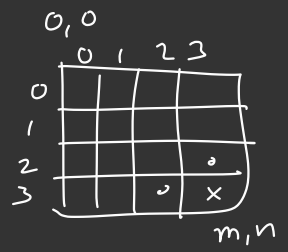
Doubt




```
int ways(int m, int n) {
    if(m==0 && n==0)
        return 1
    return ways(m-1,n) + ways(m,n-1);
}
```

⇒ ways(3,3)

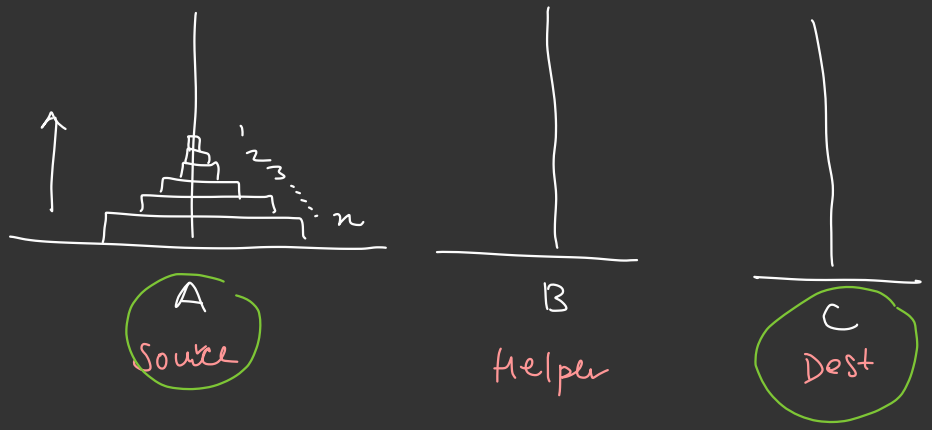
⑫ Tower of Hanoi – [Game]

⇒ 3 Towers (A, B, C)

⇒ N Disks on Tower A

Goal ⇒ Transfer all disks from A to C

- Pick 1 disk at a time
- Never place a big disk over small one.

