Note Title 01-11-2018

COL100

Nov 1, 2018

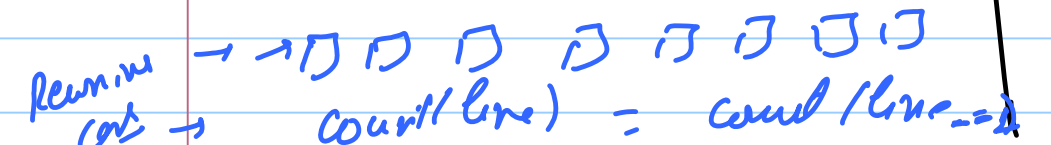
Recursion

→ Solving problems which can be broken into smaller / simpler subproblems

→ Base Case(s)

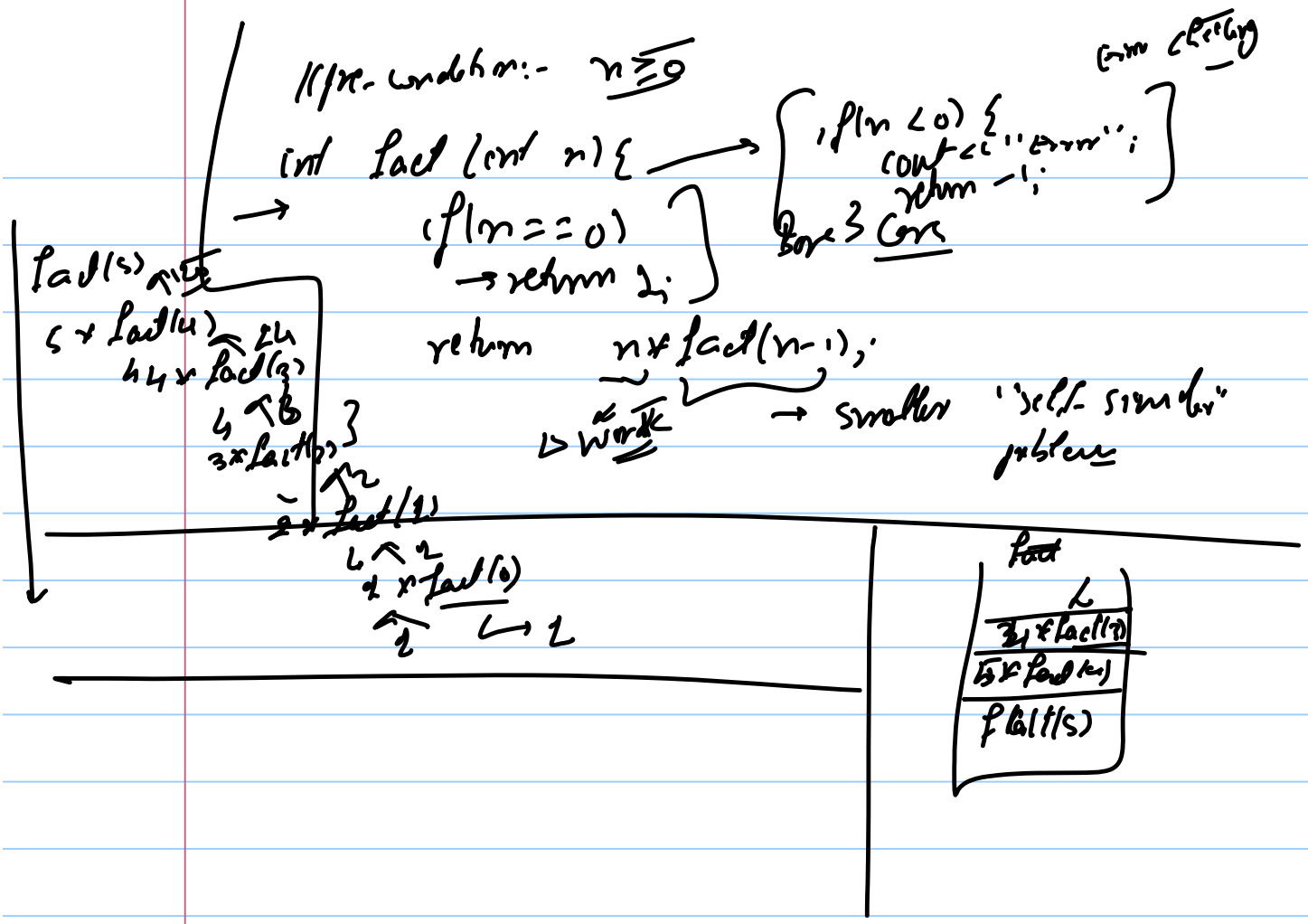
Info:-

All graded assignments count towards final score



Recursive call → count(line) = count(line-1)

Base Case: - count(line-1) = 1 ✓ / no one behind you.


