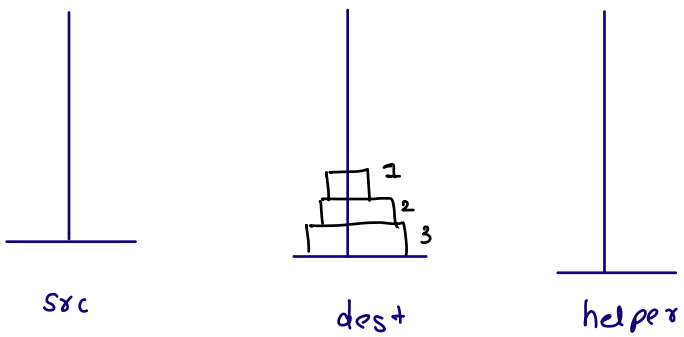
Q.1 Tower of Hanoe

Given n disks and 3 towers (src, dest and helper). Transfer all disks from src to destination and print instructions by keeping the following rules in mind:

1) we can move only one disk at a time.
2) big disk can't be placed over a small disk.
3) we can only move top-most disk from a tower.

n = 3



move 1 from s to d ✓

move 2 from s to h ✓

move 1 from d to h ✓

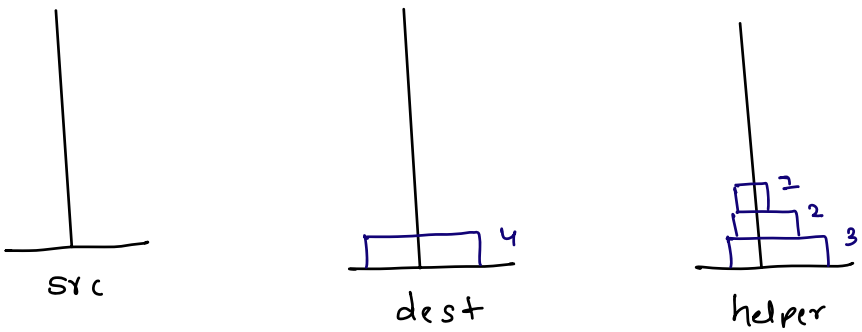
move 3 from s to d ✓

move 1 from h to s ✓

move 2 from h to d ✓

move 1 from s to d ✓

n = 4



Agenda 1 : transfer 1st 3 disks from src to helper

Agenda 2 : transfer 4th disk from src to dest

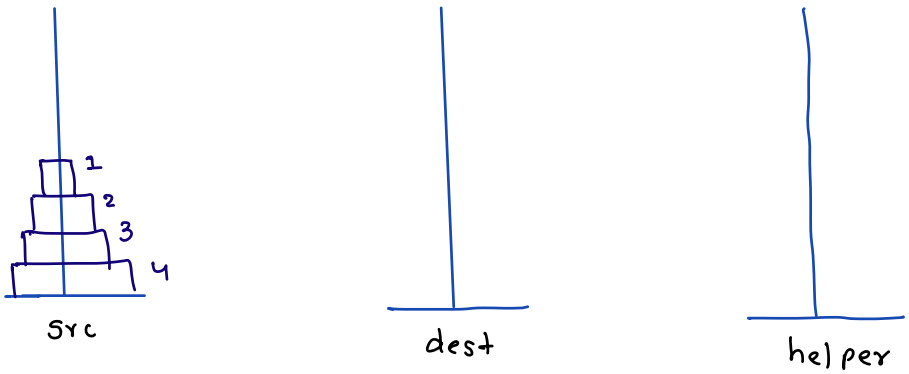
Agenda 3 : transfer 1st 3 disks from helper to dest

- move 1 from s to h
- move 2 from s to d
- move 1 from h to d
- move 3 from s to h
- move 1 from d to s
- move 2 from d to h
- move 1 from s to h

move 4 from s to d

7 more steps (transfer 3 disks from helper to dest)

n=4



toh(n, s, d, h) ⇒ transfer n disks from src to dest using helper.

