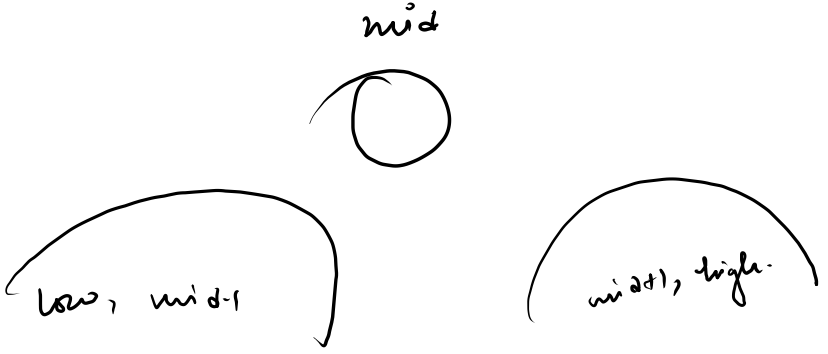
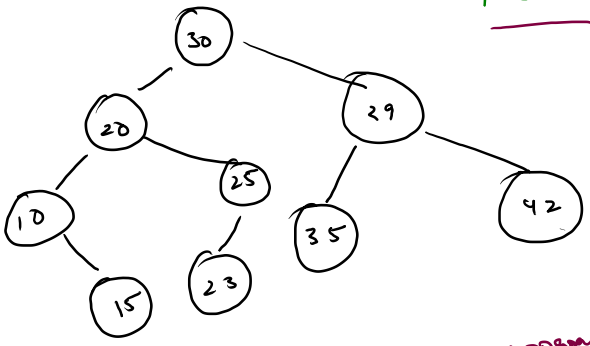
→ Recursion

Construction of BST using Inorder.



