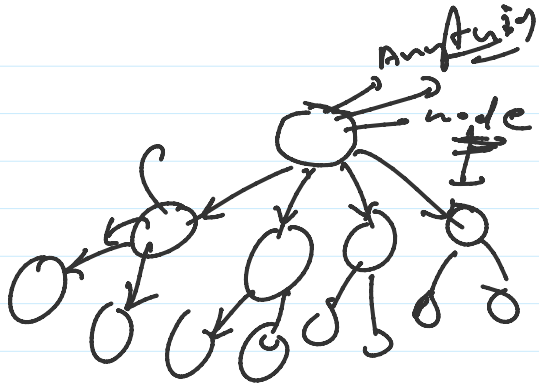
10/09/2022

10 September 2022 18:56

Tree :



→ object

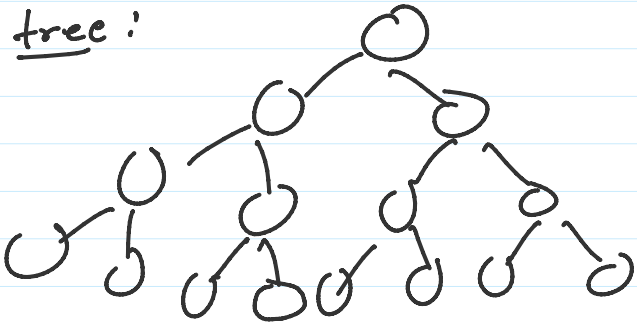
opps ⇒

{ let name = "Dinesh"
  let roll_no = "14" }

diff st

:

Binary tree :



{ 1) How to make tree?
  2) How to traverse the tree
  3) How the make operation on tree node }

```
Const n = 6 % 2

n = 0

Const y = n ? 'one' : 'two' :

y = 'two'
```

⇒ { input data : n = 2 → Pair of parentheses
  output data : All valid combination of Parent

Eg- n = 3

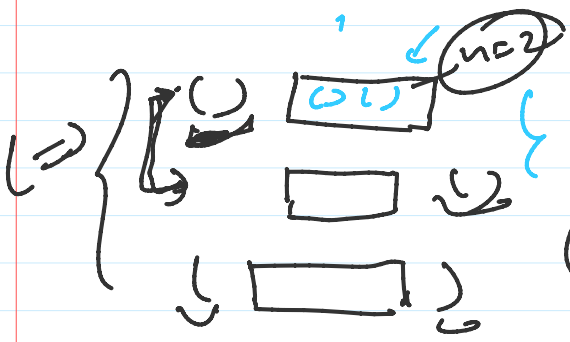
valid

( ) ( ) ( ) (not valid

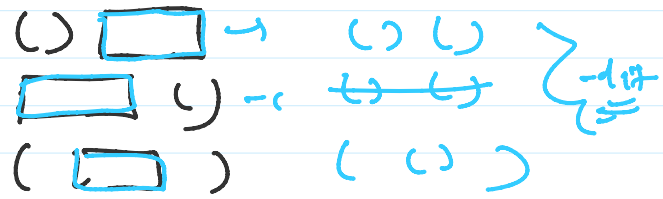
( ) (( )) valid

n = 3

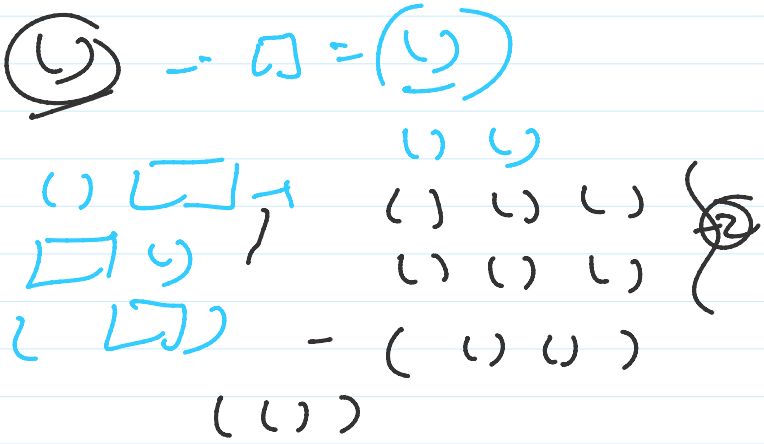
all possible comb^{n} of valid parenthesis for n = 2