↳ (i) transfer n-1 disks from src to helper

```
toh (n-1, s, h, d);
```

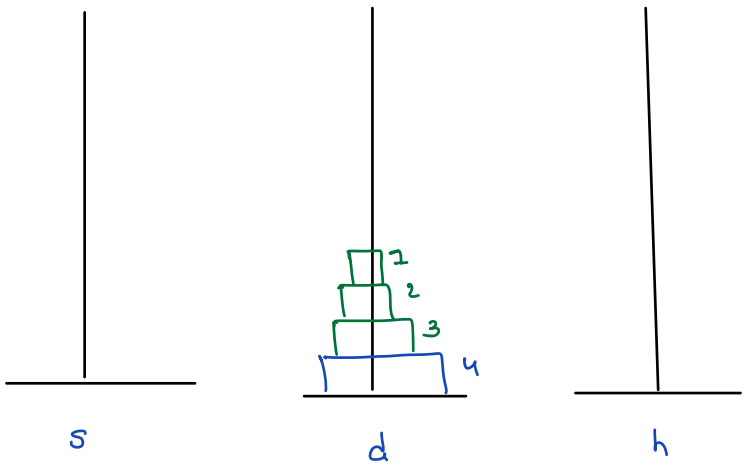
(ii) transfer n^{th} disk from src to dest

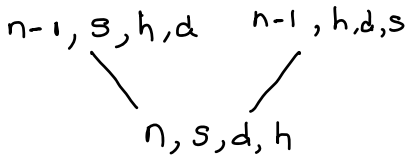
```
sopln ("move" + n + "from" + s + "to" + d);
```

(iii) transfer n-1 disks from helper to dest

```
toh (n-1, h, d, s);
```

```
Move 1 from s to h
Move 2 from s to d
Move 1 from h to d
Move 3 from s to h
Move 1 from d to s
Move 2 from d to h
Move 1 from s to h
Move 4 from s to d
Move 1 from h to d
Move 2 from h to s
Move 1 from d to s
Move 3 from h to d
Move 1 from s to h
Move 2 from s to d
Move 1 from h to d
```





n = 3

```
static void toh(int n, char s, char d, char h) {
    if(n == 0) {
        return;
    }

    //move n-1 disks from src to helper
    toh(n-1,s,h,d);

    //move nth disk from src to dest
    System.out.println("Move " + n + " from " + s + " to " + d);

    //move n-1 disks from helper to dest
    toh(n-1,h,d,s);
}

public static void main(String args[]) {
    toh(4,'s','d','h');
}
```

move 1 from s to d

move 2 from s to h

move 1 from d to h

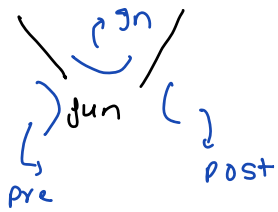
move 3 from s to d

move 1 from h to s

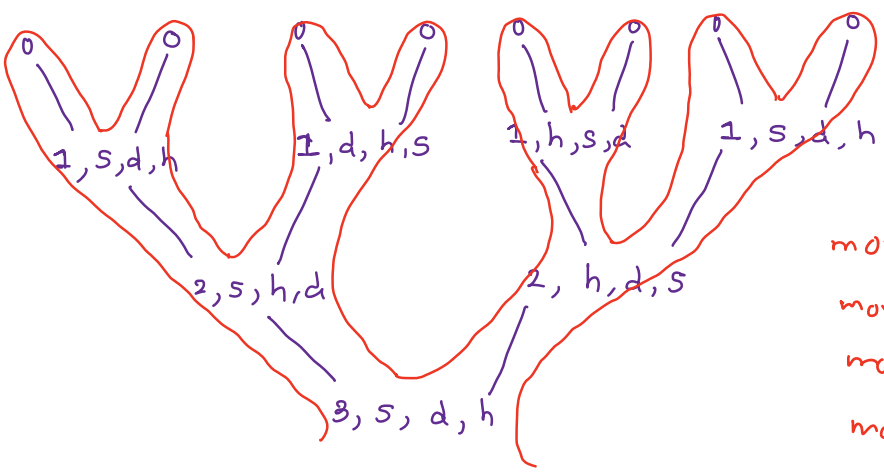
move 2 from h to d

move 1 from s to d

```java
static void toh(int n, char s, char d, char h) {
    if(n == 0) {
        return;
    }

    //move n-1 disks from src to helper
    toh(n-1,s,h,d);

    //move nth disk from src to dest
    System.out.println("Move " + n + " from " + s + " to " + d);

    //move n-1 disks from helper to dest
    toh(n-1,h,d,s);
}

public static void main(String args[]) {
    toh(4,'s','d','h');
}
```



