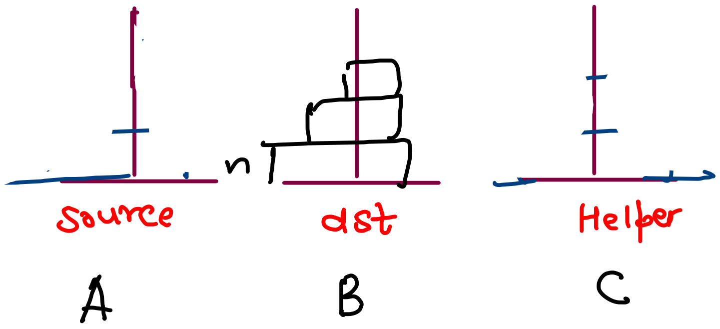
Tower of Hanoi

print steps of moving n disks from source to destination using Helper



function - toh(n, src, dst, hlp)
{

Rule →
1. Move one disk at a time
2. Heavy disk on Light disc is not allowed
3. Main - Steps:

Expectation ← { toh(n, A, B, C)

A → source
B → dst
C → helper

faith → toh(n-1, A, C, B)

A → Source
C → dst
B – helper

move n disc from src to dst

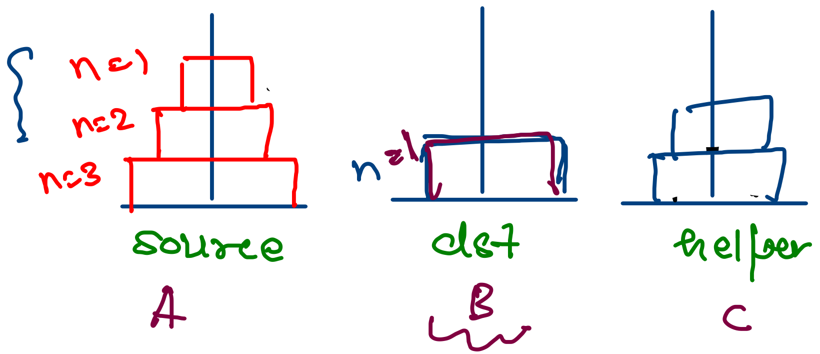
toh(n-1, C, B, A); }

C → Source
B → dst
A → helper

High level Analysis

Expectation →

toh (n, A, B, C)



toh (n, source, dst, helper)

faith →

toh (n-1, A, C, B);

