Fundamentals of Programming

Class Notes

Dated: 25 Jan 2022

Presented By
Amarnath

Agenda

- Skillsets
- Toolset
- History of Python Program
- Problem Understanding
  - Requirement Analysis
  - Proposals
    - Algorithm Design
    - Flow Chart Design

Agenda

- Program Development
- Testing / Test Cases
- Complexity Analysis
  - Time & Space
  - Stack Memory
- Assembly Programming
- Platform Dependent / Independent
- Recursion & Time Analysis
- Coding Standards [C, Java, Python] + Comments/formats + Modules.
- Organization Structure

* Programming Languages as a skill set

Wrong

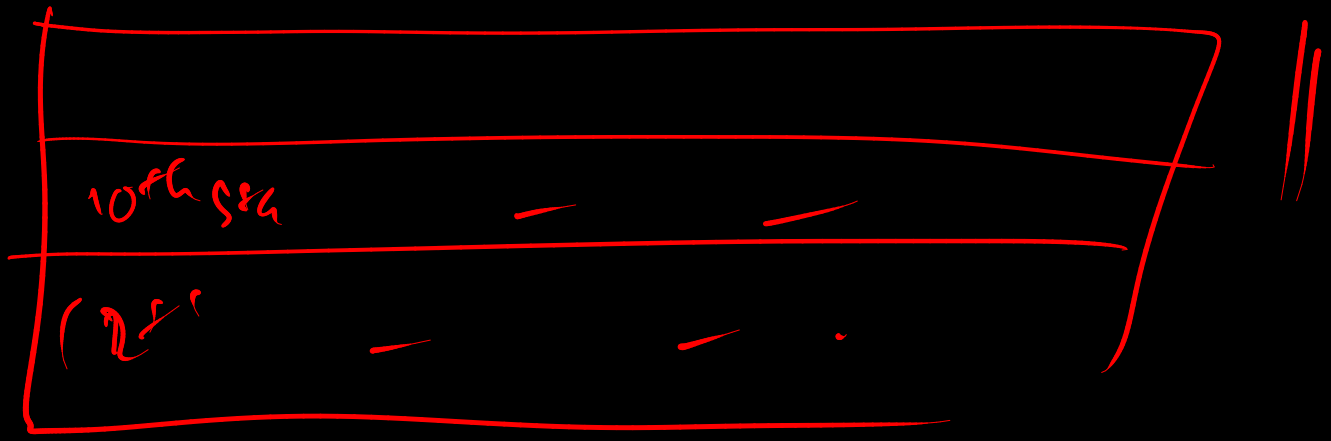
→ Toolset → C, C++, Java, Python

* Skillsets : → Team mem
  - Individual Contribut.
  - Public Speak
  - Even Organize
  - Analytic Skills
  - Many Lang [Hindi, Kannada, Tamil]

* Toolsets :- Program Analytics
  - C, (Turbo), GCC, //

* Computational skillset

L) Graph Theory, AI, Deep Learn, Pattern Recog, Prob & Statistics



Coding fundamentals

* Programming Language
  - Python
    - SnakeX
    - Circus
* 1980's
  - ABC Language
    -
* 1995 -> Python -> Circus

L) Solve some problems

L) Code it solve

Client:

Python L) IOT device

↑ → C++

* Understand the problem

* Questions → Requirement Analysis

* Propose - Strategies

A -

B -

* Choose the optimal solution

* ~~Option~~ Code
* Python
* Java
* C++, C,

Problem : finding greatest number in a given list

Solution : Ambiguity (confusion)

- Length of the list ?
- Integer Numbers / floating
- Empty list → -1, 0, NULL

Ambiguous

PS finding the greatest number ?

Requirement

1. List
2. Length of the list =
3. If no number are given.
4. Range of the list int -127 to +127
5. Floating / Decimal number
6. Is the number unique / duplicate number.
7. Which program language I can use
8. Ascending / Descending Order
9. Length of a list

C prog

Py.

* No Empty list
* Integer numbers
* 1 to 10,000
* No duplicates
* C programming

Re A Completed

Strategies / Algorithms

list = [10, 20, 40, 50, 100, 5, 4]

0 1 2 3 4

len = 5

largest = list[len-1]

Largest Number

max = 10

list[0] = 10

max = 20

list[i] = 20

max <-> 40

max = 40

max = 100

Worst case scenario

len O(1)

O(i)

length = 10 / 20

Length of the list = 10000000

[1 , 1 , - - - - - - - - - -] n = 10,000000

O(n) 1 million times

O(1)

=

C, C++:

Algorithm!

- for loop
- while loop

Sol 1: Loop // for loop

2: // while

1 Sorting

- Recursion
- Comp

3: 1 Sorting, last num

Ascending.

4: Recursion?

5:

Algorithm: find the largest [for loop]

Inputs: list = [10, 100, 2030], n = 4
integer number

Output: largest number.

Step 1: max = list[0] // max = 10

Step 2: for i ⟹ 1 to n-1 // i++
if max < list[i]
max = list[i];

Algorithm: find the largest [for loop]

Inputs: list = [ ], n = 0
integer number

Output: largest number.

Step 1: if n == 0
return empty [-1]

Step 2: max = list[0]

Step 3: for i -> 1 to n-1
if max < list[i]
max = list[i]

Step 4: Return max

Algorithm : find largest number [while loop]

Inputs : list = [ ], n

Step 1: if n == 0
return 'empty'

Step 2: max = list[0]

Step 3: i = 1

Step 4: while max < list[i]
max = list[i]
i = i+1

Step 5: return max

for loop / while loop

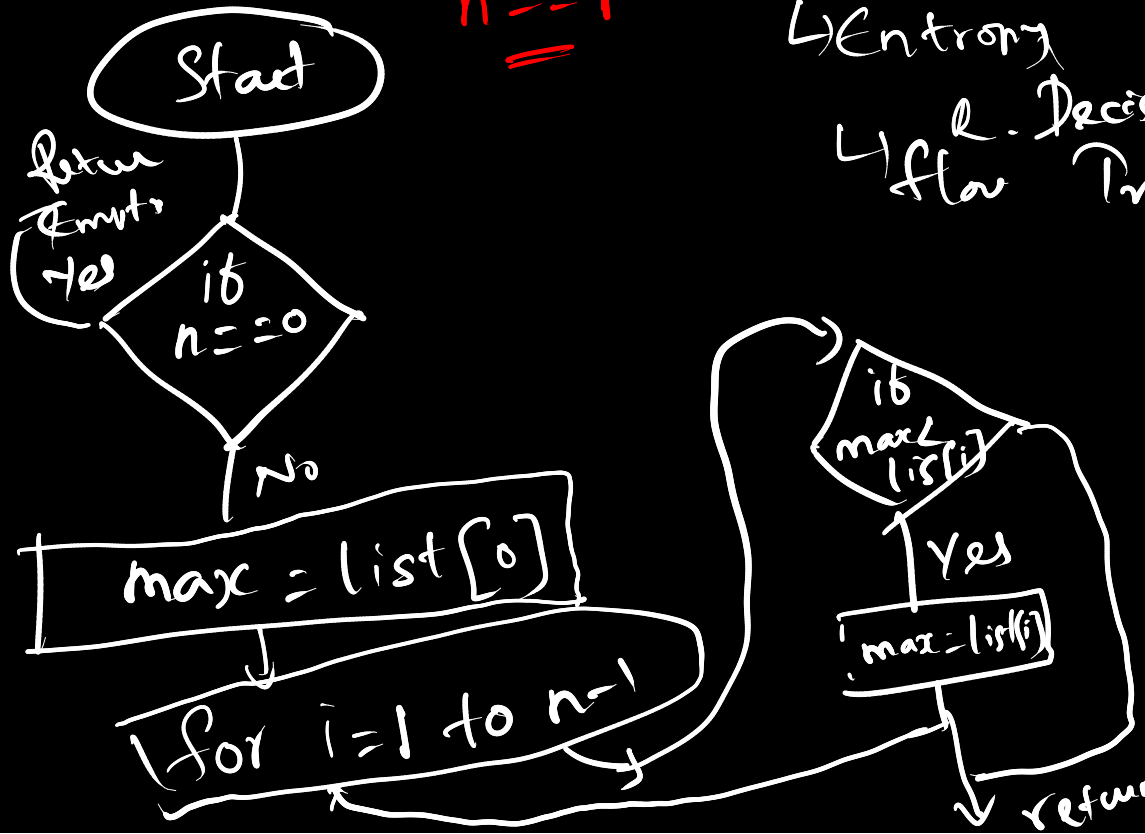
n determines

n is unknown

for loop

Flow Diagram :

n==1



Machine Learning

- Entropy
- R. Decision
- flow Tree

C program

```
#include<stdio.h>
int main(){
    int list={10,20,100,50}, max;
    int i;
    n=4
    if(n==0){
        return -1; // Empty
    }
    max=list[i]
    for(i=1; i<n ; i++){
        if(max<list[i]){
            max=list[i];
        }
    return max
```

Pseudocode

```c
#include<stdio.h>
int main(){
	int list[4] = {10, 20, 500, 5};
	int max, i, n = 4;
	if(n == 0){
		printf("Empty list");
		return 0;
	}
	max = list[0];
	for(i=1; i<n; i++){
		if(max < list[i]){
			max = list[i];
		}
	}
	printf("max = %d", max);
	return 0;
}
```

Analyse

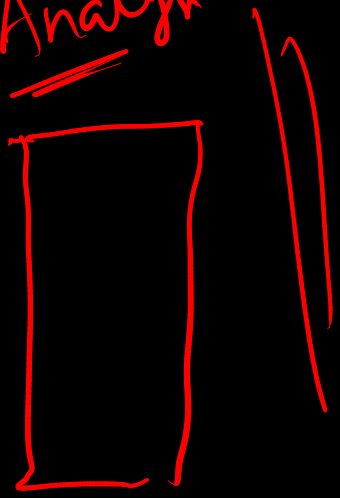
- Errors
- Boundary Conditions
- Empty list n==0
- n==1

[-127 to +127]

Time Complexity & Space Complexity.

```
int main(){
    int a = 20, b = 40;
    if (a > b){
        printf("a is greater");
    }
    else
        printf("b is greater");
    return 0;
}
```