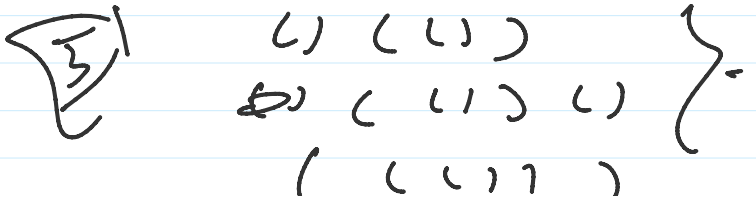
n = 2



n = 1



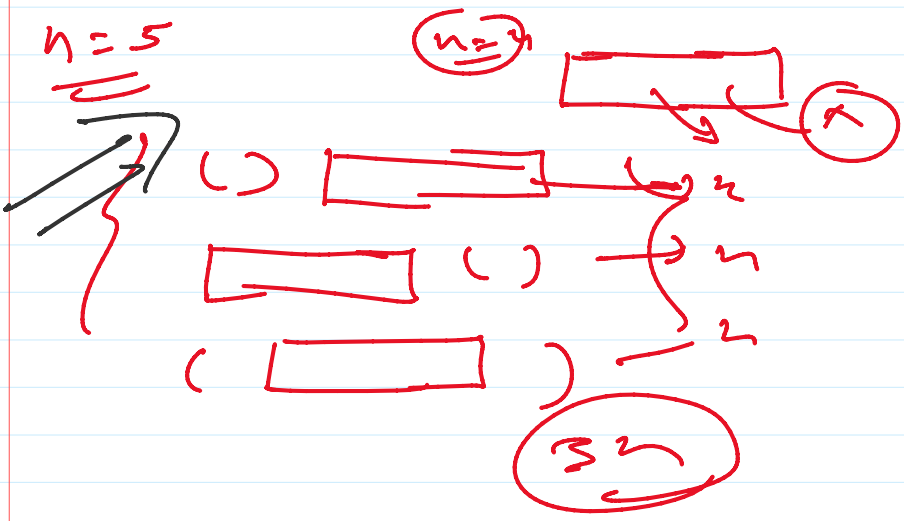
3



(( )) ( )

((( )))

Is Approach clear?



for ["()()", "(())"]

```
let found = false        n = arr.length
for i = 0 -- n-1
    if (arr[i] == s)
        found = true;

if ( found == false)
    arr.push (s)
```

```
let arr = [ ];

function solve (n):

    if n = 1
        return ["()"]

    let temp = solve (n-1)
```

let tmp = solve(n-1)

tmp.()
()tmp
(tmp)

3x

{
let res;

function solve(str, n)
if n == 0 return "";
solve(str + "()", n-2);
solve("()" + str, n-2);
solve("(" + str + ")", n-2);

let arr = [ ]

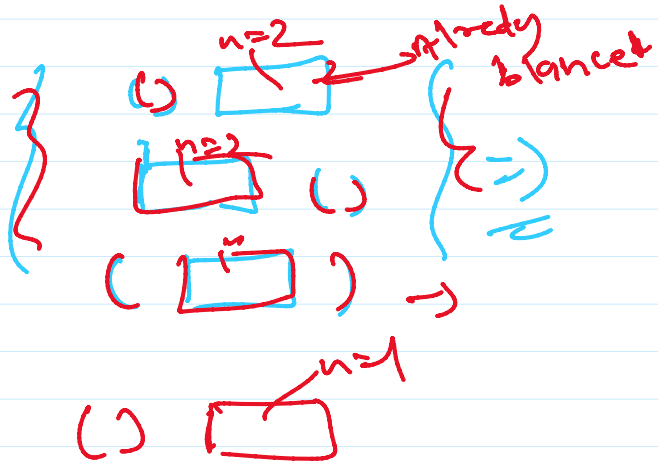
function solve(str, n)