Construct BST using Pre order Traversal



Pre →

0	1	2	3	4	5	6	7	8
30	20	10	15	25	23	39	35	42

Inorder →

10	15	20	23	25	30	35	39	42

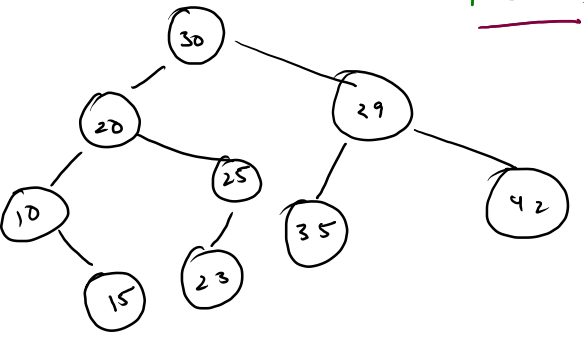
Approach-01

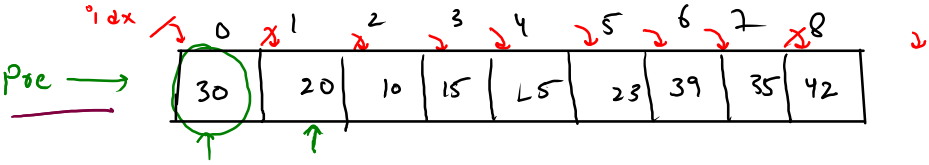
Construct BT using preorder and inorder

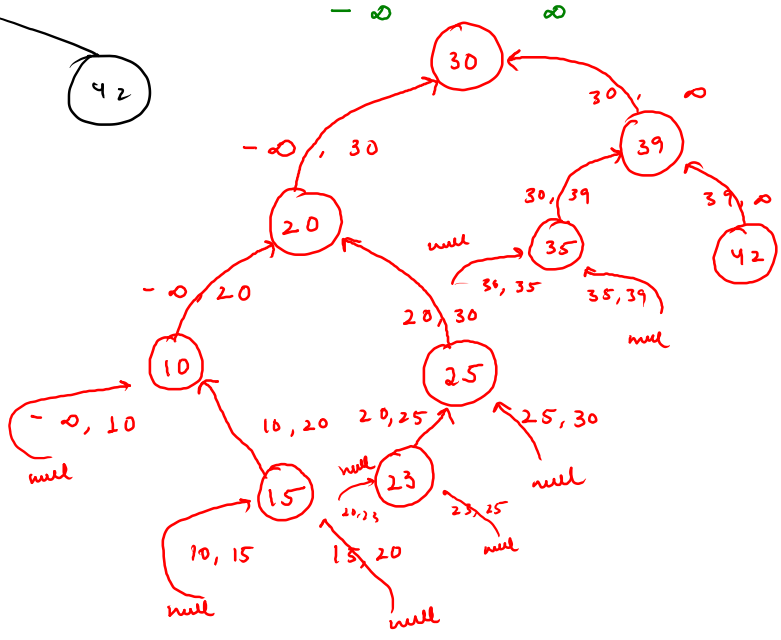
Inorder Traversal BST

↳ sort

N L R





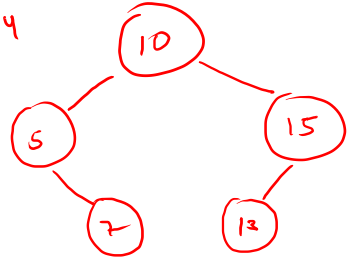


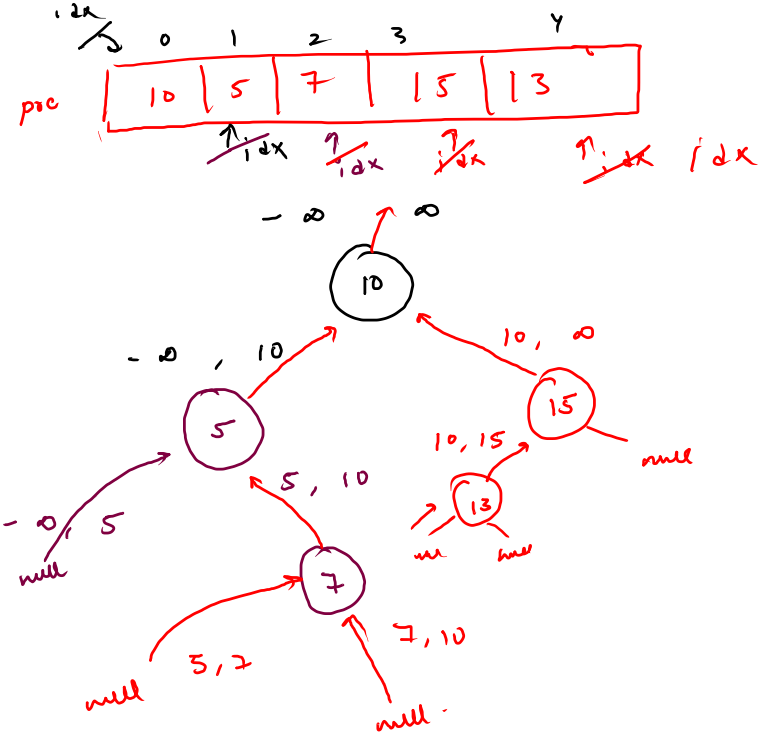
25

- ∞, 10 ℓ
- ∞, 20, ℓ
- ∞, 30 ℓ
- ∞ ∞ ℓ

```java
public static int idx = 0;
public static Node helper(int pre[], int low,int high){
    if(idx == pre.length || pre[idx] < low || pre[idx]>high){
        return null;
    }

    Node node = new Node(pre[idx++], null,null);
    node.left = helper(pre,low,node.data);
    node.right = helper(pre,node.data,high);

    return node;

}

public static Node constructFromPreorder(int [] preorder){
    return helper(preorder,-(int)1e8, (int)1e8);
}
```