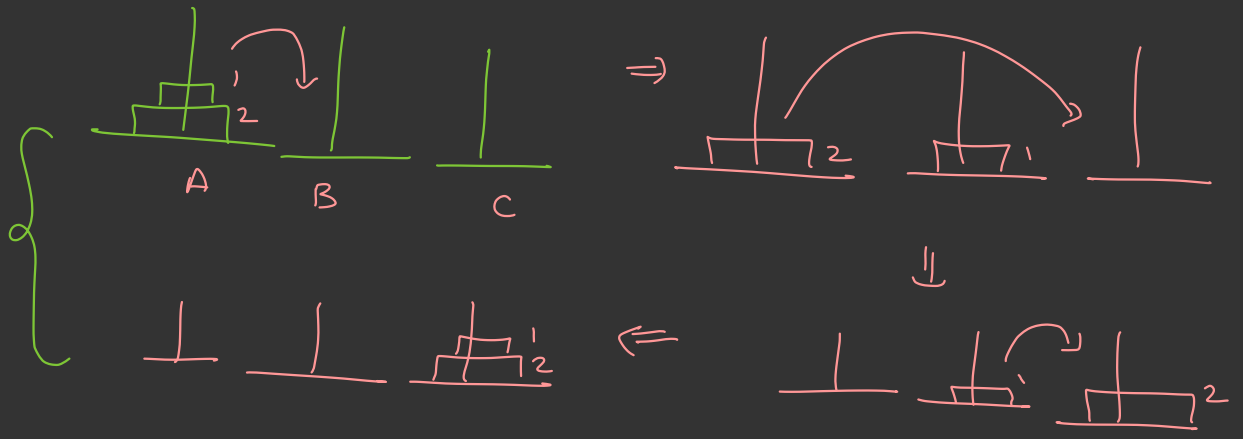


Input – N

output → what steps to perform

Example - N = 1

N = 2



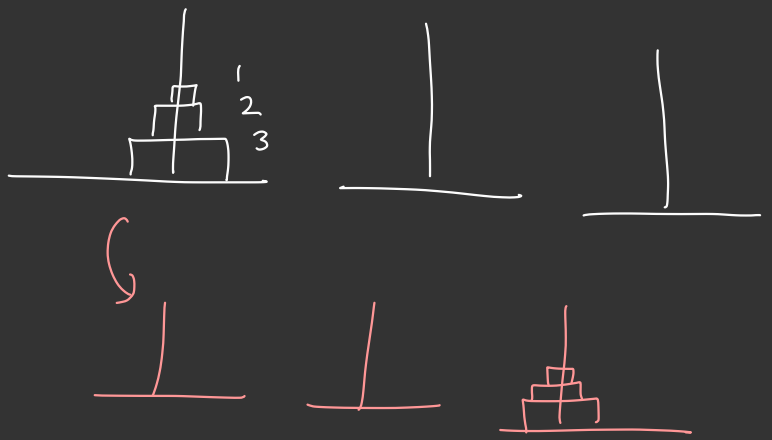
3 steps

$2^2 - 1$

$= 3$

N=3

7 Moves



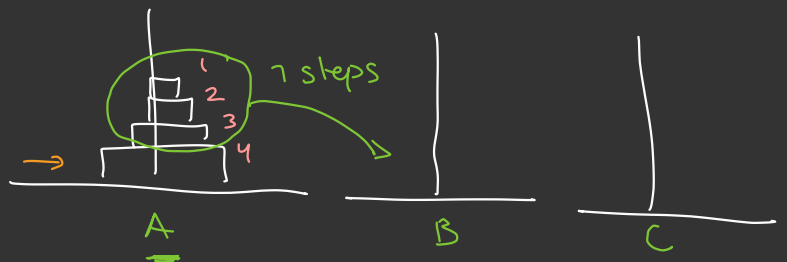
$2^3 - 1$

$= 7$

$2^n - 1$

N=4

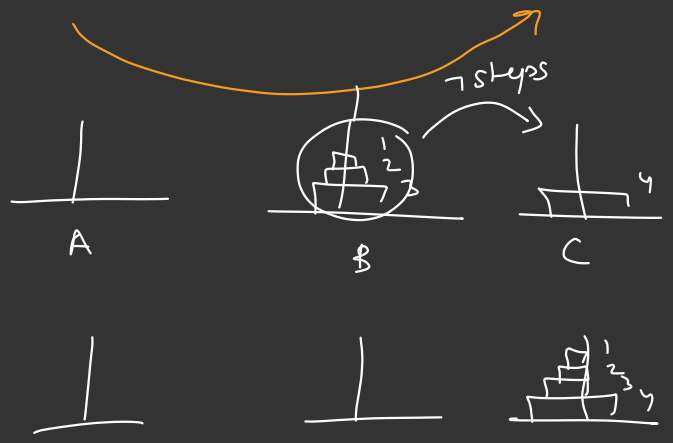
15 Moves



7 (1) Move(N-1 disks, A to B)