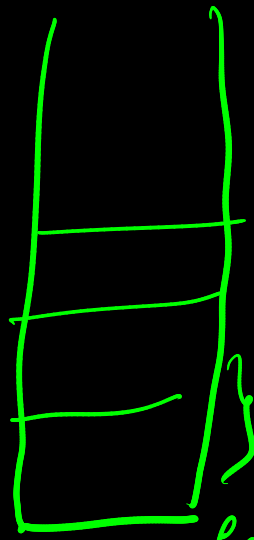
Analysis

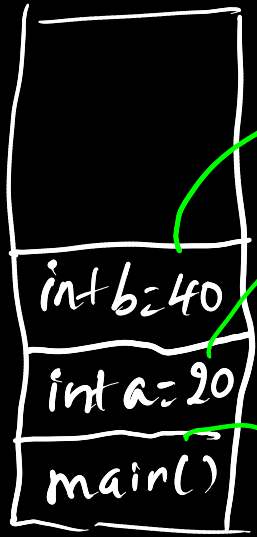


RAM

Stack Memory

Heap Memory





pop out

4 bytes pop out

2 bytes [32 address]

4 bytes

2 bytes [32 address]

pop out

Empty space

4 bytes = Memory Complexity

Space Complexity 4 bytes

```
int main(){
    int a=10, b=30, max;
    if(a>b){
        max=a;
    }
    else{
        max=b;
    }
    printf("%d", max);
    return 0;
}
```

Total time Complexity

4 units

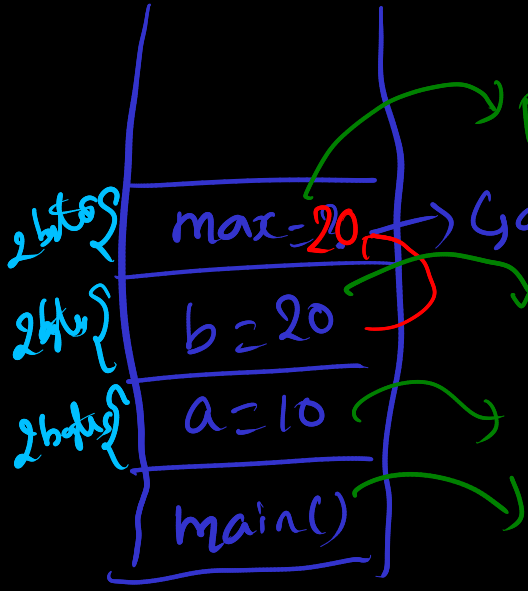
$O(4)$ units

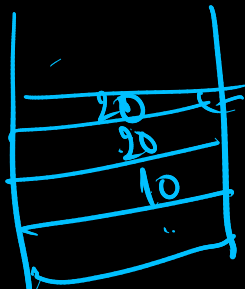
$O(c)$ (constant)

$O(1)$

Time Complexity

Assignment	Unit
=	1
> >= <= != ==	1
a[0]	1





Garbage value

↳ Assembly Programming

Output

max = 20

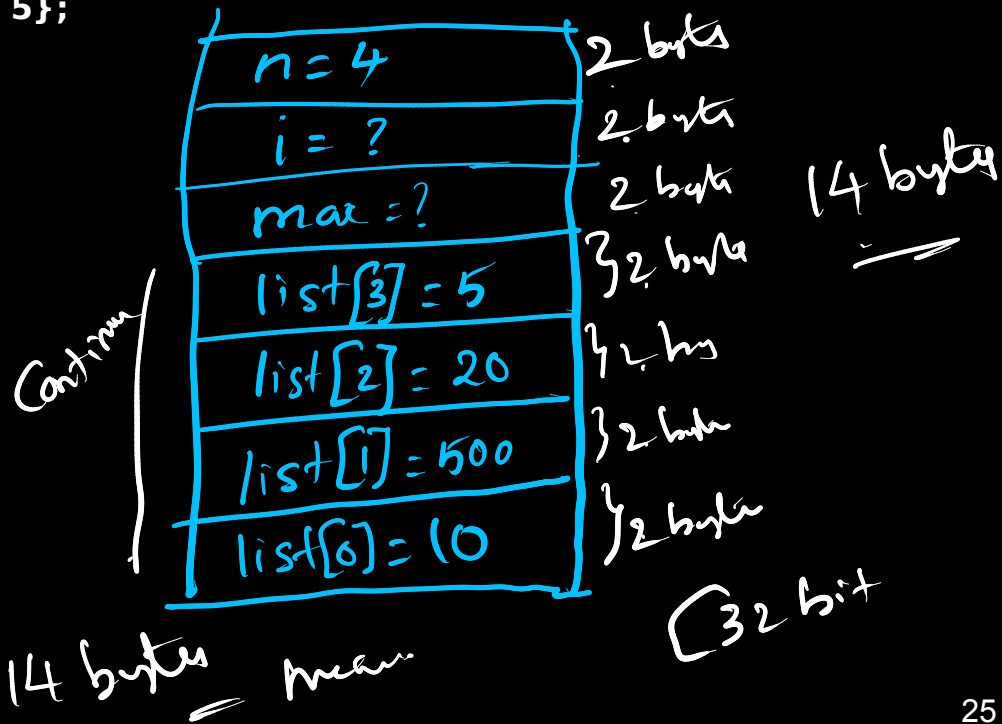
Space Complexity

4 bytes

Time Complexity

Space Complexity

```
#include<stdio.h>
int main(){
      long int list[4] = {10,500,20, 5};
      long int max, i, n = 4;
      if(n == 0){
            printf("Empty list");
            return 0;
      }
      max = list[0];
      for(i=1; i<n; i++){
            if(max < list[i]){
                  max = list[i];
            }
      }
      printf("max = %ld", max);
      return 0;
}
```



Time Analysis

```
#include<stdio.h>
int main(){
    long int list[4] = {10,500,20, 5};
    long int max, i, n = 4;
    if(n == 0){
        printf("Empty list");
        return 0;
    }
    max = list[0];
    for(i=1; i<n; i++){
        if(max < list[i]){
            max = list[i];
        }
    }
    printf("max = %ld", max);
    return 0;
}
```

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

max=list[i]	2	n-1
max<list[i]	2	n-1 ?
list[4]	4	4
n=4	1	1
n==0	1	1
max=list[0]	2	1
i=1	1	1
i<n	1	n
i++	2	n-1

i=2, i=3, i=4

1<4, 2<4, 3<4, 4<4

```
int main(){
int list[5] = {5,10 15, 20, 25}
int n=5;
int i;

for(i=0; i<n; i++){
    printf("%d", list[i]
}
return 0;
}
```

$f(n) = 4n+3$

i = i+1

list[5]	5	1
n=5	1	1
i=0	1	1
i<n	1	n+1
i++	2	n

0 < 5
1 < 5
2 < 5
3 < 5
4 < 5
5 < 5

i=1
i=2
i=3
i=4
i=5

$f(n) = n \times 1 + 1 \times 1 + 1 \times 1 + 1 \times (n+1) + 2n$

$= n+2+n+1+2n$

$4n+3$

Solution Compare

Algorithmic Terms

Space Complexity / Time Complexity

A -) Solution 1 ⟶ better in Space [less space]

B -) Solution 2 ⟶ better in Time [less time]

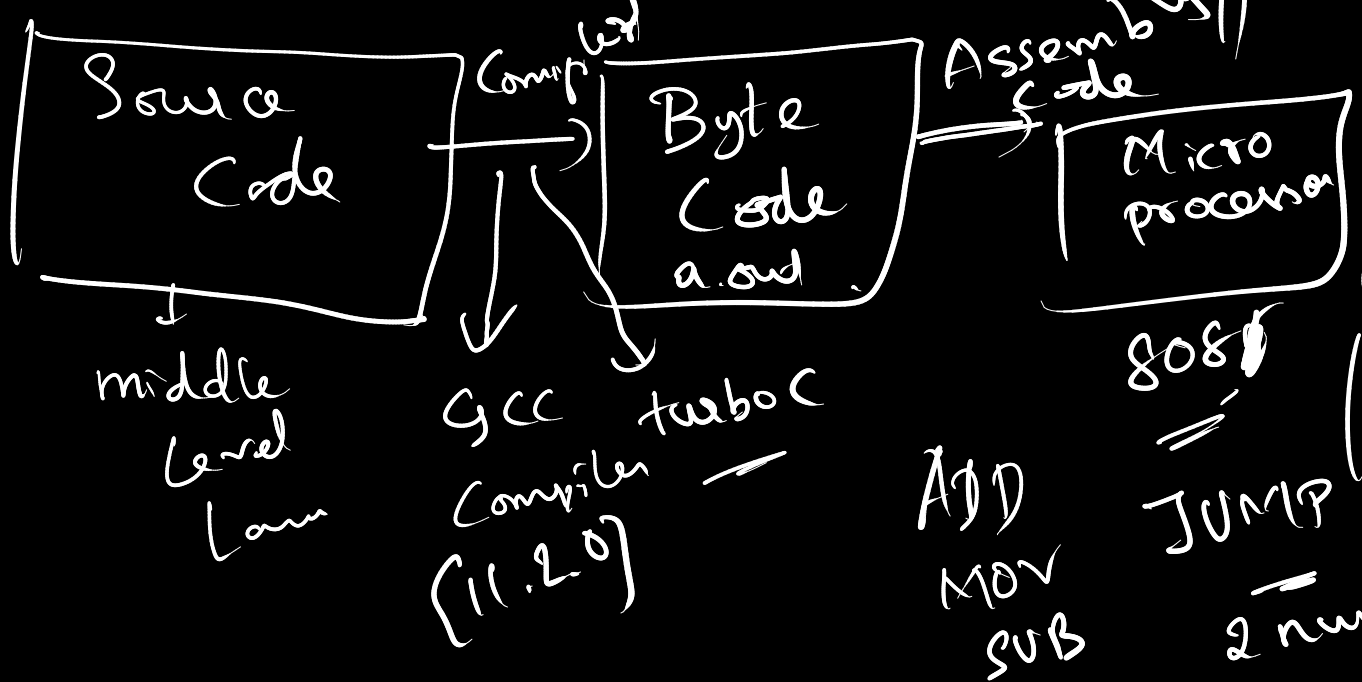
↳ Accept ? // IOT devices Space vary time or Time Micro -) Accept Solution 2

Tradeoffs

Space & Time // Application

Programming Langs.

C program (Platform dependent)



Exercise 1: Addition of 2 numbers

2: Subtraction of 2 number.