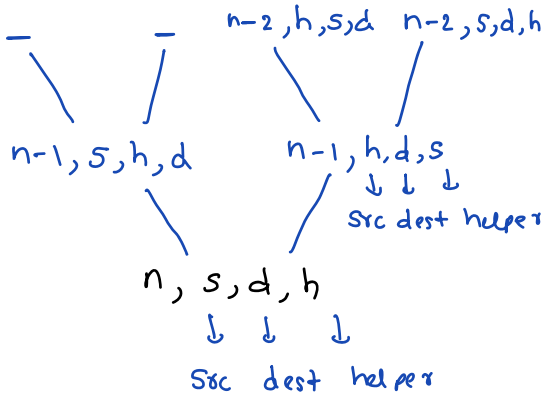
euler diagram
=

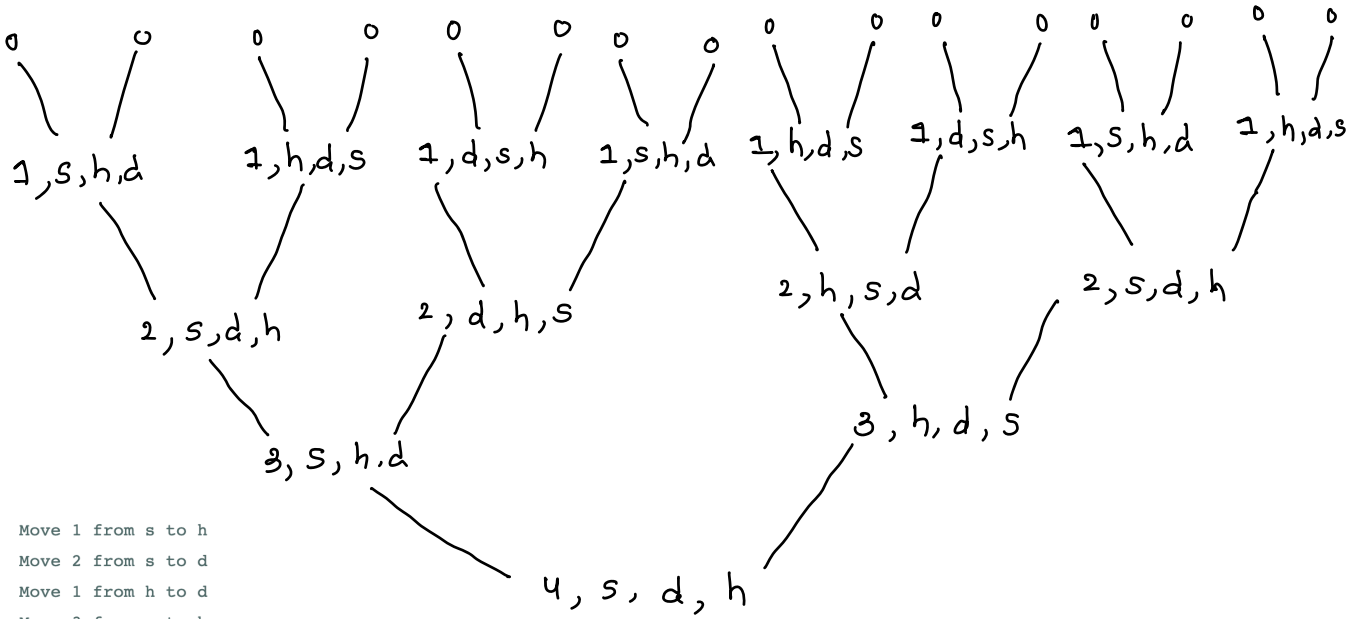


move 1 from s to d
move 2 from s to h
move 1 from d to h
move 3 from s to d
move 1 from h to s
move 2 from h to d
move 1 from s to d

n = 4

```java
static void toh(int n, char s, char d, char h) {
    if(n == 0) {
        return;
    }

    //move n-1 disks from src to helper
    toh(n-1,s,h,d);

    //move nth disk from src to dest
    System.out.println("Move " + n + " from " + s + " to " + d);

    //move n-1 disks from helper to dest
    toh(n-1,h,d,s);
}

public static void main(String args[]) {
    toh(4,'s','d','h');
}
```






```
Move 1 from s to h
Move 2 from s to d
Move 1 from h to d
Move 3 from s to h
Move 1 from d to s
Move 2 from d to h
Move 1 from s to h
Move 4 from s to d
Move 1 from h to d
Move 2 from h to s
Move 1 from d to s
Move 3 from h to d
Move 1 from s to h
Move 2 from s to d
Move 1 from h to d
```

Space complexity

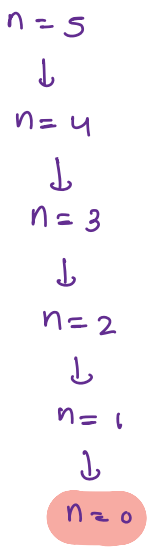
↳ function calls are getting stored in call stack.