1 (2) Shift disk n from A to C

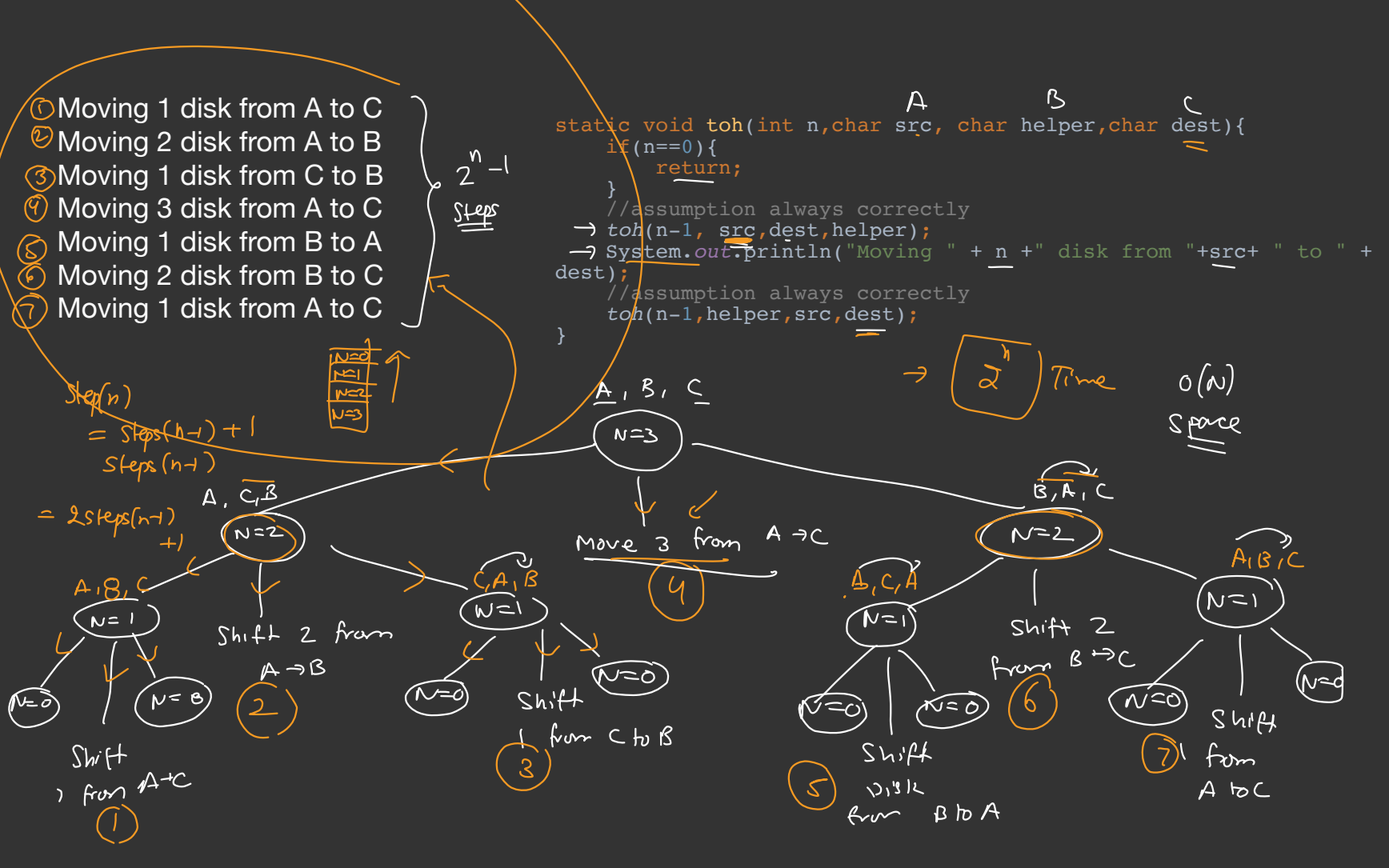
C



7 ③ Move(N-1 disks, B to C)

7+1+7

= 15 Moves


1 Moving 1 disk from A to C
2 Moving 2 disk from A to B
3 Moving 1 disk from C to B
4 Moving 3 disk from A to C
5 Moving 1 disk from B to A
6 Moving 2 disk from B to C
7 Moving 1 disk from A to C
$2^n - 1$ steps
A B C
static void toh(int n,char src, char helper,char dest){
if(n==0){
return;
}
//assumption always correctly
toh(n-1, src,dest,helper);
System.out.println("Moving " + n +" disk from "+src+ " to " + dest);
//assumption always correctly
toh(n-1,helper,src,dest);
}
Steps(n) = Steps(n-1) + 1 + Steps(n-1)
= 2Steps(n-1) + 1
N=0
N=1
N=2
N=3
2^n Time
O(N) Space
A, B, C
N=3
A, C, B
N=2
A, B, C
N=1
N=0
N=0
N=0
Shift 2 from A → B
2
Shift 1 from A → C
1
C, A, B
N=1
N=0
N=0
Shift 1 from C to B
3
Move 3 from A → C
4
B, A, C
N=2
B, C, A
N=1
N=0
N=0
Shift 1 disk from B to A
5
Shift 2 from B → C
6
A, B, C
N=1
N=0
N=0
Shift 1 from A to C
7

