

COL100 Lecture 25

Note Title

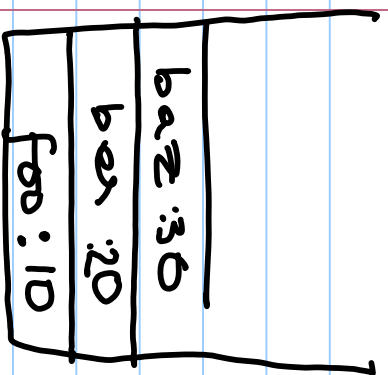
01-11-2018

Review: Recursion base case
3 rules of recursion

recursive computation of $n!$

```
int fact(int n)
{
    if (n == 0) // base case
        return 1;
    else // recursive case
    {
        int total = fact(n-1);
        total = total * n;
        return total;
    }
}
```

Function Call - stack



~~CDP~~
backtrace

baz() {
30 → return;
}

bar() {

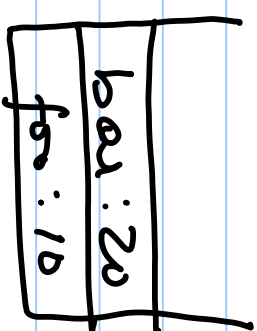
foo() {

{

10 → bar();

}

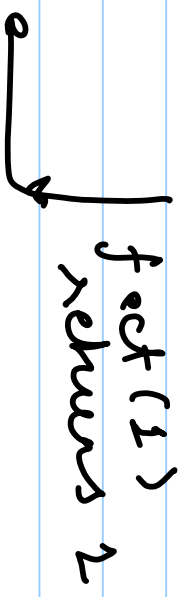
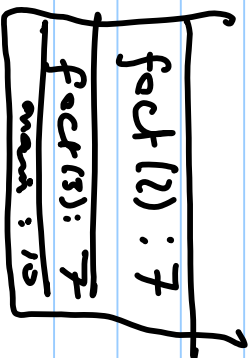
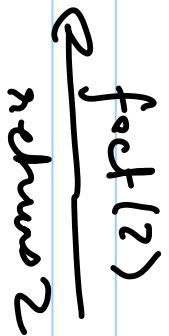
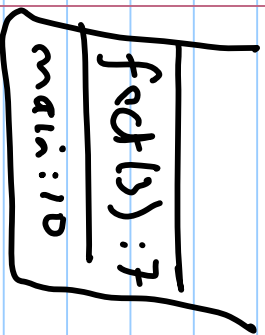
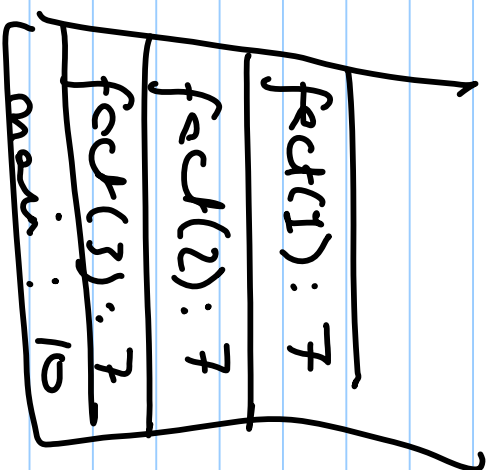
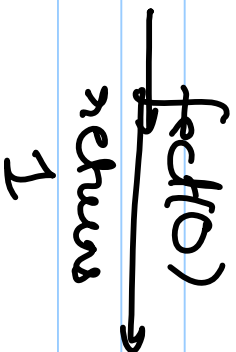
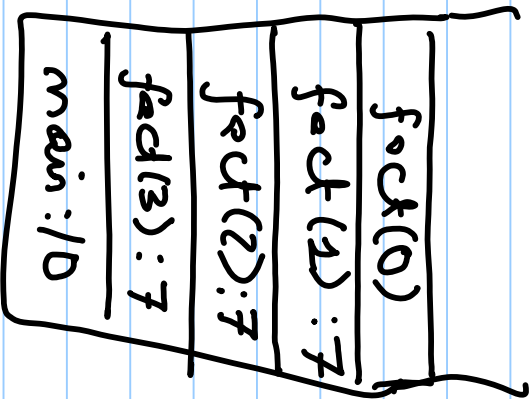
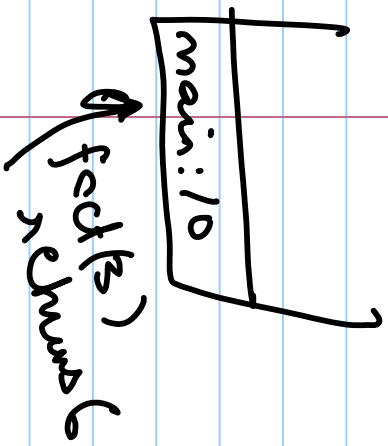
After bar returns