n : A to B

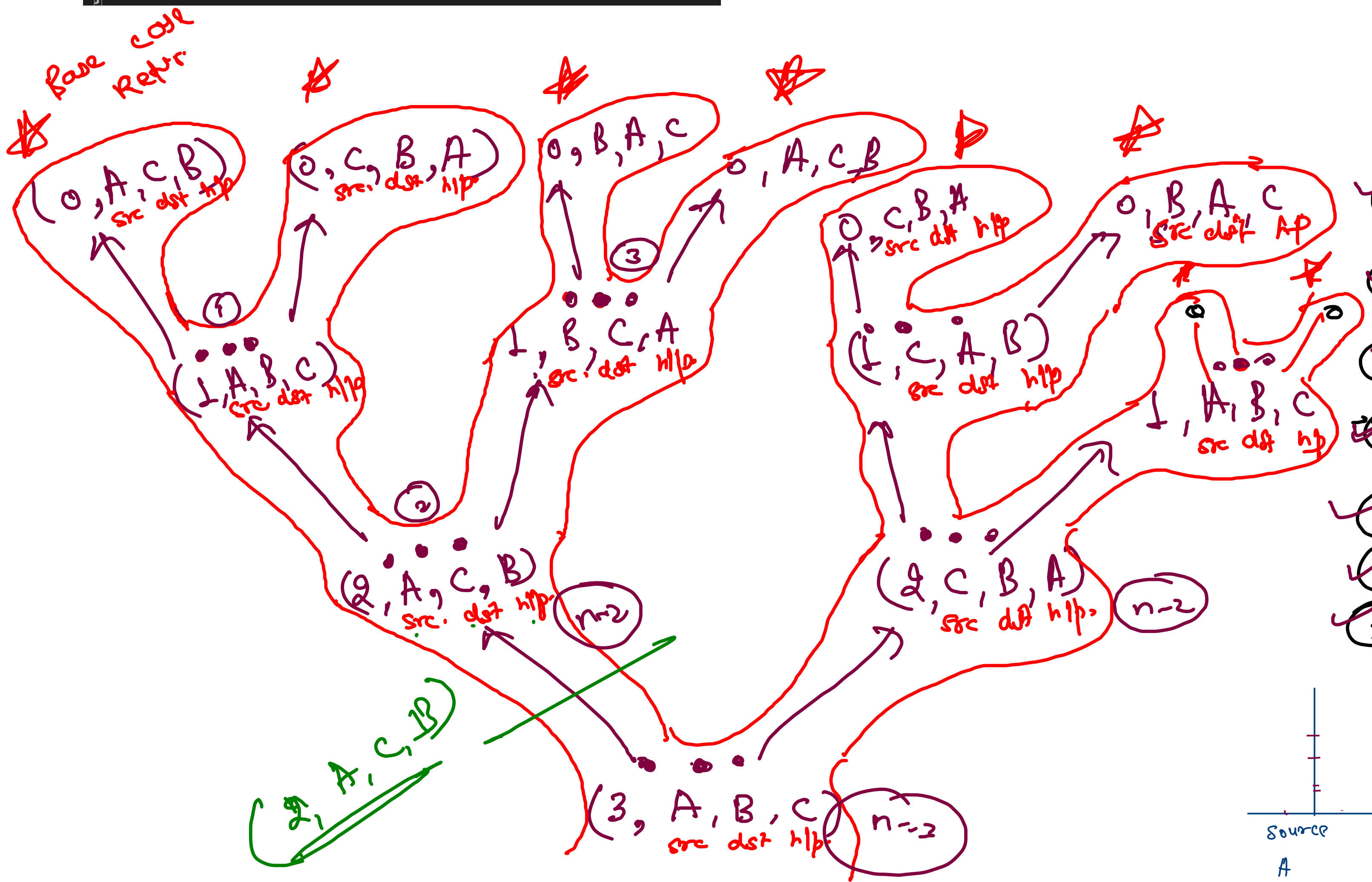
toh (n-1, C, B, A);

toh (n-1, A, B, C)
n - A to C
toh (n-1, B, C, A)
toh (n-1, C, A, B)
n → C to B
toh (n-1, A, B, C)

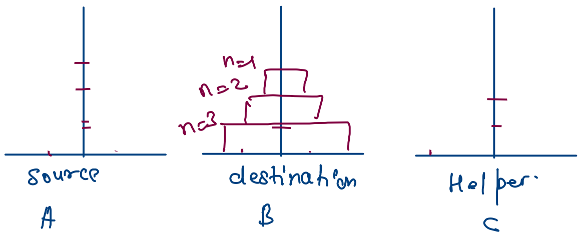
```java
public static void toh(int n, char src, char dst, char hlp) {

    toh(n - 1, src, hlp, dst);
    System.out.println("move " + n + "th disk from " + src + " to " + dst);
    toh(n - 1, hlp, dst, src);
}
```

if(n==0)
return;



1 move 1 from A to B
2 move 2 from A to C
3 move 1 from B to C
4 move 3 from A to B
5 move 1 from C to A
6 move 2 from C to B
7 move 1 from A to B