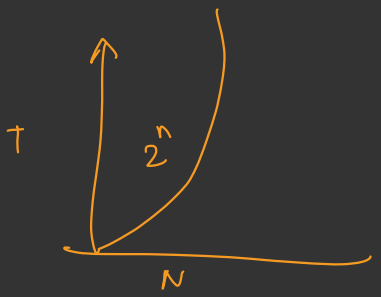
Steps

```
static long steps(int n){
    if(n==0){
        return 0;
    }
    return steps(n-1) + 1 + steps(n-1);
}

// O(n)
static long stepsFaster(int n){
    if(n==0){
        return 0;
    }
    return 2*stepsFaster(n-1) + 1;
}

static long stepsFastest(int n){
    return 1<<n - 1; // O(1)
}
```

2^n

$O(N)$

$O(1)$



$N = 10$

$2^N = 1000$

$N = 20$ $\quad 2^N = (2^{10})^2 = 10^6 \Rightarrow 0.01\,s$

$N = 30$ $\quad = (2^{10})^3 = 10^9 \Rightarrow 10\,s$

$N = 50$ $\quad = (2^{10})^5 = (10^3)^5 = 10^{15}$

115 days

Compress String → Shortest string after dissolving consecutive similar letters.

"aa bb ccc b xad"

aa bb bbx ad

aa xad

xad

aa bbb ccc b xxad

aa ccc b xxad

ccc b xxad

b xxad

bad

a bbbb cc bxxad

a cc b xx ad

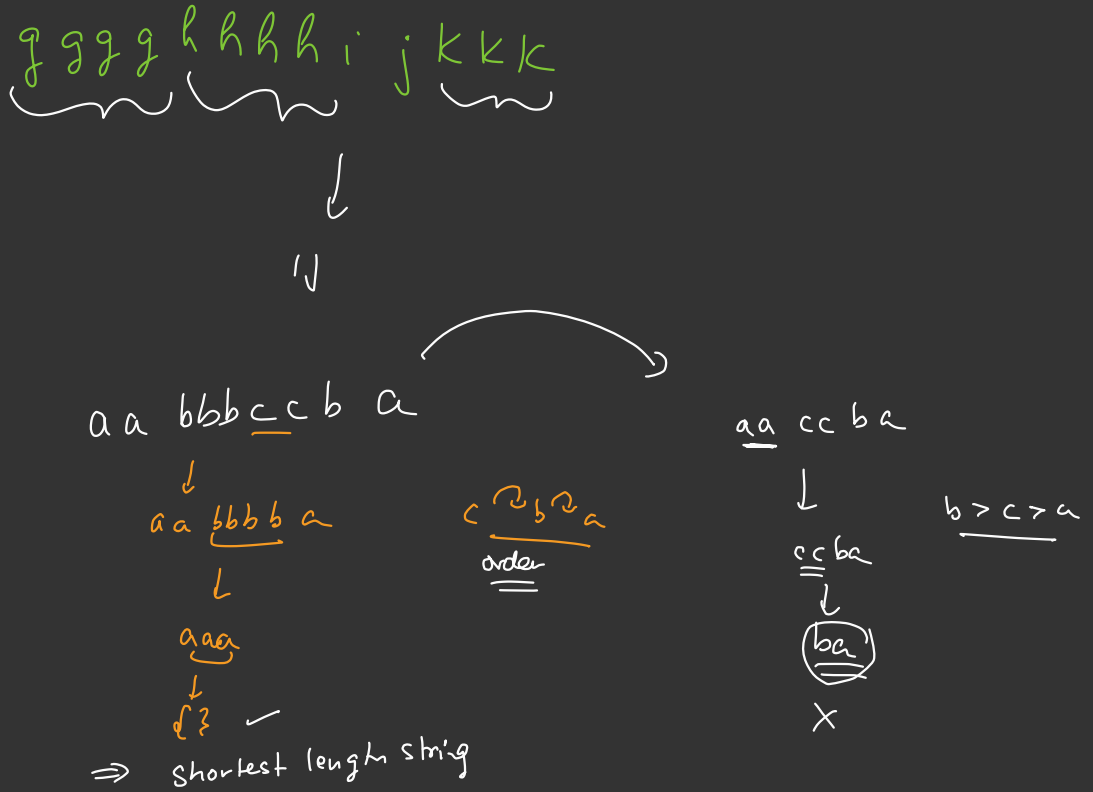
ab ad

aa bb b xxad

aa xx ad

aaa d

d


g g g g h h h h i j k k k
aa bbb c c b a
aa bbb a
aaa
{}
⇒ Shortest length string
c b a
order
aa cc b a
cc ba
ba
X
b > c > a

Brute force

↳ Try all possible ways of dissolving

a a b b b c c K K c c b a a

k > c > b > a

b > - - -

a > - - -

c > - - -

(a)

Input

Kaaa bbccc baxd

Kaaa bbbaxd

=>

k aaa a x d

output → K x d

Smallest string

output

String dissolve(S)

Base Case

if no consecutive letters are same return (S).

best output = S;

=> for (i=0 ____ i<=n-1) {

s' = String with segment from "i dissolved".

→ output = dissolve(s')