3: Multiplication of 2 numbers

4: Division of 2 numbers

Assembly 8085

Time

Space

Machine

```
int a=10
int b=20
int c=a*b
```

.005% error tolerance

a=10+10+

20 add:

20+20+20 (10 times)

(-) Reduction of time

Java Programming

- Platform independent
  - Execute a source program [Ubuntu]
    - Executable program → [Windows]
      - Does not work.
  - GCC compiler Different
  - Turbo Compiler

Java Virtual Machine

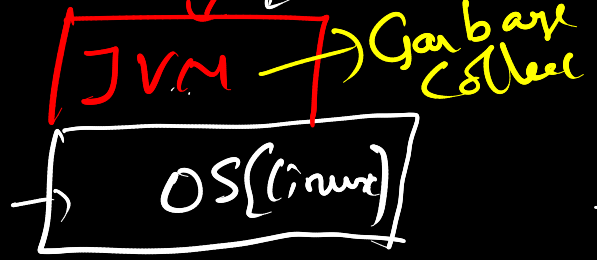
JDK → Source → Executable Code

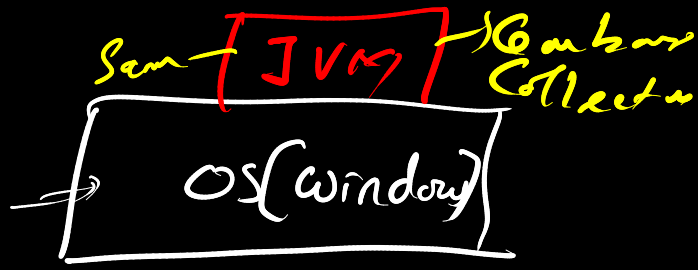
JVM →

Source program

Platform independent

Source Code
↓
Executable Code
↓
JVM
↳ bytecode



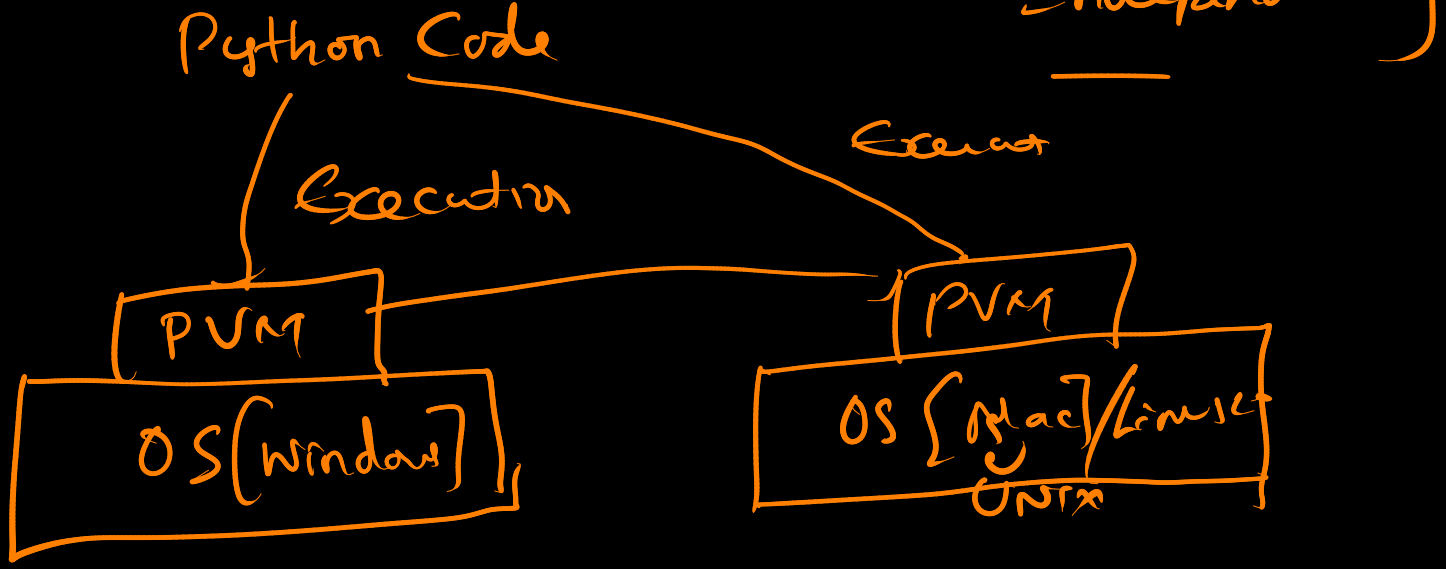


Time higher

→ Assembly → OS

Executable Prog → JVM → OS

Python Program [Platform Independent]



↳ JVM & PVM →

GITHUB

=

=

↳ Recursion function

=

↳ Complex Problem

↳ Coding Standards

break statement

↳ If

↳ else

↳ Looping → for loop

↳ while loop

Recursion

↳ function which makes a copy & calls the copied function

↳ Which calls itself [Wrong definition]

function [C programming]

```
int greatest(int a, int b){
    if (a >= b){
        return a;
    }
    else return b;
}
```

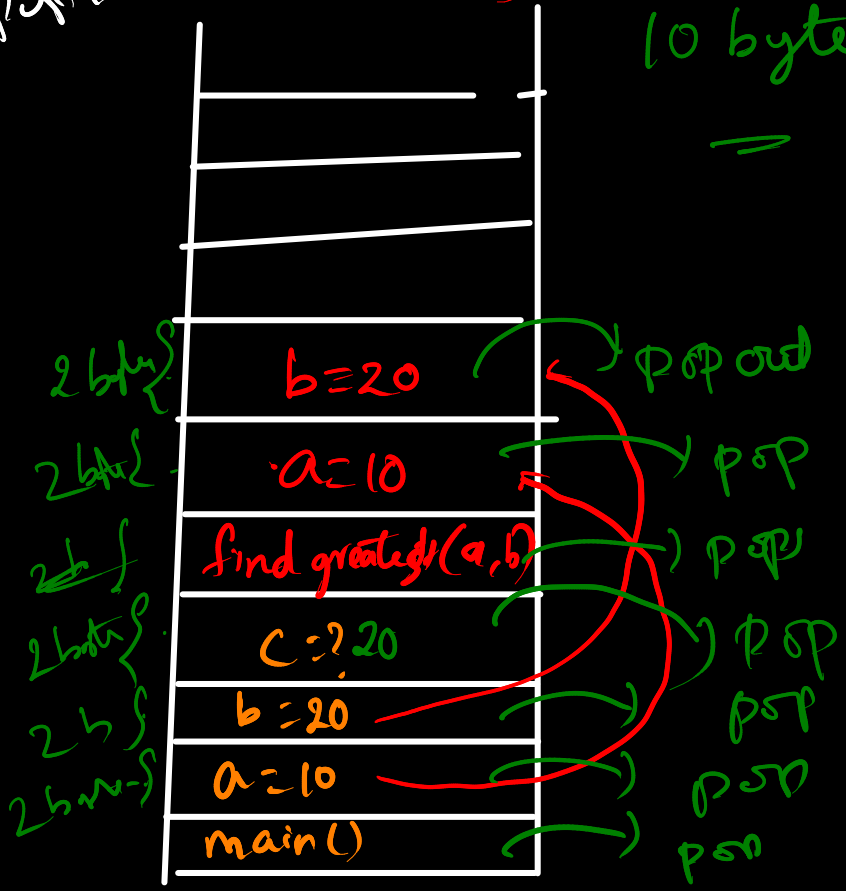
```
int main(){
    int a=10, b=20;
    int c;
    c=greatest(a,b);
    printf("%d",c);
    return 0;
}
```

C Program Path

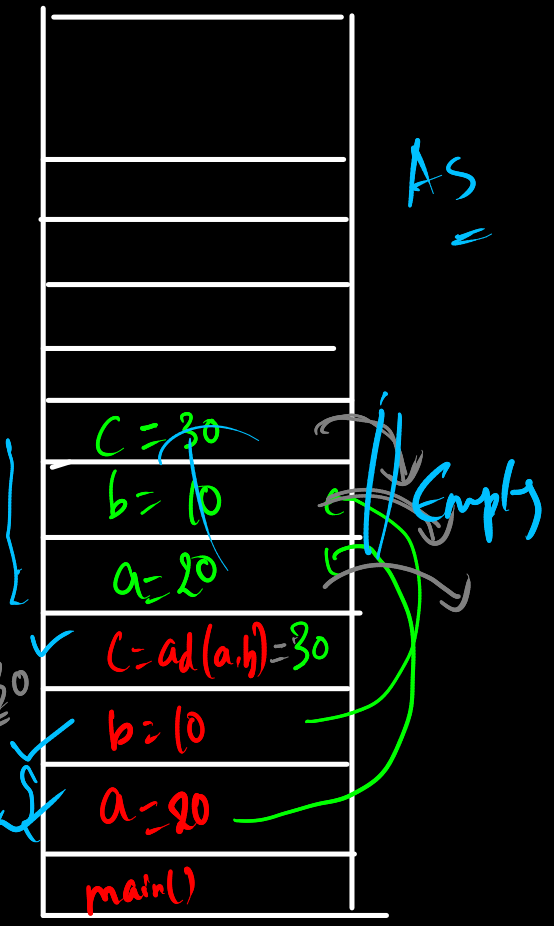
Stack m/y

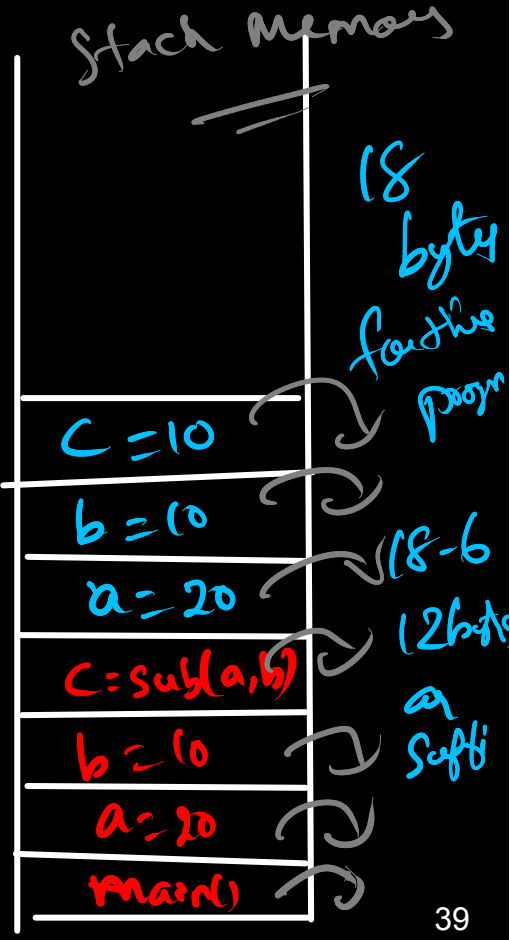
10 bytes

```
#include<stdio.h>
int findgreatest(int a, int b){
        if(a>b){
                return a;
        }
        return b;
}

int main(){
        int a = 10, b = 20, c;
        c = findgreatest(a, b);
        printf("%d",c);
        return 0;
}
```



```c
#include<stdio.h>
int add(int a, int b){
        int c;
        c = a+b;
        return c;
}
int sub(int a, int b){
        int c;
        c = a-b;
        return c;
}
int main(){
        int a,b,c;
        a = 20;
        b = 10;
        c = add(a, b);
        printf("addition %d", c);
        c = sub(a, b);
        printf("sub %d", c);
}
```