idx = 0 ~~1~~ ~~2~~ ~~3~~ 4





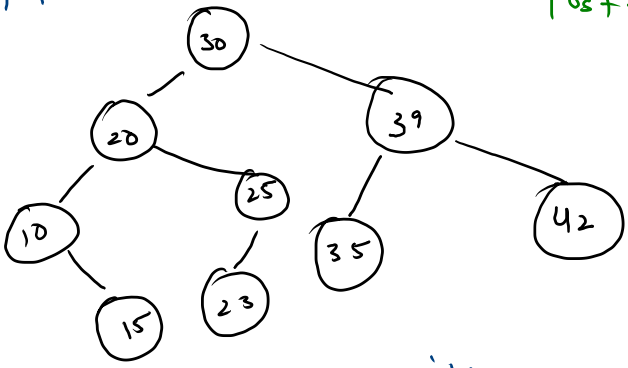
⇒ idx → iterating over the preorder array.

⇒ Range $-\infty$, ∞.

base case

⇒ idx == pre.length, pre[idx] < lo, pre[idx] > rr.

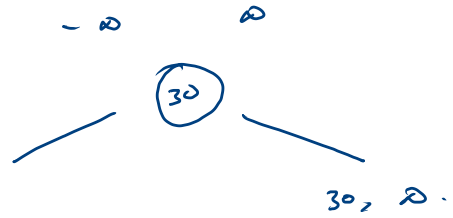
⇒ null.



idx = post.length - 1;

Post →

15	10	23	25	20	35	42	39	30



idx--

→ Create node.

→ Right side

→ left side.

```java
public static int idx;
public static Node helper(int post[], int low,int high){
    if(idx <0 || post[idx] < low || post[idx]>high){
        return null;
    }

    Node node = new Node(post[idx--], null,null);
    node.right = helper(post,node.data,high);
    node.left = helper(post,low,node.data);

    return node;

}

public static Node constructFromPostorder(int [] postorder){
    idx = postorder.length-1;
    return helper(postorder,-(int)1e8, (int)1e8);
}
```

5 min.

→

+

10 mins

↓

break

11:50 pm

→ Recursion :- Solving a problem using subproblems

Factorial (5) ⇒ 5! = 5 * 4 * 3 * 2 * 1

↳ 5 * (4!)

↳ 4 * 3!

↳ 3 * 2!

↳ 2 * 1!

↳ 1 * 0!

5 * 4!

↳ 4 * 3!

↳ 3 * 2!

↳

Steps :

→ expectation → Factorial(n)

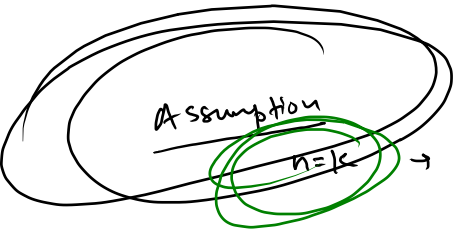
→ Faith / Assumption → Factorial(n-1) ⇒ x

→ My work + n * x → Factorial(n)

→ Dry Run

Sum of n natural nos.

↳ $S_1 = \frac{1(1+1)}{2} = 1$



→ $S_k = \frac{k(k+1)}{2}$

$n = (k+1)$

$1 + 2 + \cdots + k + (k+1)$

↳ $\frac{k(k+1)}{2} + (k+1)$ → $\frac{(k+1)(k+2)}{2}$

→ $\frac{(k+1)(k+1+1)}{2}$ ∎

Print Increasing (Day 23)

Problem | Submissions | Leaderboard | Discussions

1. You are given a positive number n.
2. You are required to print the counting from 1 to n.
3. You are required to not use any loops

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Sample Input 0

```
5
```

Sample Output 0

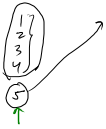
```
1
2
3
4
5
```

```
void printIncreasing(int n){
    if(n==0) return;
```

n = 5
↳ n = 4
↳ n = 3
↳ n = 2
↳ n = 1
↳ n = 0 → stops.
↳ n = -1
↳ n = -2

Infinite recursion

n = 5



Expectation :- Print increasing order for no. 'n' i.e n=5

Faith / Assumption :- Print increasing order for no (n-1) i.e n=4.