if (output len < best output) {

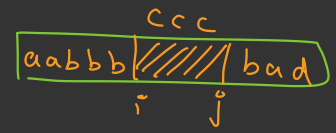
update best output

}

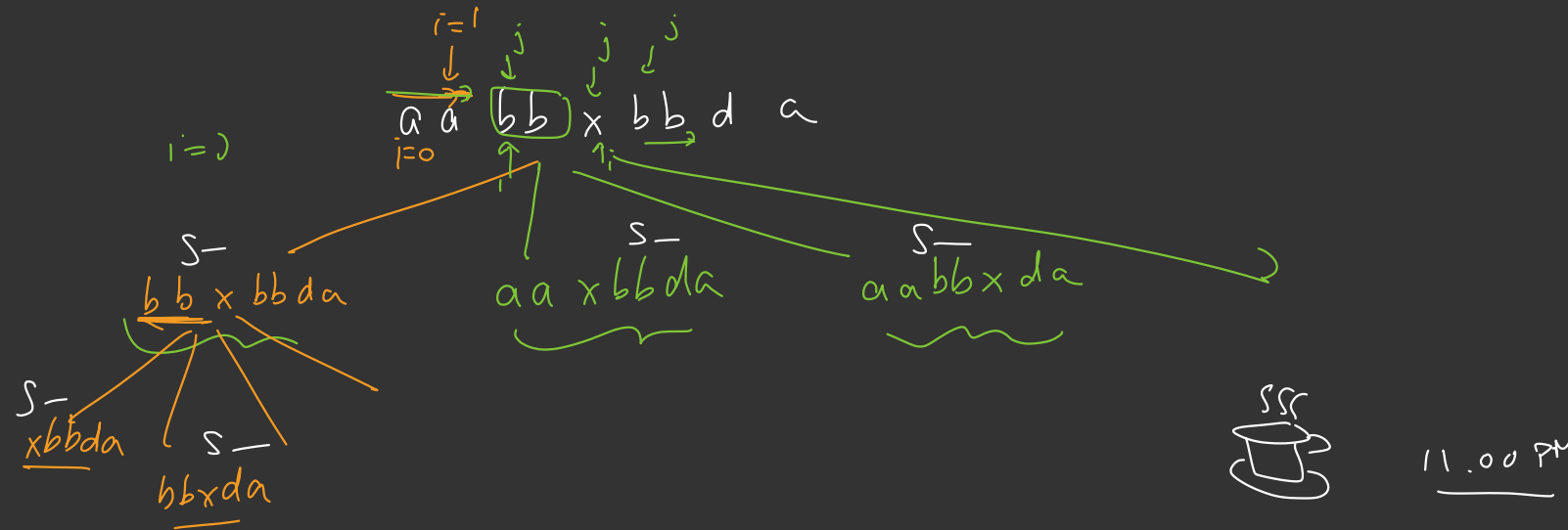
}

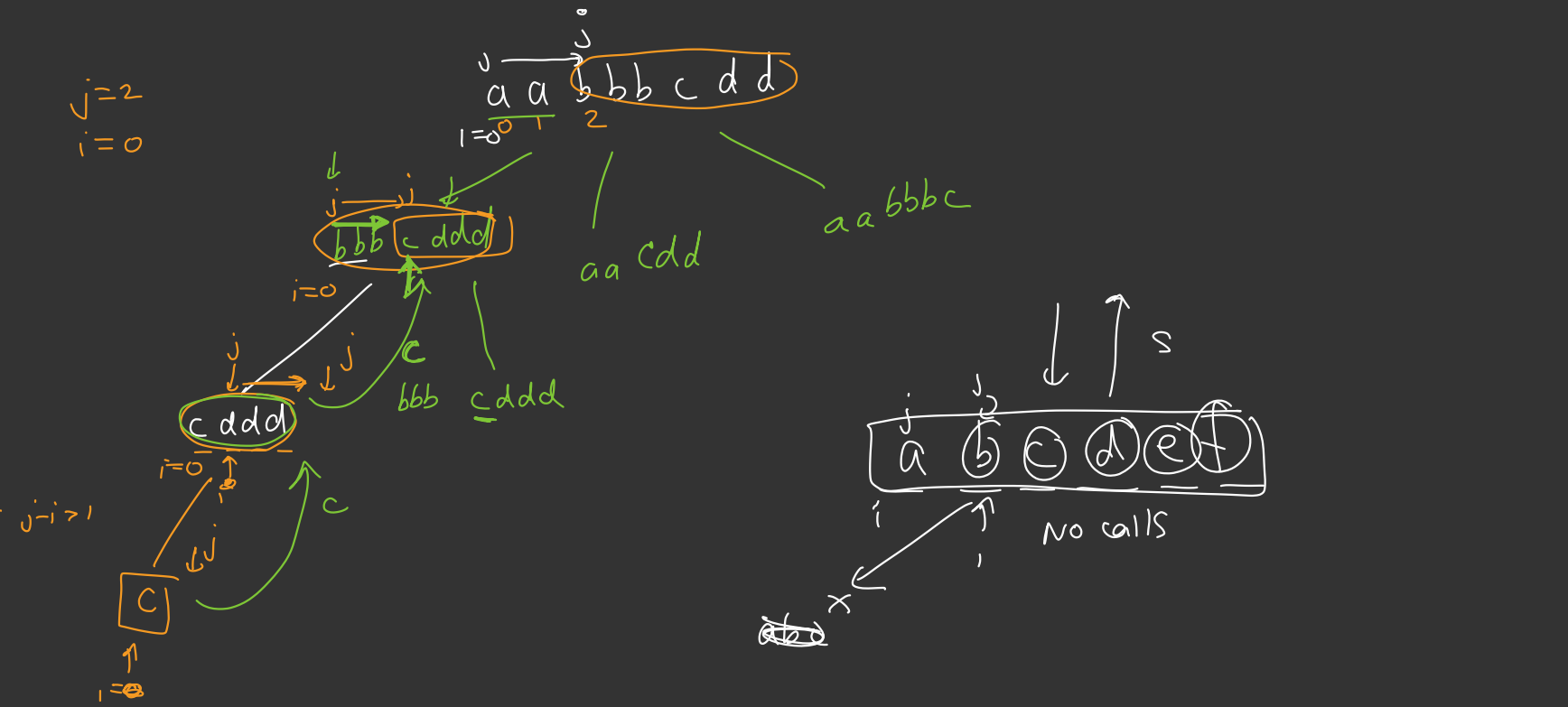
return best output

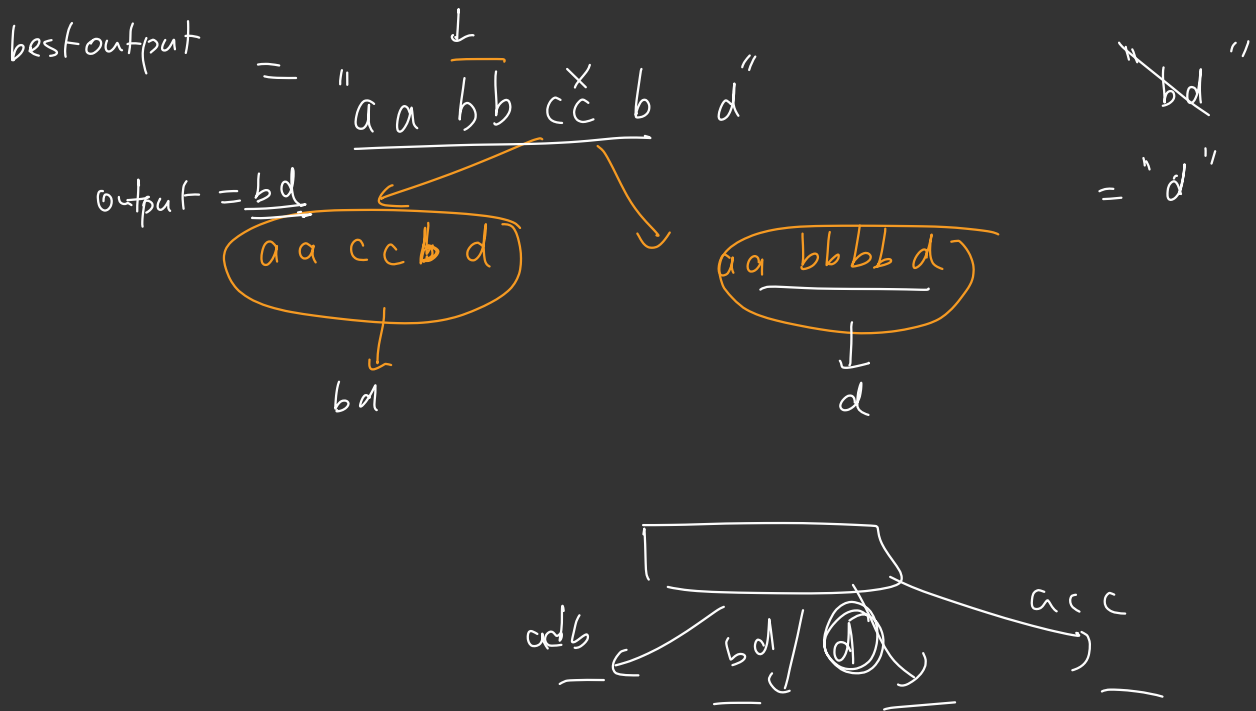
}



```java
static String dissolve(String s){
    //rec case
    String bestOutput = s;
    for(int i=0; i< s.length(); ){
        int j=i;
        while(j<s.length() && s.charAt(j)==s.charAt(i)){
            j++;
        }
        //found a segment that can can compressed
        if(j-i>1) {
            String s_ = s.substring(0, i) + s.substring(j);
            String output = dissolve(s_);
            if (output.length() < bestOutput.length()) {
                bestOutput = output;
            }
        }
        i = j;
    }
    return bestOutput;
}

public static void main(String[] args) {
    String input =  "aabbbccdddeeefff"; //"aabbccckkdaab"; //"kaaabbccbaxd";
    System.out.println(dissolve(input));
}
```


i=1
j
j
j
a a b b x b b d a
i=0
i=0
S—
b b x b b d a
S—
a a x b b d a
S—
a a b b x d a
S—
x b b d a
S—
b b x d a
11.00 PM


j=2
i=0
a a b b c d d
i=0 1 2
aa cdd
aa bbc
bbb
c ddd
i=0
bbb
c ddd
c ddd
i=0
j-i>1
C
i=0
s
a b c d e f
No calls



Print permutations of a given string

Input abc

n!

abc bac cab
acb bca cba

6 output

Subset

{1,2,3}

{}
{1,2}
{1,3}
:

$2^3 = 8$