After baz returns



main
cout << fact(3) << endl;



```
int mystery (int n)
```

```
{  
    if (n < 10) {
```

```
        return n;
```

```
    } else {
```

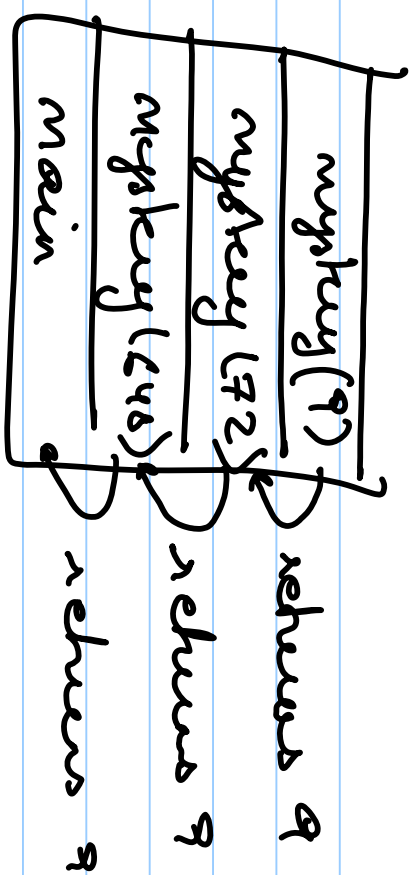
```
        int a = n / 10;
```

```
        int b = n % 10;
```

```
        return mystery (a+b);
```

```
    }
```

```
}  
  
cout << mystery (642);
```



main printo ✓

Palindrome

MALAYALAM



MADABAM

KACECAK

"STEP ON NO PETS"

"Q"


" "

"Java"

```

bool isPalindrome (string s) {
    if (s.length() == 0) {
        return true;
    } else if (s.length() == 1) {
        return true;
    } else if (s[0] == s[s.length()-1]) {
        return false;
    } else {
        return isPalindrome(s.substr(1,
            s.length()-2));
    }
}

```

is Palindrone ()

Recursion Big-O

Big-O computation is self-similar
i.e. can be expressed as

combination of
Big-O computation
of smaller programs

PSEUDO-CODE.

↳ rough sketch
of the code
(includes the
algorithm)

find Big O (code Snippet) :

if code Snippet is a single statement :

Recalls

else if code Snippet is a loop :

How many are in the

return

* number-of-times-loop-runs
find Big O (loop-body) ;