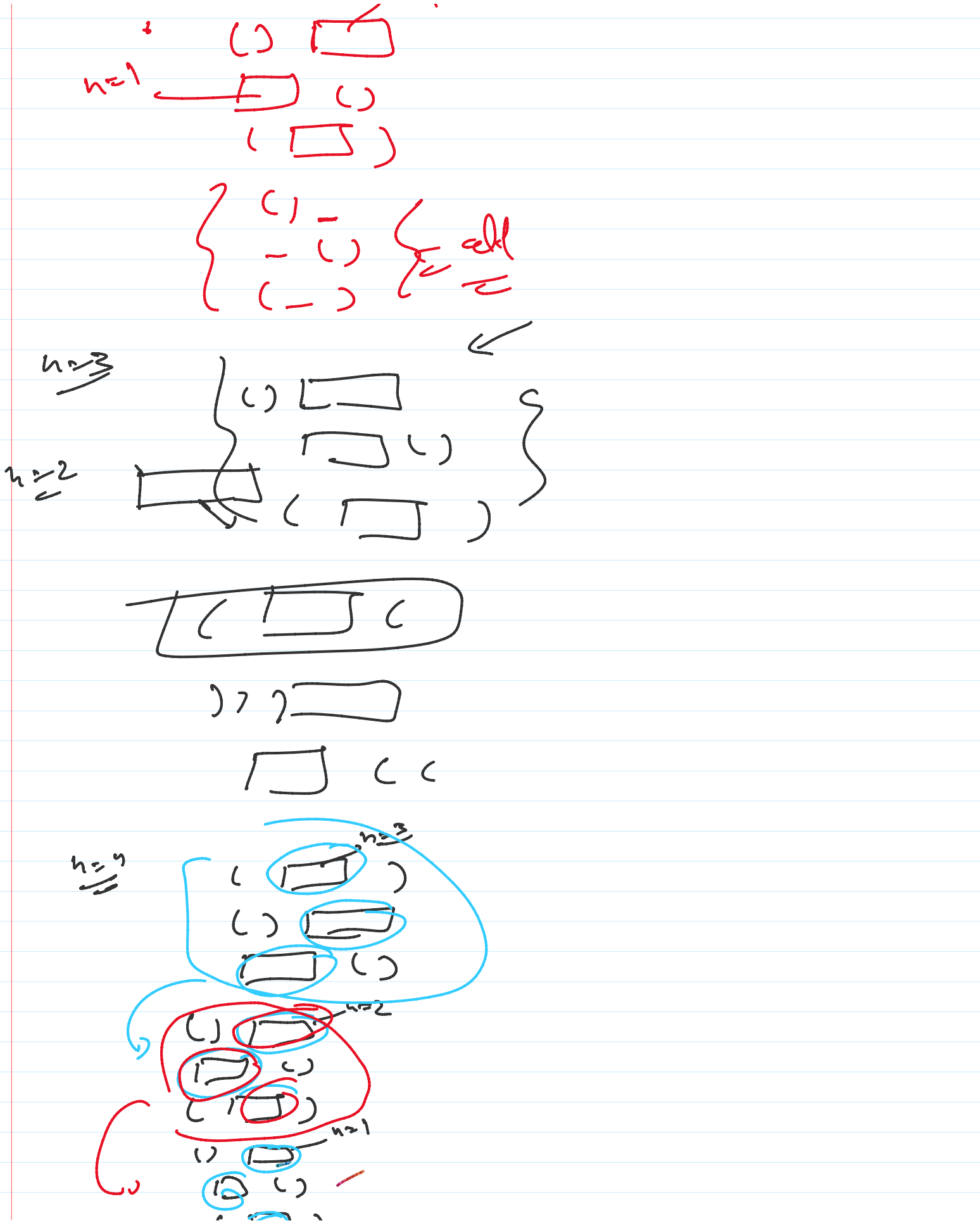
if n == 0:
arr.push(str);

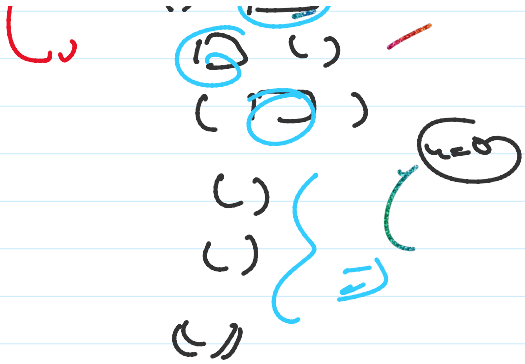
solve("()" + str, n-1)
solve(str + "()", n-1)
solve("(" + str + ")", n-1)

Solve("", 3)

n=3




n=1
n=3
n=2
n=4



Solution:

```
let ans = [];
function solve(str, n){
    if(n == 0){
        if(ans.indexOf(str) == -1 ) ans.push(str);
        return;
    }
    solve( "()" + str, n-1);
    solve( str + "()", n-1);
    solve( "(" + str + ")", n-1);
}
solve("", 3);
console.log(ans);
```