Recursion → Decision Tree Randomized

L) Why we use recursion / Purpose

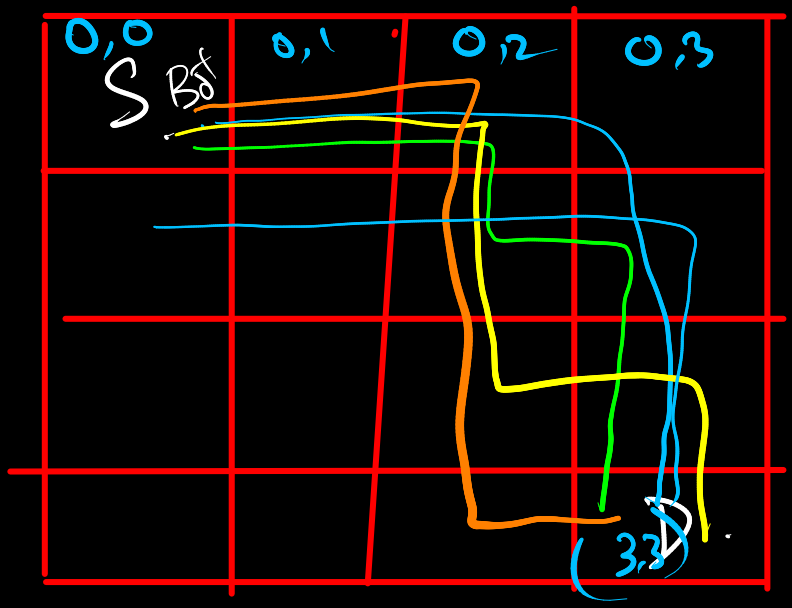
- L) Reduce the code ✗
- L) Time Complex ✗
- L) Space Complex ✗

L) Routines

L) Simpler Code

L) Repeatation
↳ Re

Maze



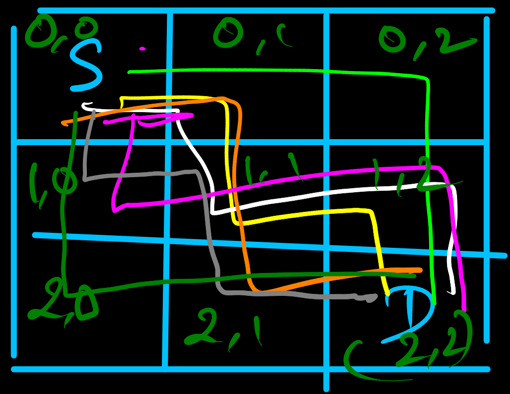
- Right/Down
- No diagonal

for loop?
while loop

Number of paths

[0,0
2,2]

Recursion ⇐



→ Right/Down

7 paths

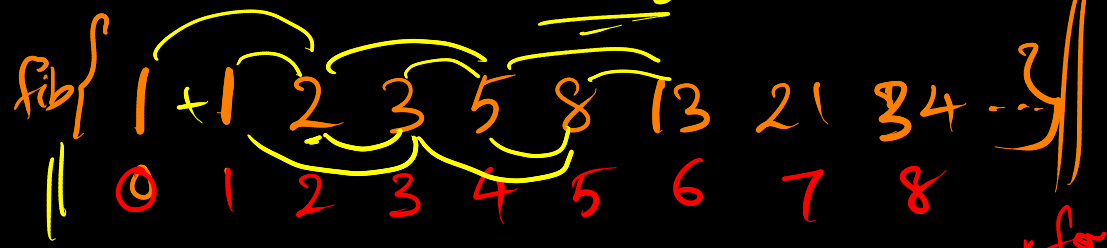
|| for (Difficu
while loop?

× Try it

Fibonacci Series

Pic

Story: Recursion



for Recursion

Business
2 goats
3 goats
5 goats
8 goats
13 goats

Factorial Problem

input: 4! = 4 x 3 x 2 x 1

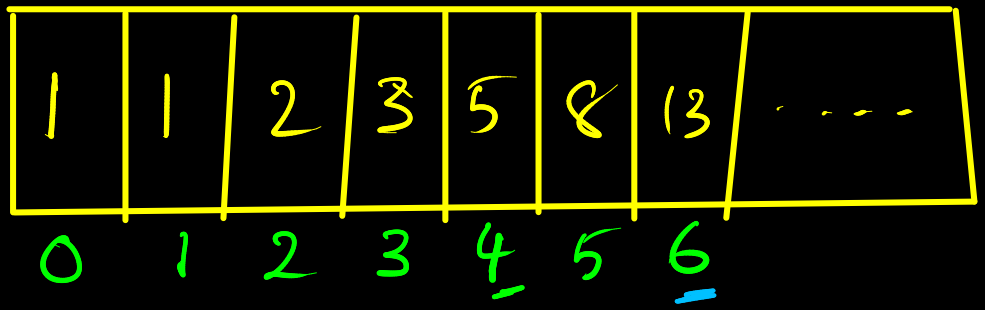
5 = 5 x 4 x 3 x 2

4 * (4-1) * (4-2) * (4-3) ... for while Recursion

Fib

=

Recursion



Problem:

=

Position : 4 = 5

6 = 13

Can you write a recursion program

Factorial! input 5

output $5 \times 4 \times 3 \times 2$

$20 \times 3 = 120$

input 4 =

$= 24$

Client input n ⊂ factor

↳ factor' Recursion

Recursion

1. find out termination code / base condition / Exit Condition
2. call the function duplicate
3. Simplify the dataset & call it

Wrong Definition

Recursion will call itself

Algorithm: fact n: n = 1

- if 0! = 1
- if 1
  - return 1
- if 2
  - return 2
  - 3 * 2 * 1

n! = ?

0! = 1

1! = 1

2! = 2

3! = 6

4! = 24

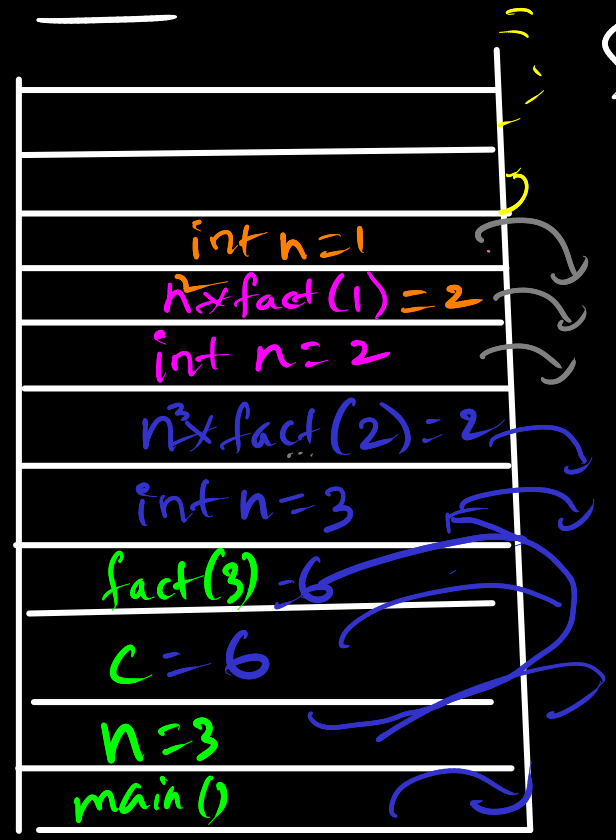
⋮

```
int fact(int n){
  if(n==0){
    return 1
  }
  if(n==1){
    return 1
  }
  if(n>1){
    return n*fact(n-1)
  }
}
```

```
int main(){
  int n=5
  int c;
  c=fact(n);
  printf c
}
```

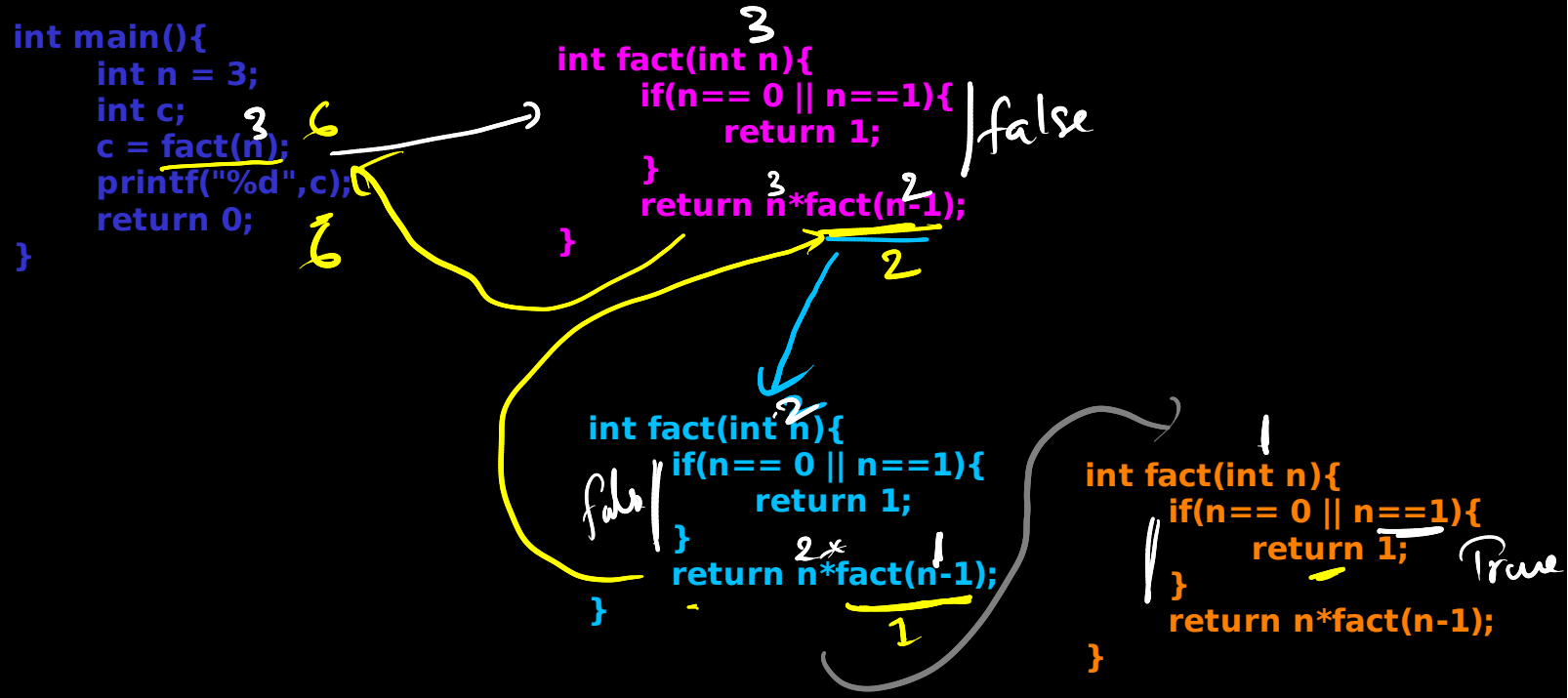
Recursion

```
#include<stdio.h>
int fact(int n){
        if(n== 0 || n==1){
                return 1;
        }
        return n*fact(n-1);
}

int main(){
        int n = 3;
        int c;
        c = fact(n);
        printf("%d",c);
        return 0;
}
```