TC: (TC of single function) * (total no. of function calls)

SC: (SC of single function) * (max no. of function calls in call stack at any point of time).

examples :-

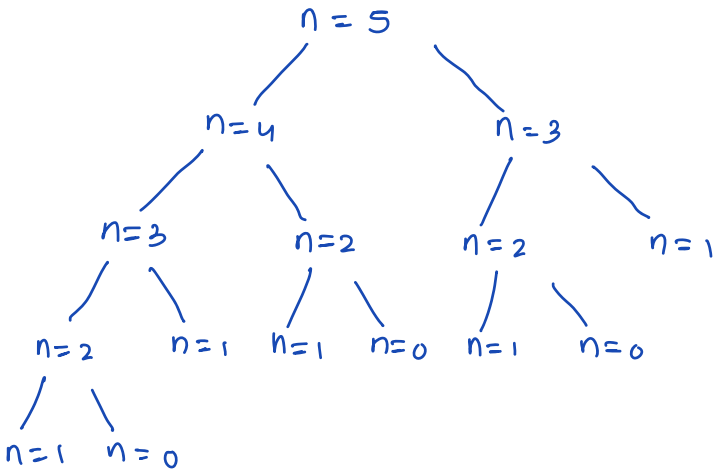
```
int fact (int n) {
    if (n==0) {
        return 1;
    }
    int temp= fact(n-1);
    return temp*n;
}
```



TC: O(n)

SC: O(n)

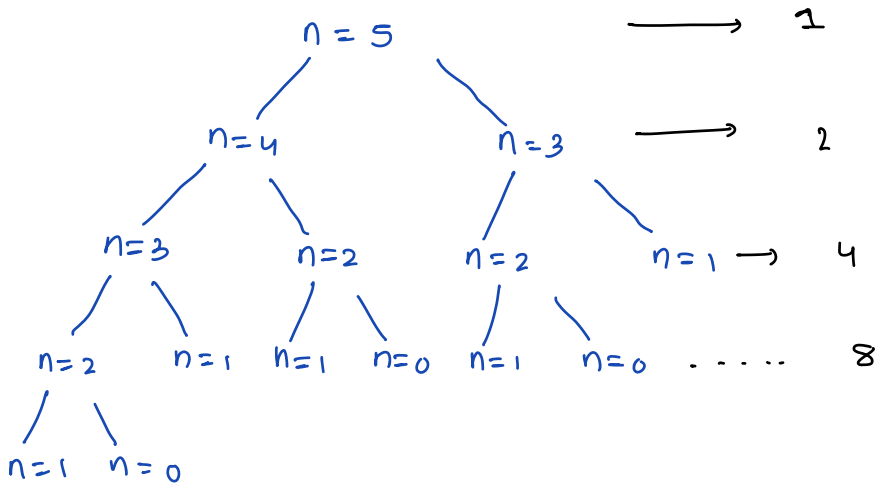
```
int fib (int n) {
    if(n==0 || n==1) {
        return n;
    }
    int temp1= fib(n-1);
    int temp2= fib(n-2);
    return temp1 + temp2;
}
```



SC: O(n)

TC: $O(2^n)$

max no. of function calls in call stack at any point of time) = length of longest branch



total calls = $1 + 2 + 4 + 8 \ldots\ldots$

$$S_t = \frac{a(r^t - 1)}{r - 1}$$

$a = 1$

$r = 2$

$t = n$

$$= \frac{1(2^n - 1)}{1} \approx 2^n$$

```
int pow (int a, int n) {
    if (n==0) {
        return 1;
    }
    int temp = pow(a, n-1);
    return temp*a;
}
```

3,4
↓
3,3
↓
3,2
↓
3,1
↓
3,0

TC : O(n)

SC : O(n)

```
int pow (int a, int n) {
    if(n==0) {
        return 1;
    }
    int temp = pow(a, n/2);
    if(n%2==0) {
        return temp*temp;
    }
    else {
        return temp*temp*a;
    }
}
```

3,20
↓
3,10
↓
3,5
↓
3,2
↓
3,1
↓
3,0

TC: $O(\log_2 n)$

SC: $O(\log_2 n)$

```
int pow (int a, int n) {
    if(n==0) {
        return 1;
    }
    if(n%2==0) {
        return pow(a,n/2) * pow(a,n/2);
    }
    else {
        return pow(a,n/2) * pow(a,n/2) * a;
    }
}
```