Mywork :- Print no. 'n' i.e. n=5

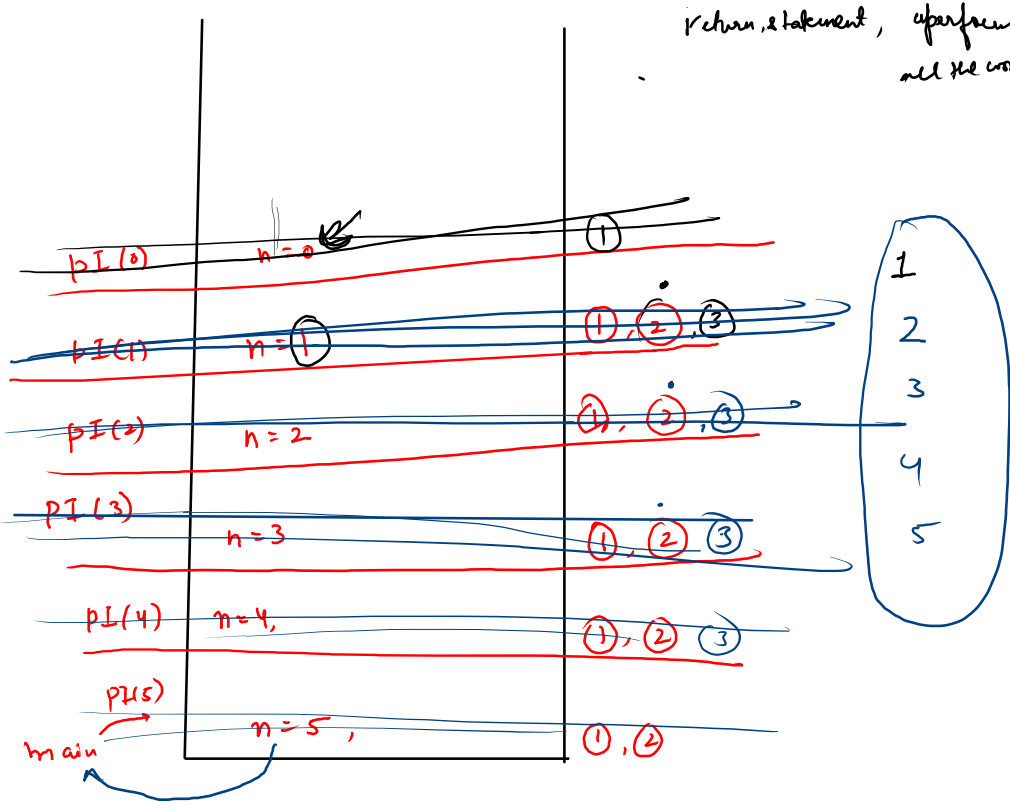
```
    printIncreasing(n-1);
    System.out.println(n);
}
```

Stopping condⁿ → base case

```
void pI (int x) {
    if (n == 0)
        return;
    pI (n-1);
    Sysout(n);
}
```

main

pI(n)



```java
public static void pI(int n){
    if(n==0){
        return;
    }

    pI(n-1);
    System.out.println(n);
}
```