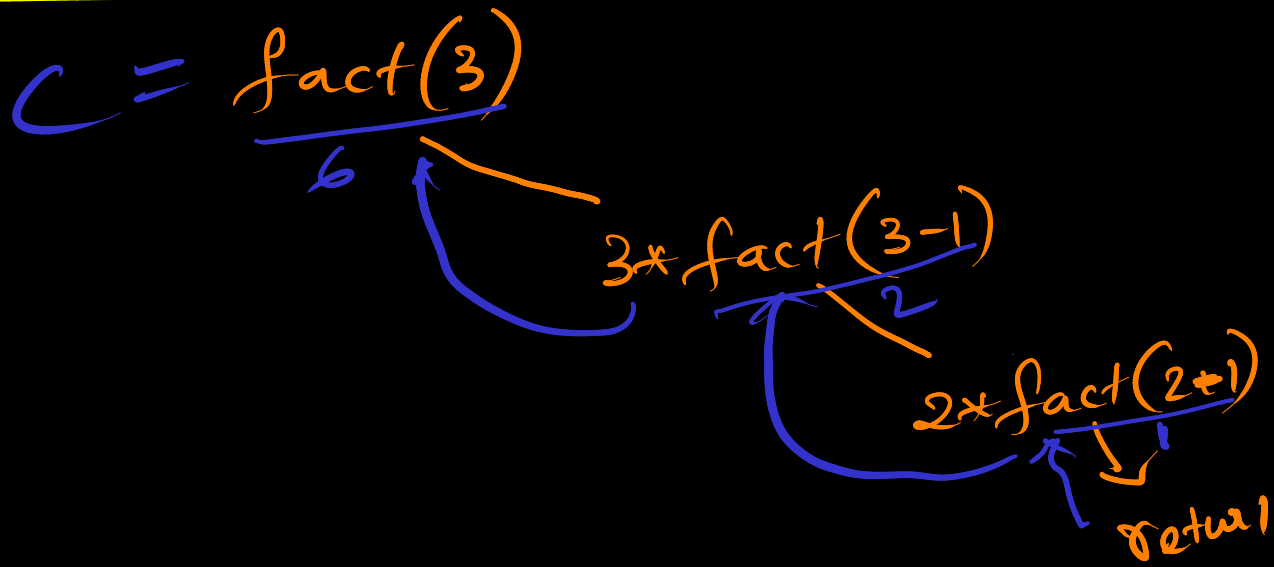
Stack memory

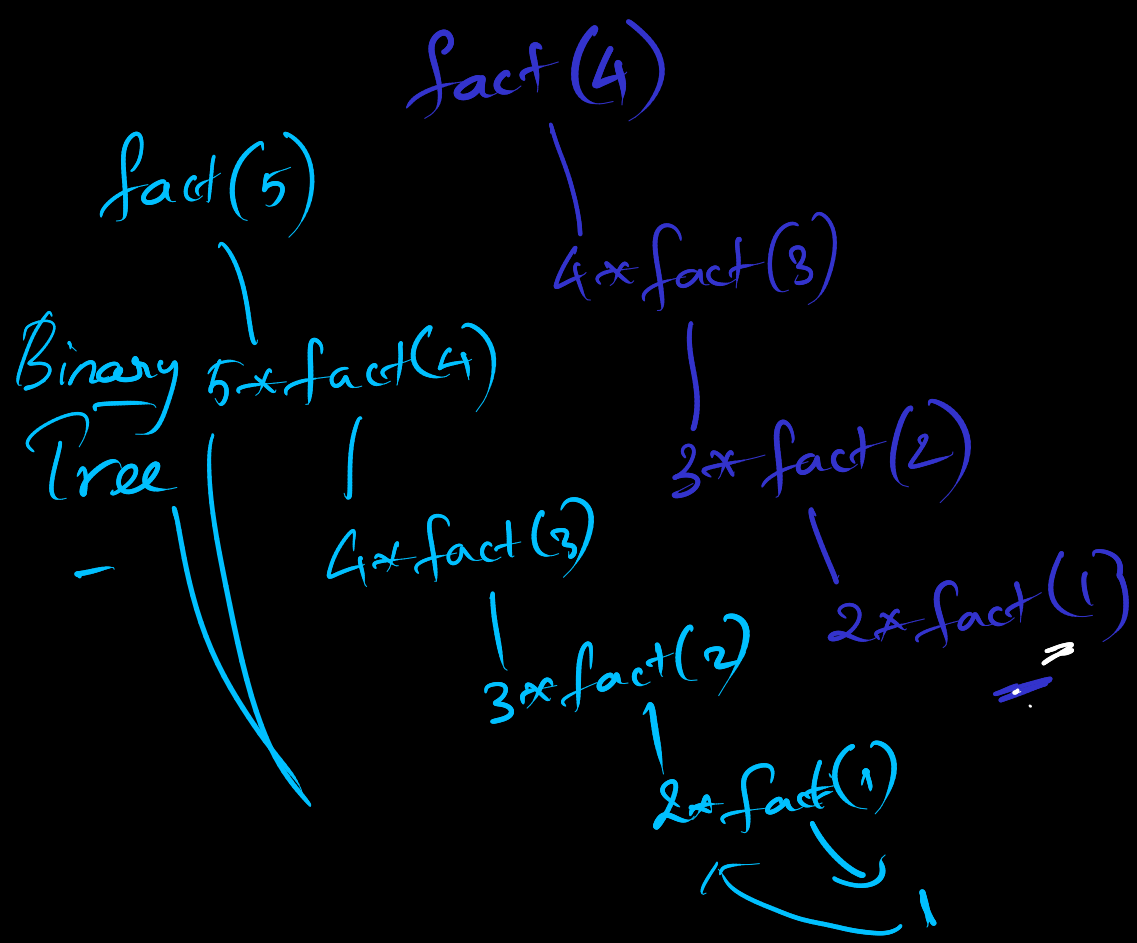
Heap memory


int main(){
int n = 3;
int c;
c = fact(n);
printf("%d",c);
return 0;
}
int fact(int n){
if(n== 0 || n==1){
return 1;
}
return n*fact(n-1);
}
int fact(int n){
if(n== 0 || n==1){
return 1;
}
return n*fact(n-1);
}
int fact(int n){
if(n== 0 || n==1){
return 1;
}
return n*fact(n-1);
}
false
false
True

n = 100 Computationally intensive

Tree Structure





Exit Condition:

Stack overflow

fibonacci Series.

1 1 2 3 5 8 13

1 2 3 4 5 6 7

n=

inp 5 → 5

inp 6 → 8

inp 7 → 13

Exit Condition

```
if n==1
    return 1
if n==2
    return 1
else:
    return fib(n-1) + fib(n-2)
```

```
int fib(int n){
    if (n==1 || n==2){
        return 1
    }
    return fib(n-1)+fib(n-2)
}
int main(){
    int n=4
    int c;
    c = fib(n);
    printf("%d", c);
    return 0;
}
```

Stack Memory

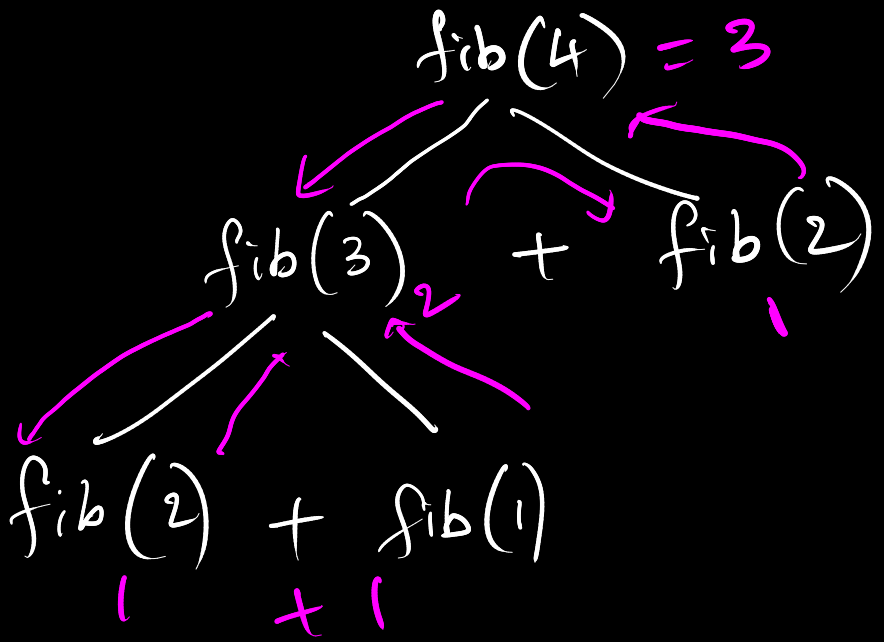
```
#include<stdio.h>
int fib(int n){
	if(n==1 || n == 2){
		return 1;
	}
	return fib(n-1)+ fib(n-2);

}

int main(){
	int n=3;
	int out;
	out = fib(n);
	printf("%d",out);
	return 0;
}
```