comparing

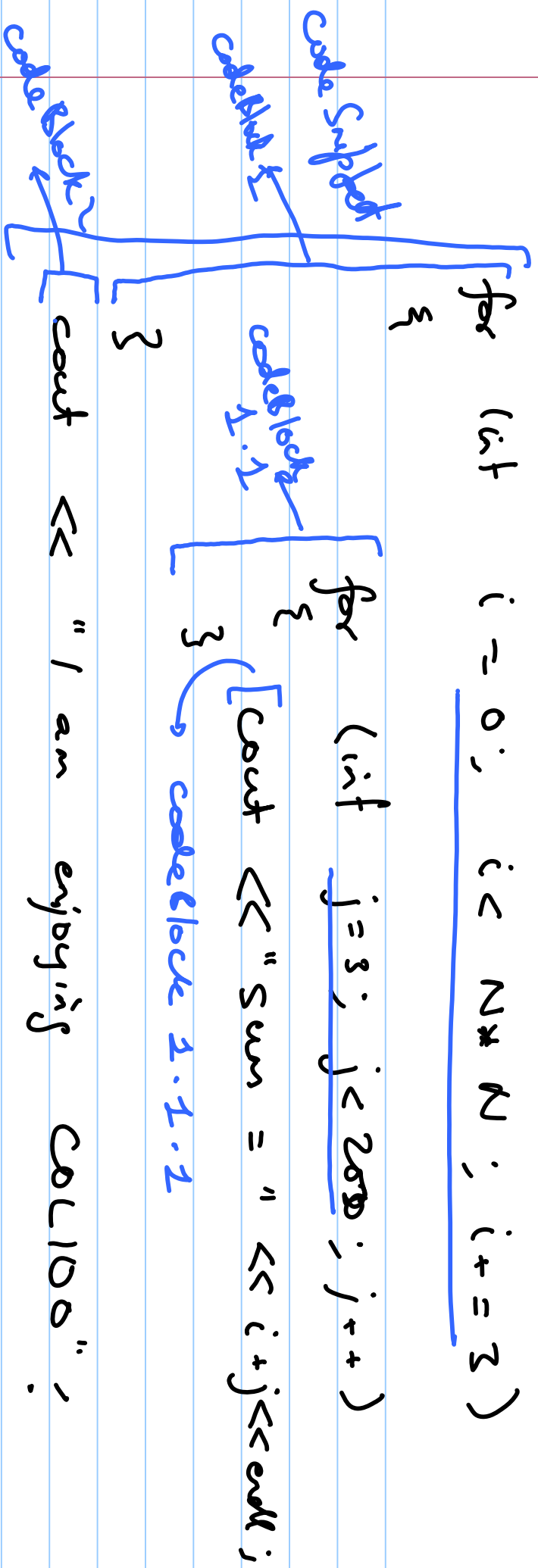
cost

Recursion
costs

else { // code Snippet = codeBlock1; codeBlock2; ...
for codeBlock in code Snippet :

return sum of find Big O (code Block) ;

}



find BigO (code Snippet) \rightarrow 3rd case

findBigO (code Snippet)

= findBigO (codeBlock1) + findBigO (codeBlock2)

= $O(N^2)$ * findBigO (codeBlock1.1)

+ findBigO (codeBlock2)

= $O(N^2)$ * ($O(2)$ * findBigO (codeBlock1.1.1))

+ findBigO (codeBlock2)

= $O(N^2)$ * ($O(2)$ * $O(1)$) + findBigO (codeBlock1)

= $O(N^2)$ + $O(1)$ + $O(1)$

~~9~~

x^n for an integer value of x, n
 $n \geq 0$

```
int  
{  
    power (int x, int n)
```

```
    if (n == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        return x * power(x, n-1);
```