functional equation $\rightarrow$ $T(n) = 2T(n-1) + 1$

$Step(n) = Step(n-1) + Step(n-1) + 1$

$T(n) = T(n-1) + T(n-1) + 1$

$= 2T(n-1) + 1$

$2^n - 1$

$2T(n-1) = 2^2 T(n-2) + 2$

$2^3 - 1 = 8 - 1 = 7$

$2^2 T(n-2) = 2^3 T(n-3) + 2^2$

$n = 3$

$n = x = 1$

$n = 5, \quad 2^5 - 1 = 31$

$n = x$

$\vdots \qquad \vdots$

$x = n-1$

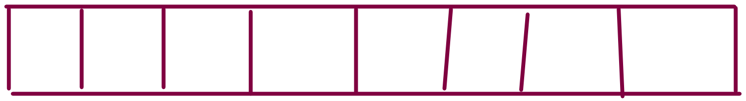
$2^{x-1} T(n-x) = 2^x T(n-x) + 2^{x-1}$

$T(n) = 2^x T(1) + \underbrace{1 + 2 + 2^2 + \cdots}_{} + 2^{n-2}$

$r = 1$

$= 2^{n-1} + \dfrac{1(2^{n-1} - 1)}{2-1}$ $\qquad \dfrac{a(r^n - 1)}{r-1}$

$= 2^{n-1} + 2^{n-1} - 1 = 2 \cdot 2^{n-1} - 1 = 2^n - 1$



Recursion on array Required

→ array, indx

arr → 10, 20, 30, 40, 50, 60
0 1 2 3 4 5

display (High level Analysis)

Expectation, display (arr, 0) → 10 20 30 40 50 60

faith display (arr, 1) → 20 30 40 50 60

Merging → display (arr, 0) → 10 $\underbrace{20 \quad 30 \quad 40 \quad 50 \quad 60}$

display (arr, 0) = print(arr[0]) + display(arr, 1)

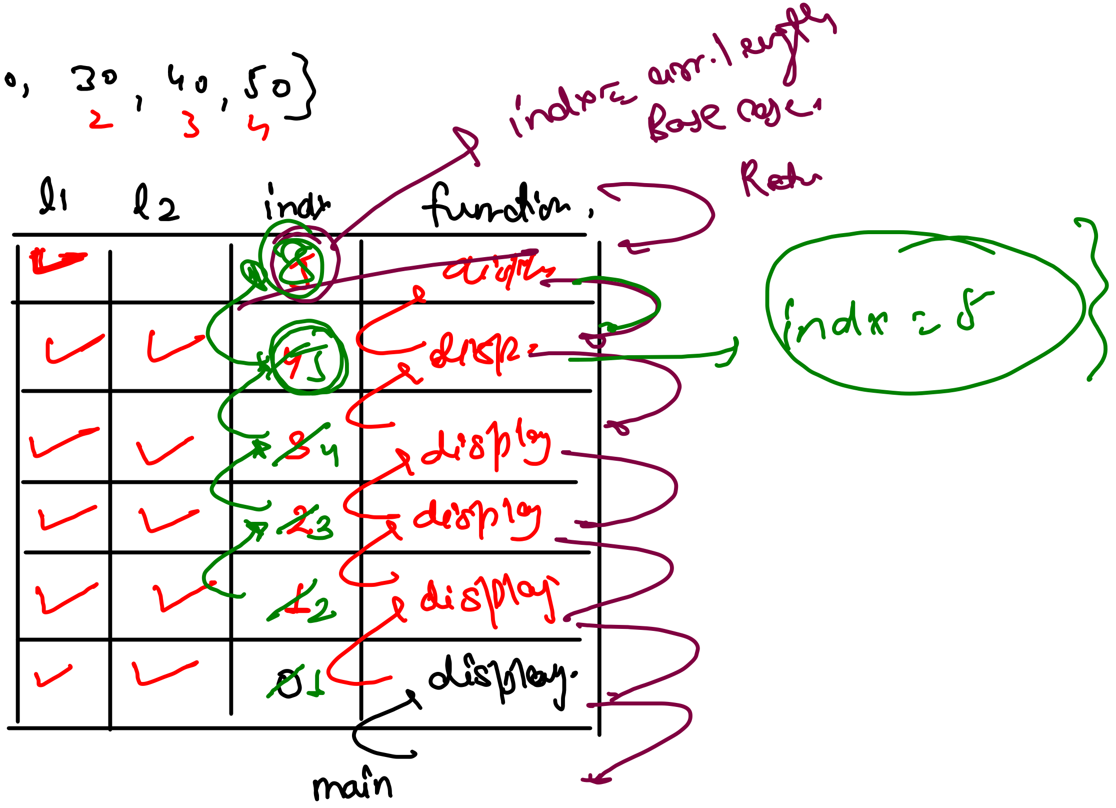
display (arr, i) → syso(arr[i]); display(arr, i+1);

arr → {10, 20, 30, 40, 50}
0 1 2 3 4

```
public static void display(int[] arr, int indx) {
1.  System.out.println(arr[indx]);
2.  display(arr, indx + 1);
}
```

