of a number
- Factorial of a number n
- Pattern Printing:
  - 1
1 2
1 2 3
1 2 3 … n
  - 1 2 3 … n
1 2 3
1 2
1
  - 1 2 3 … n
1 2 3
1 2
1
1 2
1 2 3
1 2 3 … n
- Fibonacci Sequence
- Check if a string is palindrome
- (Math based)Find nCr. [hint: nCr = (n-1)Cr + (n-1)C(r-1)]

Time Complexity

Make a tree-like structure of the recursive calls to estimate the time complexity (harder techniques like Recurrence Relations, Master Theorem etc are usually not needed)

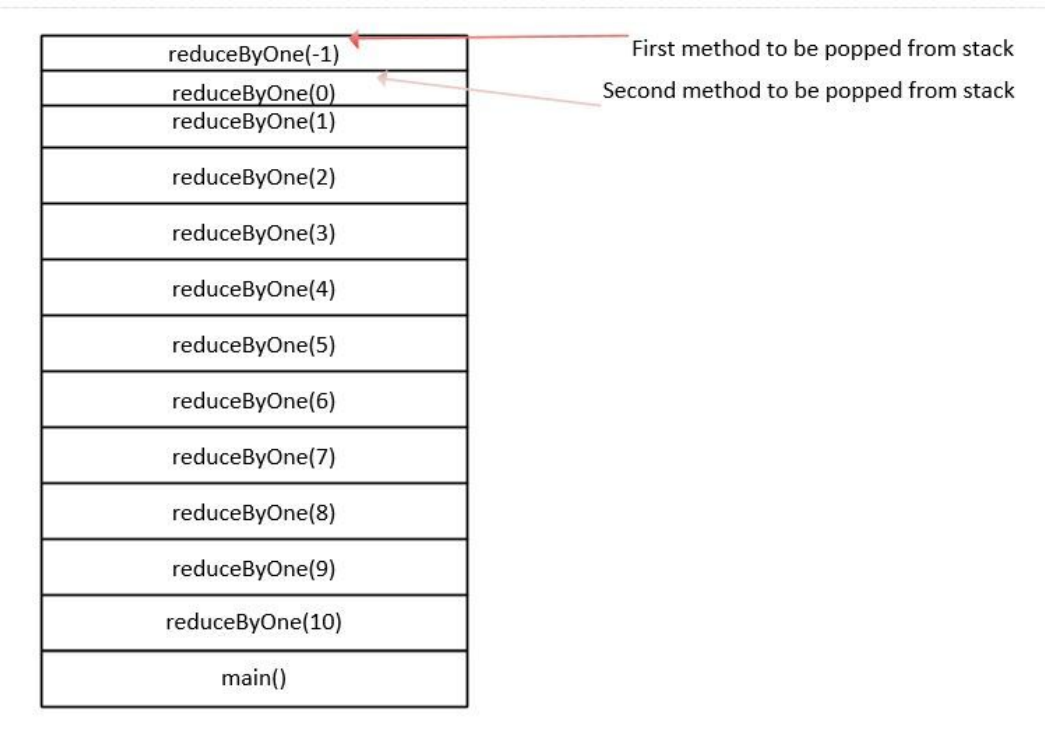
Fibonacci:

(for example)



Upon counting the elements, it can be estimated that it is exponential and loosely bounded by 2^n
Watch video for more explanation

NOT NECESSARY FOR DSA: Internal working of Recursion



Read in more detail
https://dotnettutorials.net/lesson/how-recursion-uses-stack/

Future Study

Once you know recursion, you can learn topics like Backtracking, Trees, Graphs, etc. (Will add links as soon as I teach those, subscribe to YT channel to stay updated)