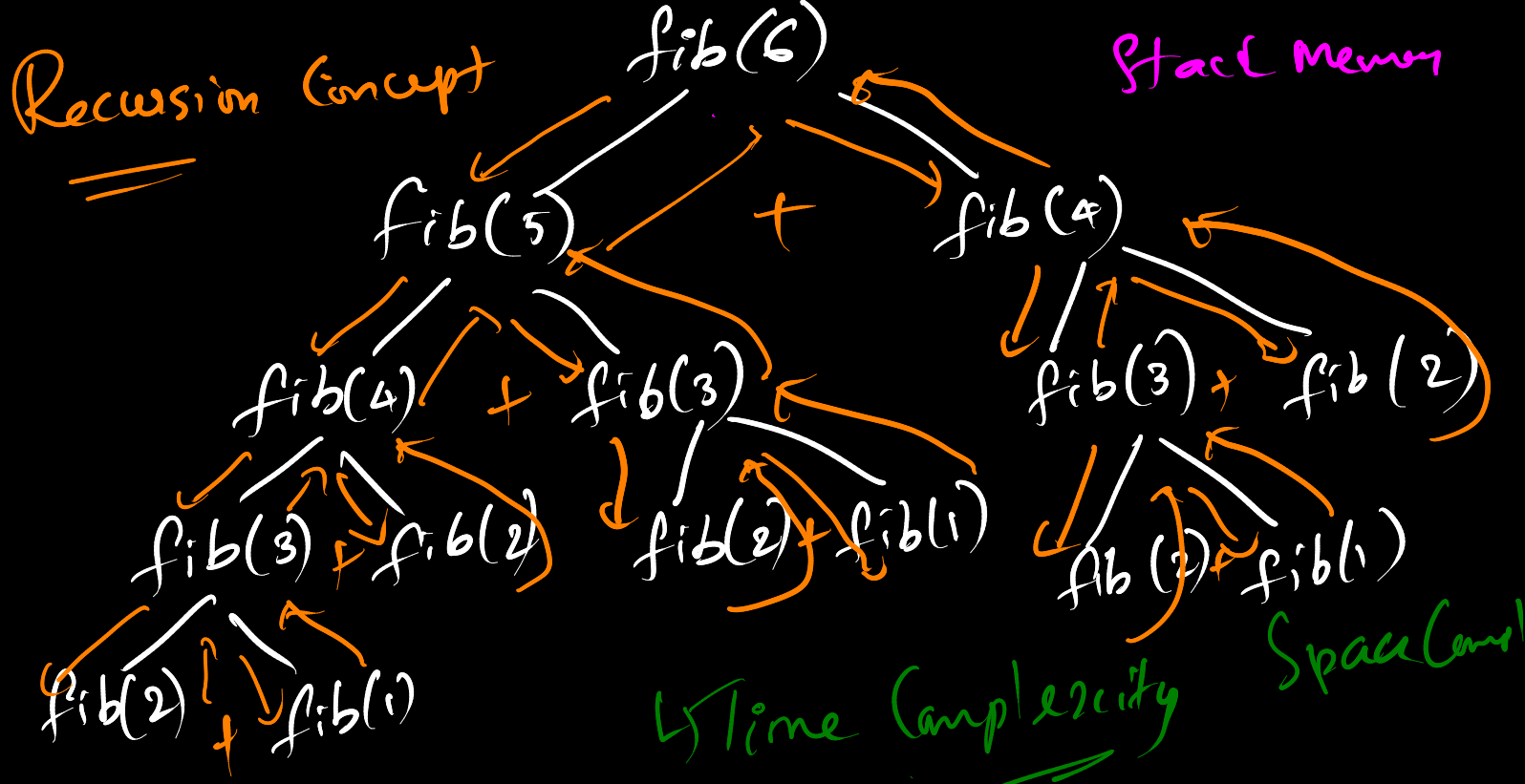
```
int main(){
        int n=3;
        int out;
        out = fib(n);
        printf("%d",out);
        return 0;
}
```

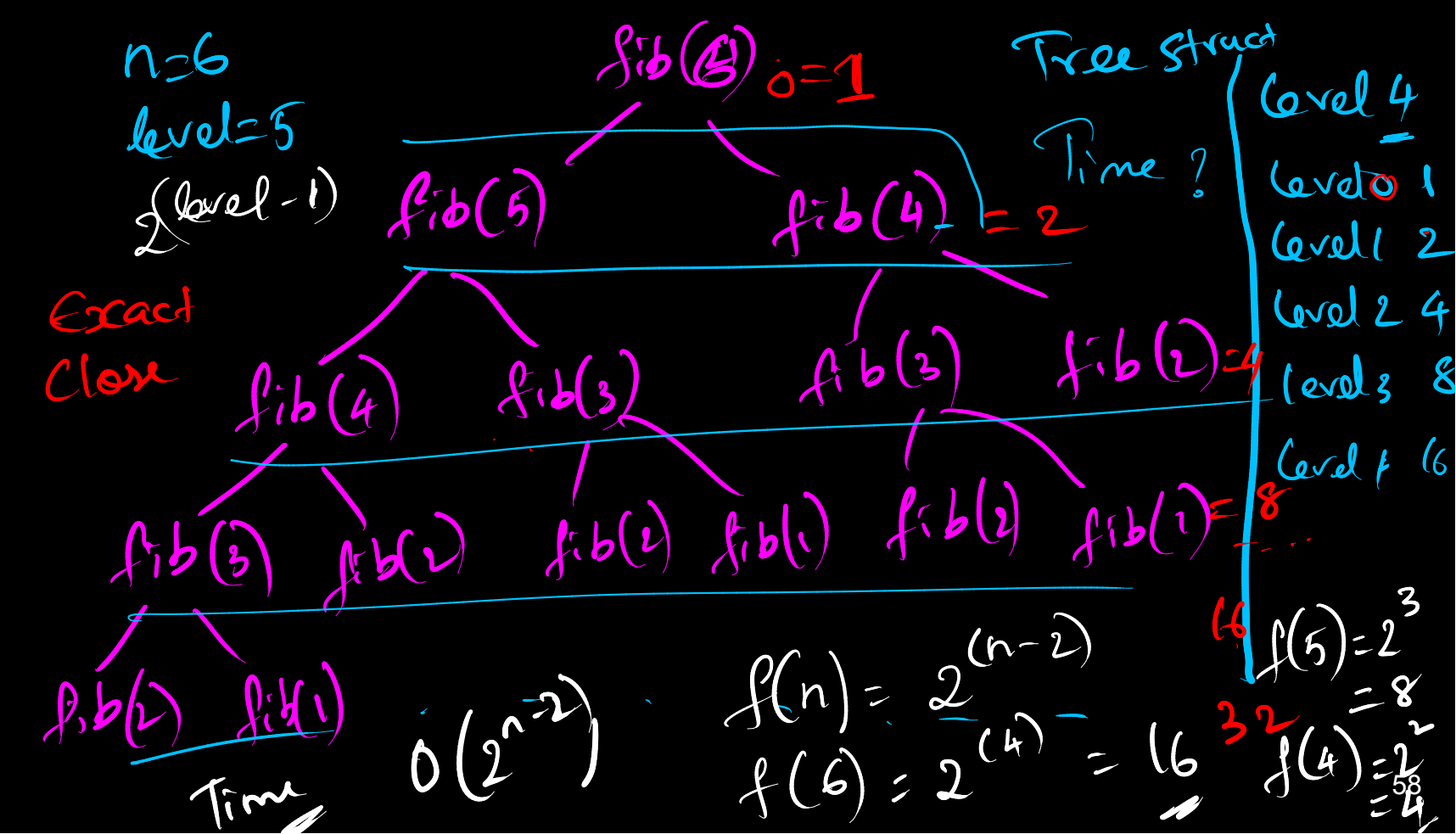
```
int fib(int n){
        if(n==1 || n == 2){
                return 1;
        }
        return fib(n-1)+ fib(n-2);
}
```

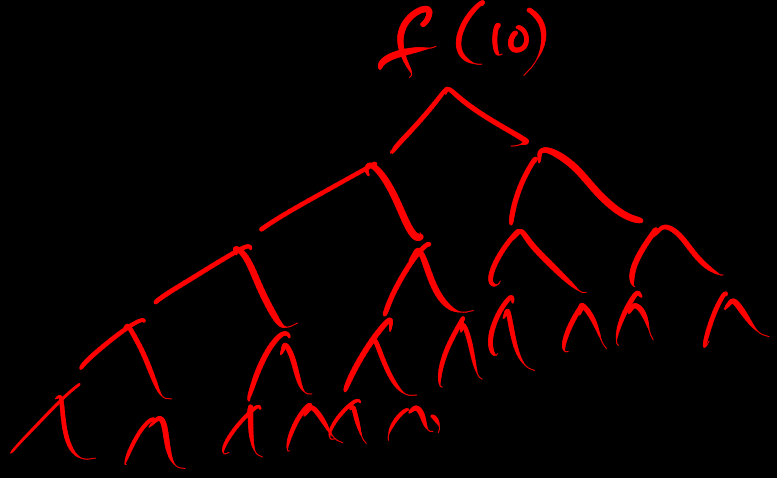
```
int fib(int n){
        if(n==1 || n == 2){
                return 1;
        }
        return fib(n-1)+ fib(n-2);

}
```

```
int fib(int n){
        if(n==1 || n == 2){
                return 1;
        }
        return fib(n-1)+ fib(n-2);

}
```


fib(4) = 3
fib(3) + fib(2)
2
1
fib(2) + fib(1)
1 + 1


Recursion Concept
fib(6)
Stack Memory
fib(5)
+
fib(4)
fib(4)
+
fib(3)
fib(3)
+
fib(2)
fib(3)
+
fib(2)
fib(2)
+
fib(1)
fib(2)
+
fib(1)
fib(2)
+
fib(1)
Time Complexity
Space Comp


n=6
level=5
2^(level-1)
Exact
Close
fib(6)
o=1
fib(5)
fib(4)
=2
fib(4)
fib(3)
fib(3)
fib(2)=4
fib(3)
fib(2)
fib(2)
fib(1)
fib(2)
fib(1)=8
fib(2)
fib(1)
Time
O(2^(n-2))
f(n) = 2^(n-2)
f(6) = 2^(4) = 16
Tree struct
Time ?
Level 4
Level 0 1
Level 1 2
Level 2 4
Level 3 8
Level 4 16
16
32
f(5)=2^3
=8
f(4)=2^2
=4
58