//pre-condition:- n ≥ 0
int fact (int n) {
if(n==0)
return 1;
return n * fact(n-1);
Error Checking
if(n < 0) {
cout << "Error";
return -1;
}
Base Case
fact(5)
5 * fact(4)
4 * fact(3)
3 * fact(2)
2 * fact(1)
1 * fact(0)
1
Work
→ smaller "self-similar" problem
fact
1
2 * fact(1)
3 * fact(2)
4 * fact(3)
5 * fact(4)
fact(5)

$n \geq 0$

```
int mystery(int n) {
    if (n < 10) {
        return n;
    } else {
        int a = n/10;
        int b = n%10;
        return mystery(a+b);
    }
}
```

base case

mystery(648)

$a+b < n$

23

1234

=

mystery(648) = mystery(64+8)

72

9

↳ mystery(7+2)

↳ 9

```
int mystery(int n) {
    if(n < 10)
        return n;
    else {
        int a = n/10;
        int b = n%10;
        return b + mystery(a);
    }
}
```

Base Case

mystery(648) = 8 + mystery(64)

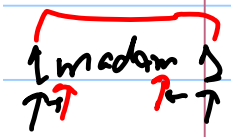
= 8 + 4 + mystery(6)

= 8 + 4 + 6 = 18

Palindromes:- "aka" "madam"

"step on no pets"

"able was I ere I saw elba"




```
bool checkPalindrome(string str){
    int len = str.size();
    for(int i=0; i<n/2; i++){
        char ch1 = str.at(i);
        char ch2 = str.at(n-1-i);
        if(ch1 != ch2)
            return false;
    }
    return true;
}
```