power - linear :-

↳ x , n ⇒ x^n

↳ x=2, n=5 ⇒ 2^5 → 32

$x^n \Rightarrow x^{n-1} * x$

$2^5 \Rightarrow 2^4 * 2$

Expectation → Find x^n

Faith / Assumption :- Find x^{n-1}

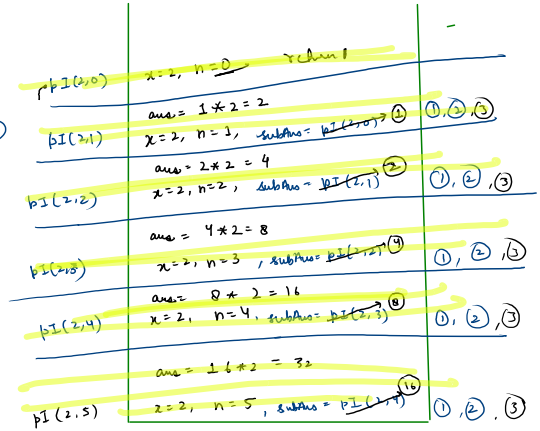
My work :- Multiply with 'x'

$x^n \rightarrow x^{n-1}$

$2^5 \rightarrow 2^4 \rightarrow 2^3 \rightarrow 2^2 \rightarrow 2^1 \rightarrow 2^0$ → 1

p.s.

```
int pL (int x, int n) {
    if (n == 0)
        return 1;                    → (1)

    int subAns = pL (x, n-1);        → (2)

    int ans = subAns * x;            → (3)
    return ans;
}

main → 32
```



```java
public class Solution {
    public static int pL(int x, int n){
        if(n==0){
            return 1;
        }

        return pL(x,n-1)*x;
    }

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print output to STDOUT. Y
        Scanner scn = new Scanner(System.in);
        int x = scn.nextInt();
        int n = scn.nextInt();

        System.out.println(pL(x,n));
    }
}
```

T.C ?

n → n-1, n-2, n-3 ——— 0 ⇒ O(n)

2^5 $n = 5$, calls = 6 ↓ (n+1)

2^{64} ⇒ 65 calls

↳ 2^4

↳ 2^3

↳ 2^2

↳ 2^1

↳ 2^0

$2^{64} \Rightarrow 2^{32} * 2^{32}$ → n/2

$2^{32} \Rightarrow 2^{16} * 2^{16}$

$2^{16} \rightarrow 2^{8} * 2^{8}$

$2^{8} \rightarrow 2^{4} * 2^{4}$

$2^{4} \rightarrow 2^{2} * 2^{2}$

$2^{2} \rightarrow 2^{1} * 2^{1}$

$2^{1} \rightarrow 2^{0} * 2^{0} * 2$

$2^{64} \rightarrow 65 \leftrightarrow 8$

T.C. $\log_2(n)$

x^n

$2^{18} \rightarrow 2^{9} * 2^{9}$

$2^{9} \rightarrow 2^{4} * 2^{4} * 2$

$2^{4} \rightarrow 2^{2} * 2^{2}$

$2^{2} \rightarrow 2^{1} * 2^{1}$

$2^{1} \rightarrow 2^{0} * 2^{0} * 2$

$2^{0} \rightarrow 1$

$2^{4+4} = 2^{8}$

∴ for odd → need to multiply with extra 'x'

Expectation :- Find x^n

Faith / Assumption :- Find $x^{n/2}$

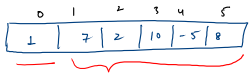
My work :- if ($n \rightarrow$ odd) $\Rightarrow$ Multiply it with extra x

Dry Run :-

```java
public class Solution {
    public static int pL(int x, int n){
        if(n==0){
            return 1;
        }

        int subAns = pL(x,n/2);
        int ans = subAns*subAns;
        if(n%2!=0) ans *= x;
        return ans;
    }
```

Recursion with Array

Display an array



Expectation :- display an array elements from (0, n-1)

Faith / Assumption :- display an array elements from (1, n-1)

My work :- Sysout (arr[i]).

Dry Run

```
O/P
1
7
2
10
-5
8
```

```
p.s void display (arr, idx) {
    if (idx == arr.length)  → (1)
        return;
    Sysout (arr[idx]) → (2)
    display (arr, idx+1); → (3)
}
```