Level

0 - 1
1 - 2
2 - 4
3 = 8
4 - 16
5 = 32
6 = 64
7 = 128
8 = 256

9 = 512

$f(n) = 2^{(n-2)}$

$= O(2^{n-2})$ [Approximately]

Coding Standards:

C program

format the code

```
int main() {
    int a, b, c;
    for(----) {
        int d, e, g;
    }
}
```

Java Prog

```
public static void main(--) {
    Integer
    Float
    String
    for(---) {
        ---
        ---
    }
}
```

// classes objects

Python Pgm

```
-main-:
    a, b
    a=10
    a=10.5
    b="Amar"
```

Variable Names

~~int a, b;~~

Naming Conventions

C program

```
int firstvar, secondvar;
int output, additionvar;
```

Java Pgm

camel Case

```
Integer firstVar, seondVar;
Float addVal;
```

Naming the functions

C pgm

```
int addition(int firstarg, int secondarg){
    int outarg = firstarg + secondarg
    return outarg.
```

Java pgm

// Classes
obj

ClassName TitleCase

primitive int → Client

```
Integer Addition(Integer firstArg, Integer secondArg)
{
    Float out = firstArg + secondArg;
}
```

Python Program

↳ Standards C pgm

↳ Classes & objects

Indentation is important

↳ Curly Braces ✗

int ✗

float ✗

String ✗

a = 10.467

meaningful a = "amarnath"

b = "deloitte"

```
def addition(first, second):
    ans = first + second
    return ans
```

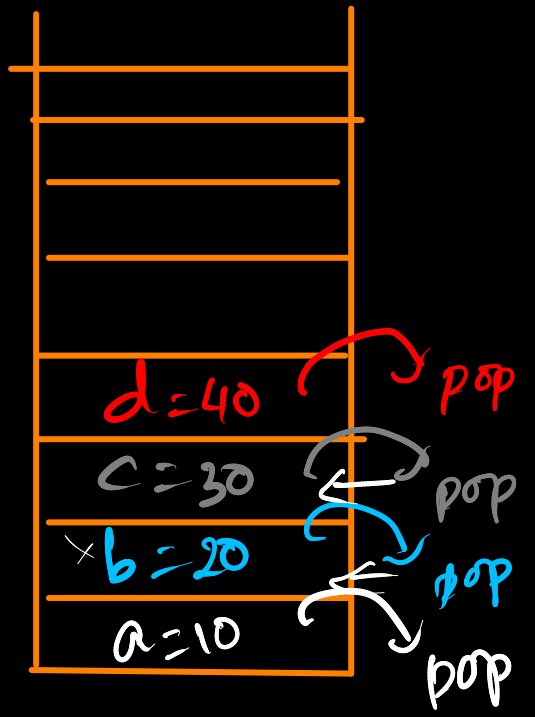
Stack memory.

↳ Local variables
↳ Global variables // Heap memory

↳ main function ← long exit
↳ cleared.

↳ local within a function
↳ Exit ← all variables declared inside the function will be cleared

Stack Data Structure



- LIFO manner
  Last In first Out
  - push() ← insertion
  - pop() ← deletion
  - peek() ← particular area

|| Queue Datastructure

|| Linked List → Double
↳ Circular LL
↳ Double Circular LL

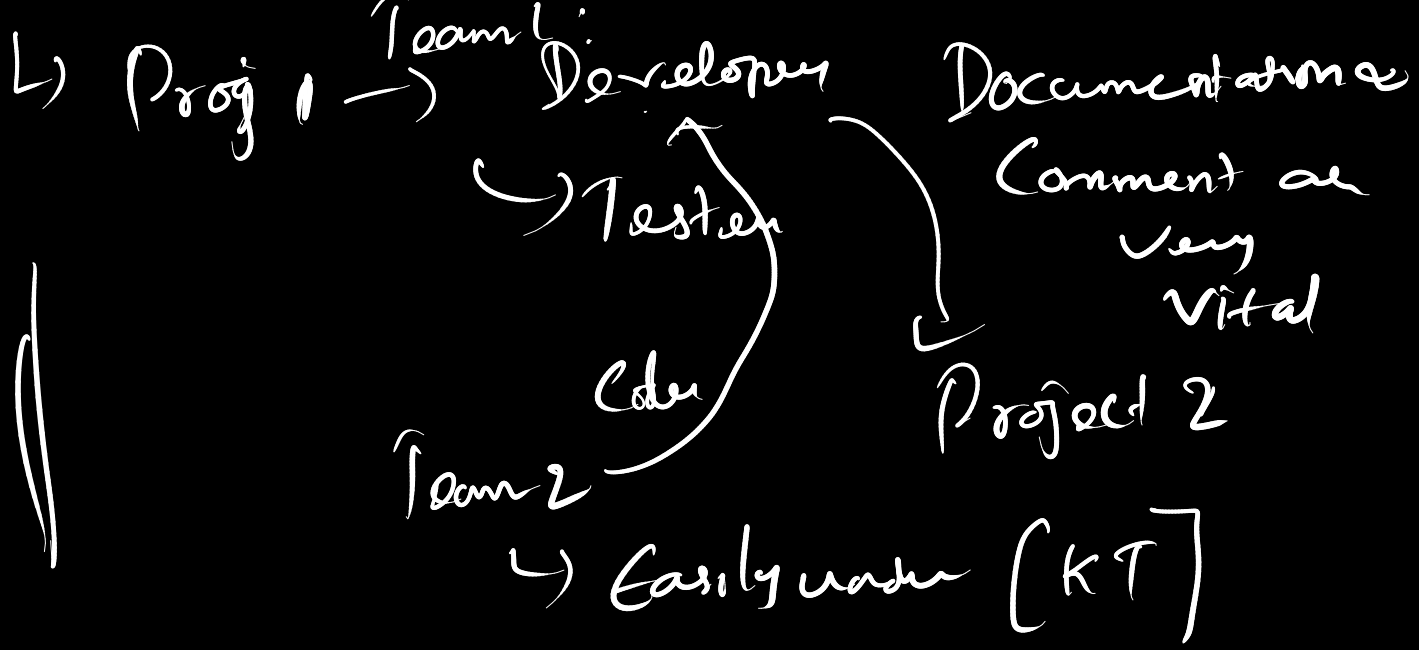
* Document
* Duplications ||

↳ Back at 4:15

↳ Cohesion
↳ Coupling ↳ Component Independent

Comment : Coding Stands

This function is for adding two numbers # input arg 2 fn
int add(int first, int sec){ # output → add
Multiline Comment int answer // this variable is temporary var.
Single line Comment // add two numbers
// This function addition of two numbers
return answer


Proj 1
Team 1:
Developer
Tester
Coder
Documentation
Comment as
very
Vital
Project 2
Team 2
Easily understand [KT]

Exception Handling / Error Handling

```
int divide(int a, int b){
    int c;
    c = a/b;
    return c;
}
int main(){
    int ans = divide(0,0);
}
```

(-127 to 127)

Negative num

Exception

if a==0 b=0 not --

Error

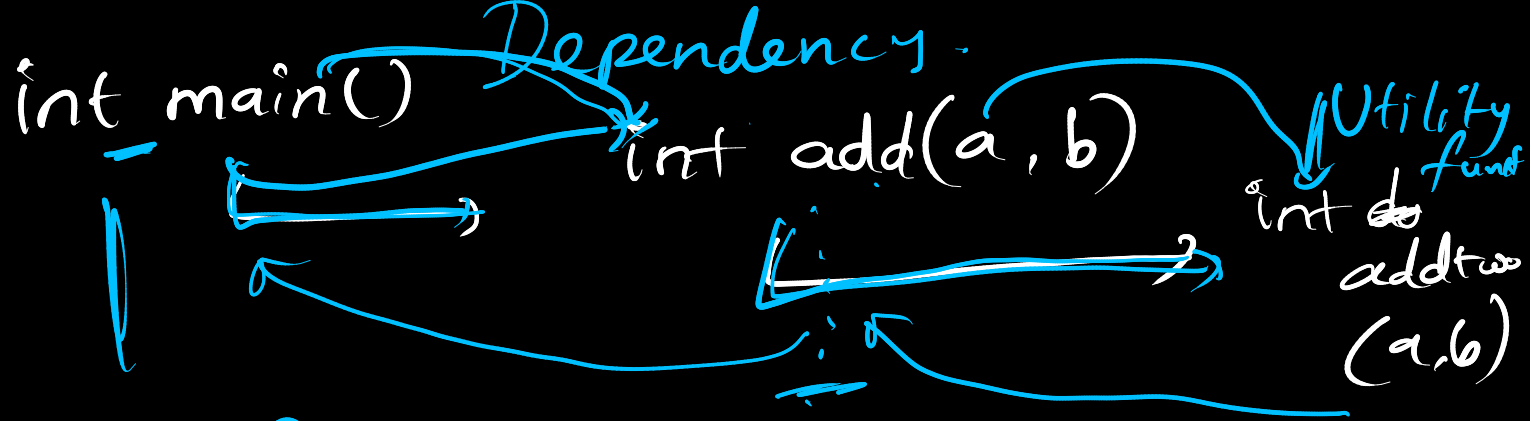
Python

Modularity:

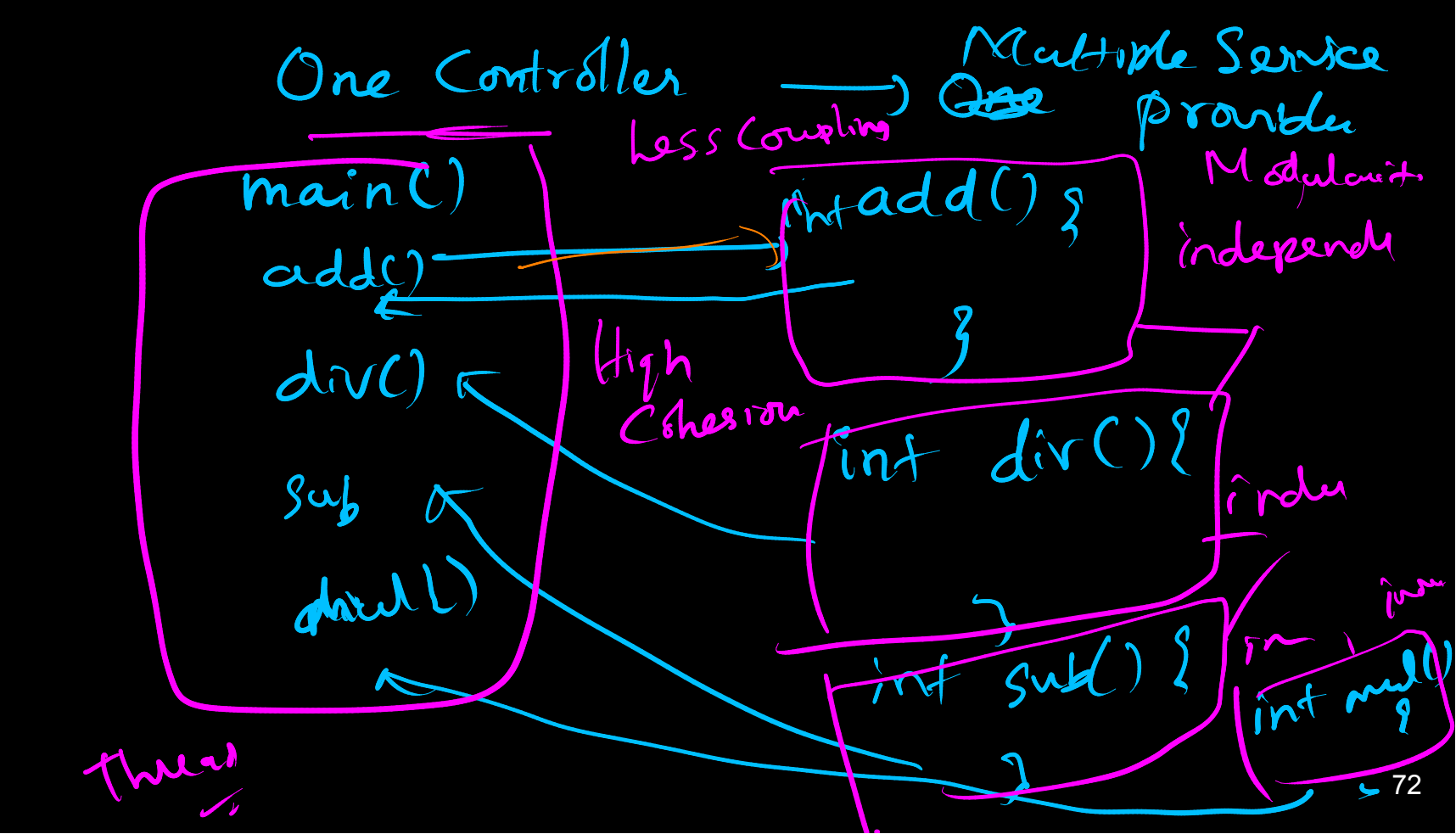
- Defining the functions
- Calling the functions

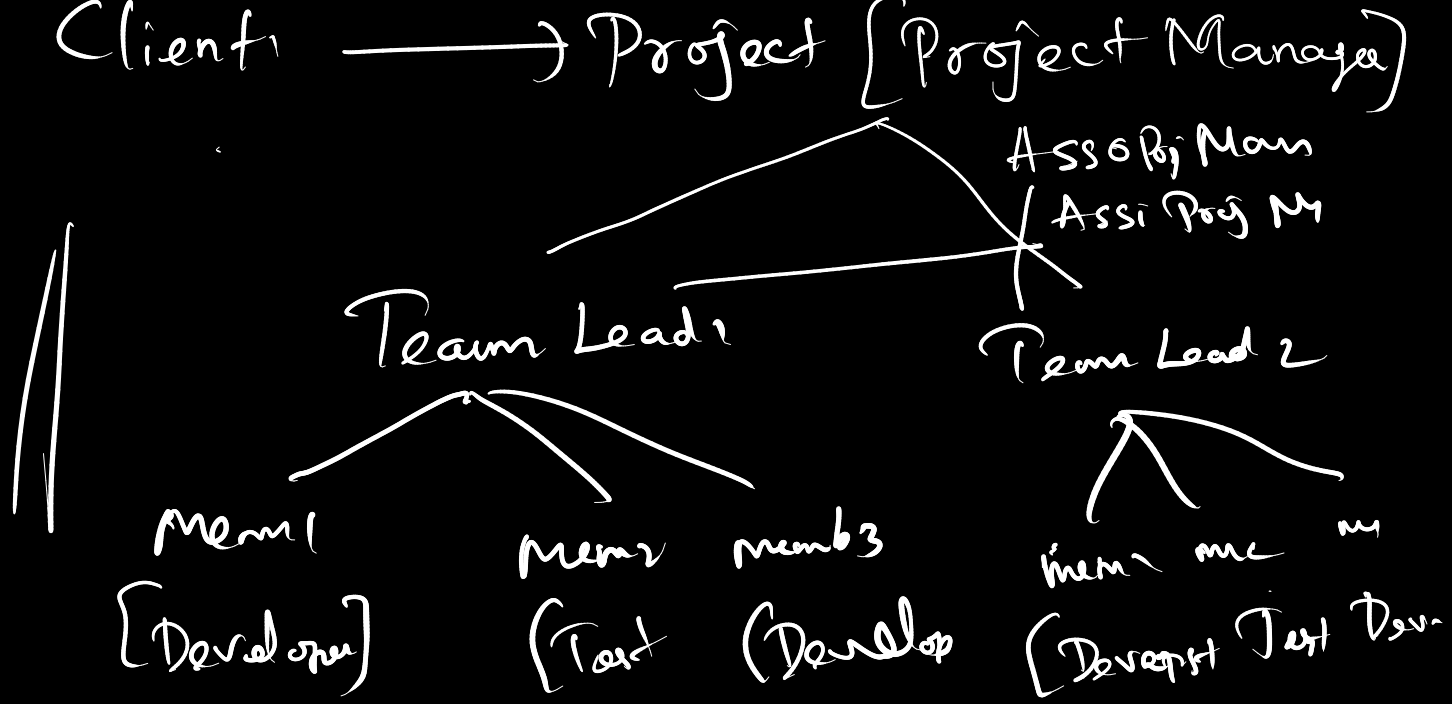
- Readable
- Stack Memory
  - Recursion
  - Local var
  - Global var

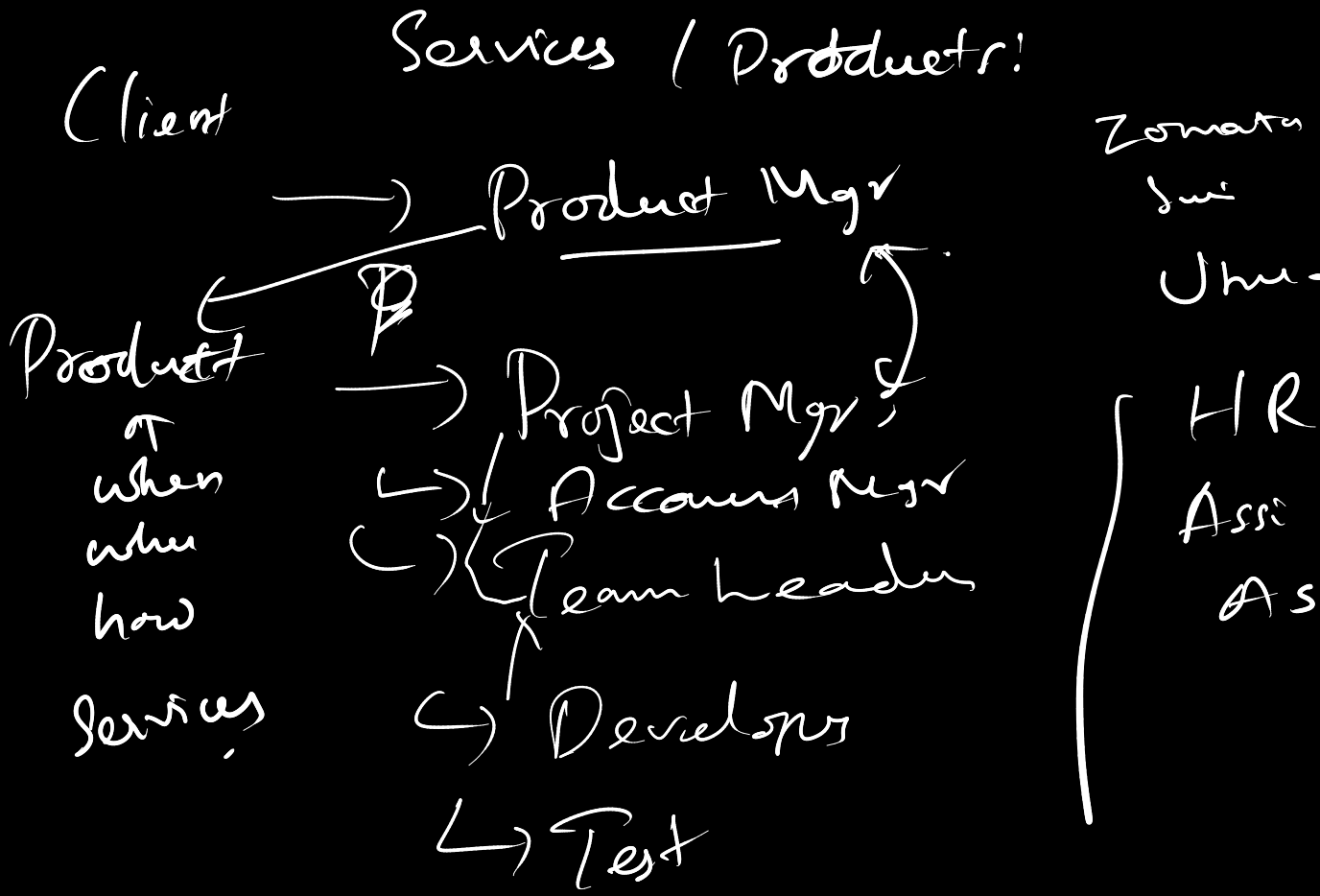
Cohesion & Coupling:



Reduce the dependency.


One Controller
Multiple Service provider
Less Coupling
Modularity
Independent
main()
add()
div()
sub()
mul()
int add() {
}
High Cohesion
int div() {
int sub() {
int mul() {
Thread


Client
Project [Project Manager]
Asso Proj Mgr
Assi Proj Mgr
Team Lead 1
Team Lead 2
Mem1
[Developer]
Mem2
[Tester
memb3
(Develop
[Developer Test Dev


Services / Products
Client
Product Mgr
Zomato
Swi
Uber
Product
when
what
how
Services
Project Mgr
Account Mgr
Team Leader
Developers
Test
HR
Assi
As

Thank You!

Note: This presentation is not for commercial purpose!