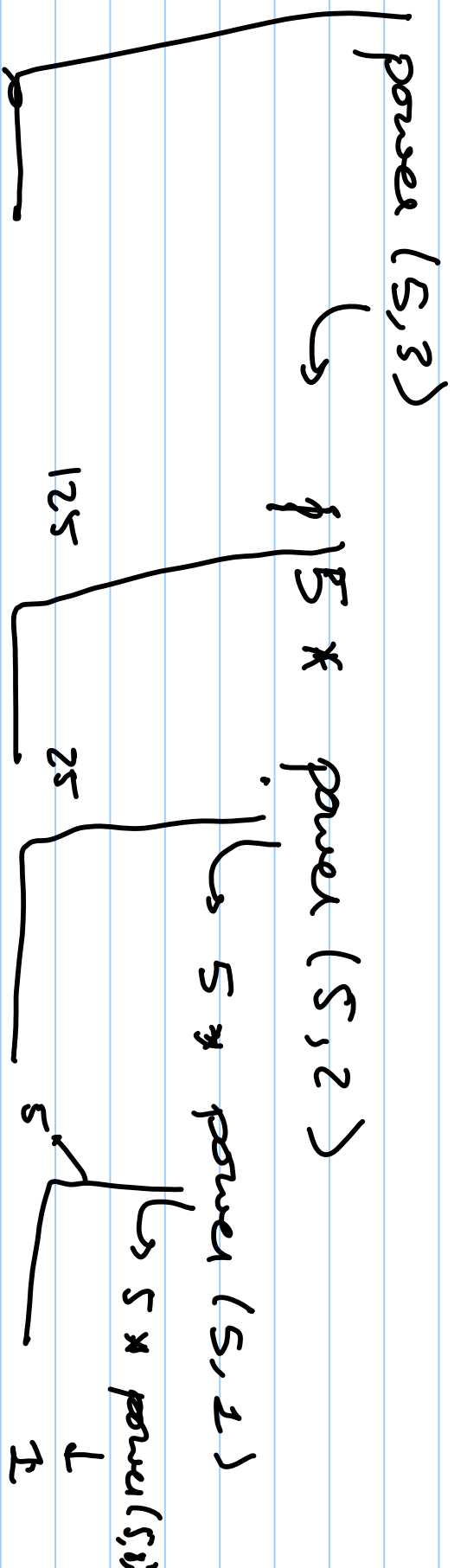
```
    }
```

```
}
```

$$x^n = x * \underbrace{x^{(n-1)}}$$

Base case : $n=0$

can it be $n=1$?



$$x^{18} = 2 x^{17} * x$$

$$x^{19} = (x^2)^9$$

$$x^9 = x * x^8$$

$$x^8 = (x^2)^4$$

$$x^4 = (x^2)^2$$

$$x^2 = x * x$$

$$\text{c.f. } n = 2^k$$

$$x^{2^k} = (x^2)^{2^{k-1}}$$

$$x^{2^{k-1}} = (x^2)^{2^{k-2}}$$

$$\vdots$$

$$x^2 = (x^2)^{2^0}$$

$$x^{2^0} = x$$

\downarrow
 k times
 $\log_2 n$ times

```
int power (int x, int n)
{
    if (n == 0) {
        return 1;
    } else if (n % 2 == 0) {
        return power(x * x, n/2);
    } else {
        return x * power(x, n-1);
    }
}
```

```
    }
    return x * power(x, n-1);
}
```

```
    return x * power(x, (n-1)/2);
}
```

$$2 \quad x^5 = (x^2)^{5/2} \quad X$$

$$x^5 = x \cdot (x^2)^{4/2}$$

Q Binary representation of numbers

Decimal representation

5 3 2
↓ ↓ ↓
100 10 1

$$5 + 3 + 10 + 5$$

$$5 \times 10^2 + 3 \times 10^1 + 9 \times 10^0$$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{matrix}$

$$= \underbrace{1 + 2 + 2 + 2}_2 + \underbrace{1 + 2 + 2}_2 + \underbrace{1 + 2 + 2}_2 =$$

$$= 128 + 32 + 2 + 1 = \underline{\underline{167}}$$

$$\begin{array}{r} 32 \\ + 0 \\ \hline 32 \end{array} \quad \begin{array}{r} 23 \\ + 0 \\ \hline 23 \end{array} \quad \begin{array}{r} 25 \\ + 0 \\ \hline 25 \end{array} \quad \begin{array}{r} 27 \\ + 0 \\ \hline 27 \end{array}$$

// eg. convert from Binary ("101")

int convertFromBinary (string binary)

}