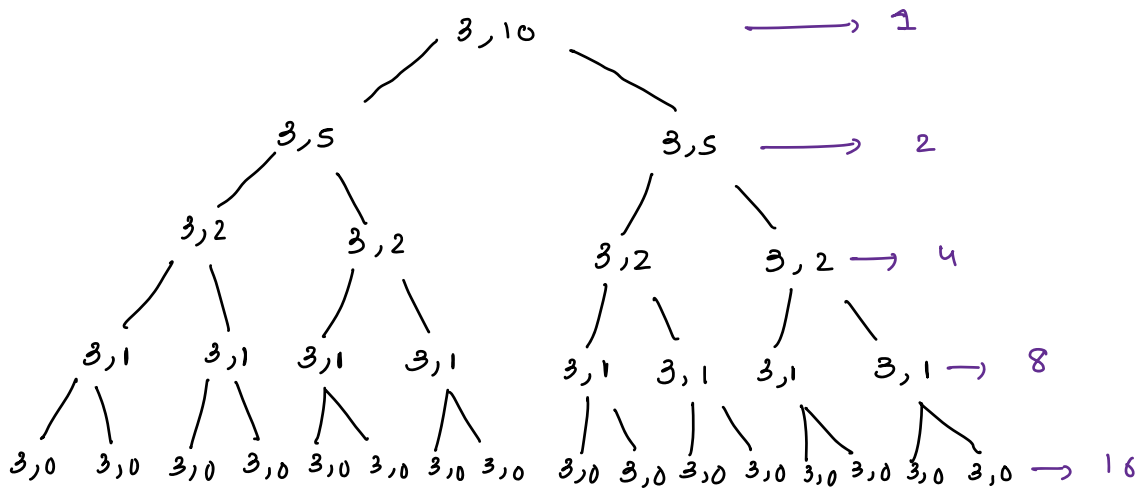
SC :

TC :

Pseudo smart

SC: $O(\log_2 n)$

TC: $O(n)$



total calls = $1+2+4+8+\dots$

$$S_t = \frac{a(r^t-1)}{r-1} = \frac{1(2^{\log_2 n}-1)}{1}$$

$a = 1$

$r = 2$

$t = \log_2 n$

$= 2^{\log_2 n} - 1$

$\{2^{\log_2 n} = n\}$

$\approx n$

Todo : TC and SC for tower of hanoe

Doubts

kth symbol

A = 0

0 → 01

1 → 10

B = 5

A = 4 temp = [0,1,1,0] ans = [0,1,1,0,1,0,0,1]

↓

A = 3 temp = [0,1] ans = [0,1,1,0]

↓

A = 2 temp = [0] ans = [0,1]

↓

A = 1

String reverse print

S = a b c d e

0 1 2 3 4

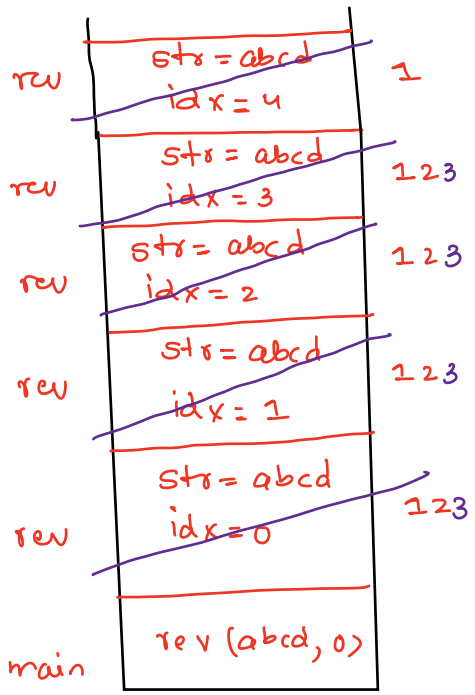
idx

```
main () {
    ==
    reversePrint (str, 0);
}
```

```
void reversePrint (String str, int idx) {
    if (idx == str.length()) {
        return;
    }
    reversePrint (str, idx+1);
    sop (str.charAt(idx));
}
```

```
void reversePrint (String str, int idx) {
    if (idx == str.length()) {
1       return;
    }
2   reversePrint (str, idx+1);
3   SOP (str.charAt(idx));
}

main () {
    ==
    reversePrint (str, 0);
}
```

str = a b c d
0 1 2 3



d c b a

	00	11	22	33	44	55	66	77	88	99
	33	55	-22	-44	11	22	11	44	-66	22
sum:	00	33	88	66	22	33	55	66	1010	44	66