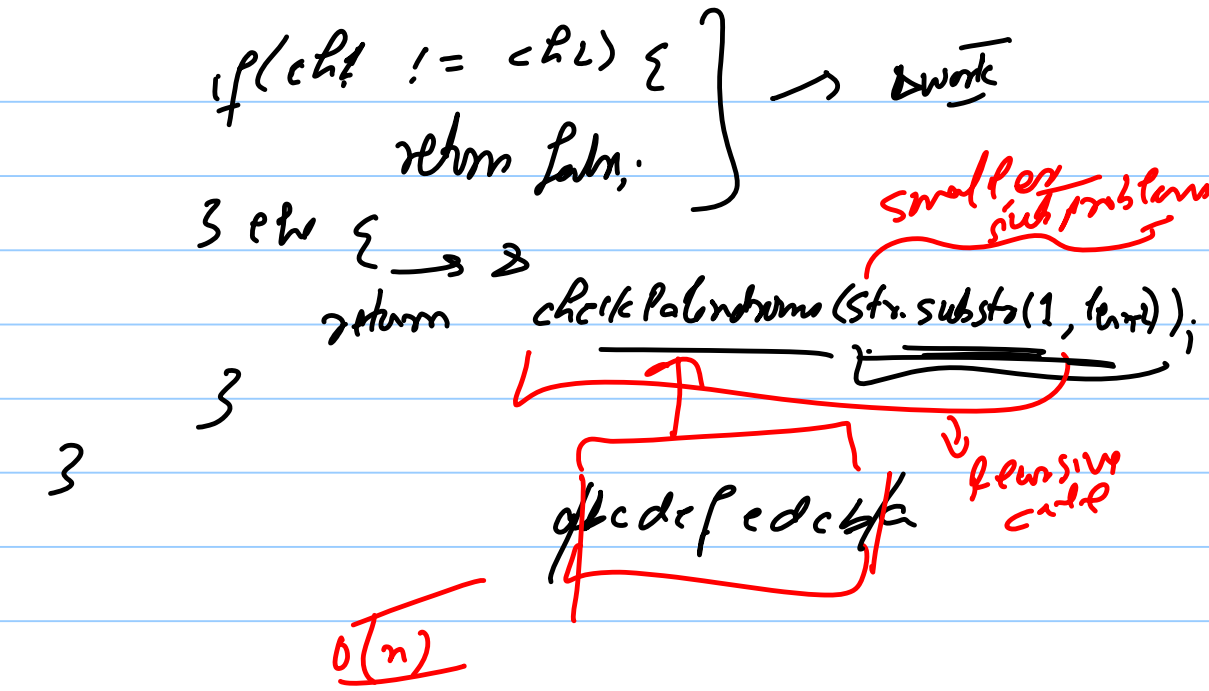
Palindrome: recursive program

1 madam 1

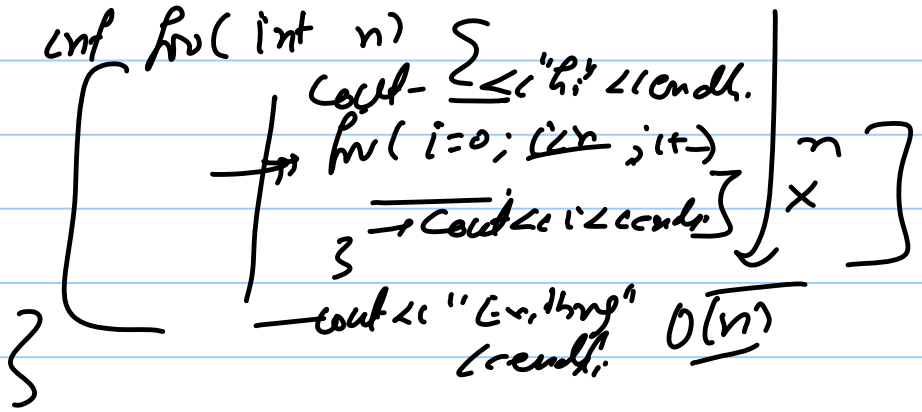
Base Cases:-

$$\begin{bmatrix} str.length() == 0 \\ str.length() == 1 \end{bmatrix} \quad \text{true}$$

```
public bool checkPalindrome(string str) {
    int len = str.length();
    if (len == 0 || len == 1) { // Base case
        return true;
    } else {
        char ch1 = str.at(0);
        char ch2 = str.at(len-1);
    }
}
```


if(ch1 != ch2) {
return false;
}
else {
return
checkPalindrome(Str.substr(1, len-1));
}
}
smaller subproblem
recursive call
O(n)

Be Big-O (Time Complexity) for a piece of code.



findBigO(codeSnippet) {
if (codeSnippet is single line) → O(1); //for loop
} #times

findBigO(codeSnippet){

if(codeSnippet is a simple line) //base case

return O(1);

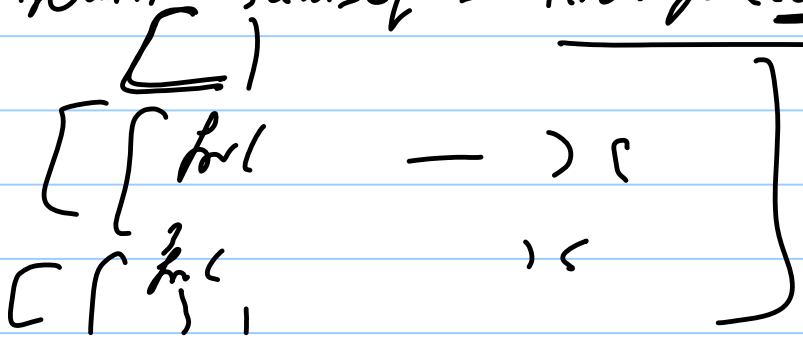
if(codeSnippet is a for-loop) //for loop

return #times-loop-executed * findBigO(code inside_loop);

for codeBlock in codeSnippet

return sum_of - findBigO(codeBlock);

}



```
for(int i=0; i<NxN; i+=3){
    for(int j=3; j<=219; j+1){
        cout<<"sum:"<<i+j
        <<endl;
    }
}
cout<<"finish"<<endl;
```

$O(n^3)$