→ Pre Area

10
20
30
40
50



Reverse Array.

indx ++ , ++indx

arr → 10, 20, 30, 40, 50

Expectation d.r.(arr, 0) → 50 40 30 20 10

faith displRev(arr, 1) → 50 40 30 20

indx = arr.length

Merging of faith & Expectation → display(arr, 0) = 50 40 30 20 10

display(arr, 0) = display(arr, 1) syso(arr[i]);

display(arr, i) = display(arr, i+1) syso(arr[i]);

Postorder

u's → arr, 2 ↑ arr, 1 ↑ arr, 0

Max of Array :

arr → {20, 15, 12, 11, 19, 1, 7}

0 1 2 3 4 5 6

arr[i]

rmax

Expectation , max(arr, 0) → 0 — to — (length-1)

20

faith , max(arr, 1) → 1 to (length-1) } → 19

Merging →

max (arr, 0) → 0 to (length-1)

max (arr, 0) = max (1 to length-1) → max(arr, 1)

= return (rmax vs arr[i]) =

Max of Array :

arr → {20, 15, 12, 11, 19, 1, 7}

0 1 2 3 4 5 6

arr[i]

rmax

Expectation , max(arr, 0) → 20 0 — to — (length −1)

faith , max(arr, 1) → 1 to (length −1)} → 19

Merging →

max (arr, 0) → 0 to (length −1)

max (arr, 0) = max (1 to length −1) → max (arr, 1)

= return (rmax vs arr[i]) =

```java
public static int maxOfArray(int[] arr, int indx) {
1.  int rmax = maxOfArray(arr, indx + 1);
    // comparsion between rmax vs arr[indx]
2.  return Math.max(rmax, arr[indx]);
}
```

arr → {-10, -7, -9, -5} — max = -5

0 1 2 3

Base case

l1	l2	rmax	indx	function
			4	max
✓	✓	-∞	3	max
✓	✓	-5	2	max
✓	✓	-9 -5	1	max
✓	✓	-5	0	max

main

res = 10

first Indx

int[] arr = { 1, 7, 6, 3, 2, 3, 2, 2, 5, 6 }

0 1 2 3 4 5 6 7 8 9

$\boxed{data = 2}$

first Indx = 4

Expectation: f.Indx (arr, 0, 2) → first Indx / -1

faith, → f.Indx (arr, 1, 2) → first Indx / -1 } Recursion

Merging → firstIndx (arr, 0, 2) = f. Indx / -1

firstIndx (arr, 0, 2) = ① check yourself, if yes return

② Ask from Recursion

Low level Analysis

```java
public static int firstIndex(int[] arr, int indx, int data) {

    if(arr[indx] == data)
        return indx;

    int rindx = firstIndex(arr, indx + 1, data);
    return rindx;
}
```

if(indx == arr.length) return -1;

arr → { 1, 3, 7, 2, 2, 5, 2 }
0 1 2 3 4 5 6

data = 2;

Trop

arr, 3, 2 — 3

arr, 2, 2 — 3

arr, 1, 2 — 3

arr, 0, 2 — 3

main = 3

arr, 7, 8 — -1

arr, 6, 8 — -1

arr, 5, 8 — -1

arr, 4, 8 — -1

arr, 3, 8 — -1

arr, 2, 8 — -1

arr, 1, 8 — -1

arr, 0, 8 — -1

main = -1

Last Indx

arr → {1, [3, 5, 7, 5, 15, 2, 8}]

0, 1, 2, 3, 4, 5, 6, 7

data = 5

Expectation, find (arr, 0, 5) → if present 4 5 / if not −1

faith → find (arr, 1, 5) → if present = 4 / if not −1

Merging of faith and Expectation, find (arr, 0, 5) → 0 to length-1 5

find (arr, 0, 5) =

find (arr, indx, data) =

rans = find (arr, indx + 1, data)

← check

```
rans = find (arr, 1, 5)
if (rans == -1) {
    if (arr[indx] == data)
        rans = indx;
}
return rans
```