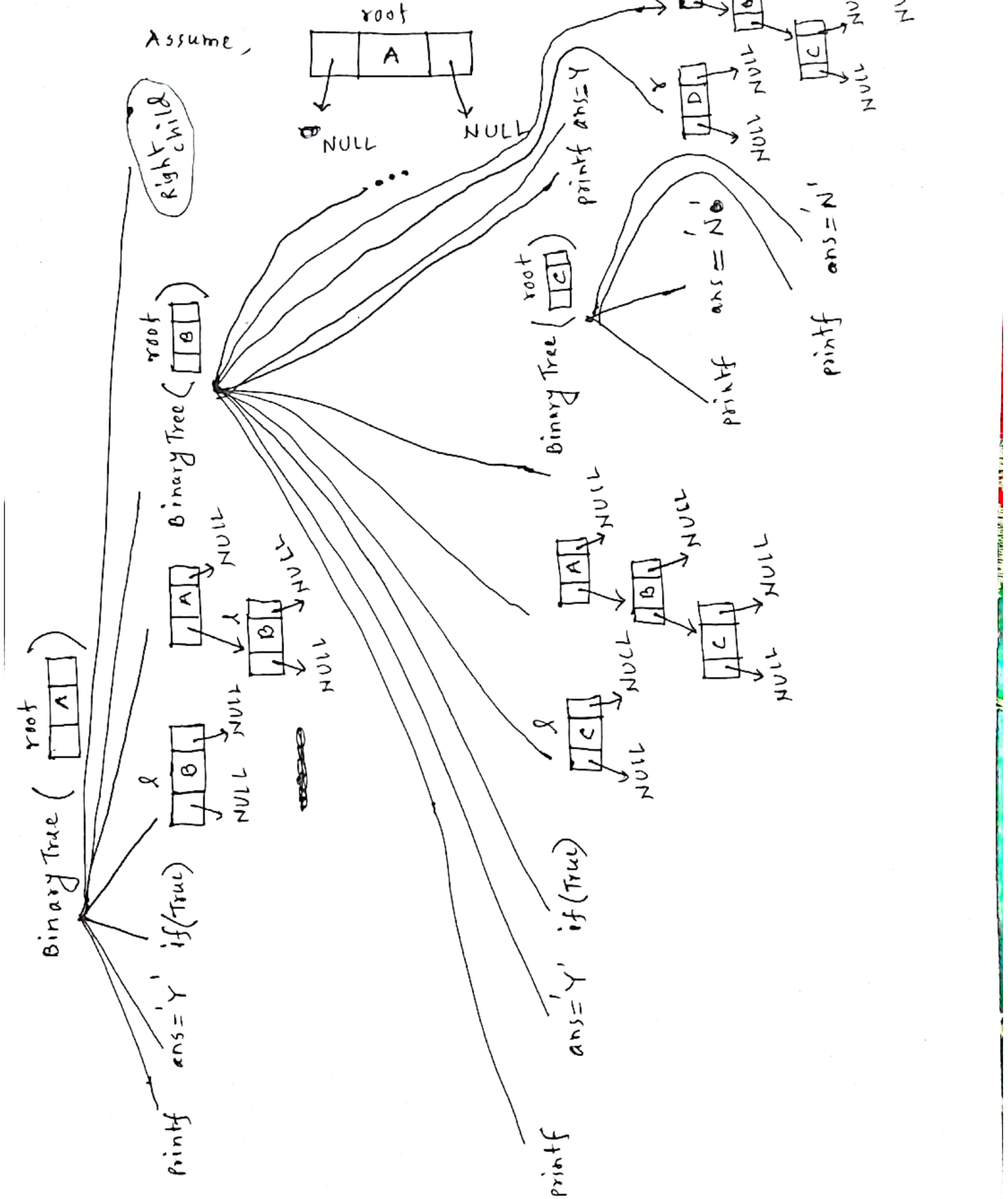
Binary Tree Construction using linked list

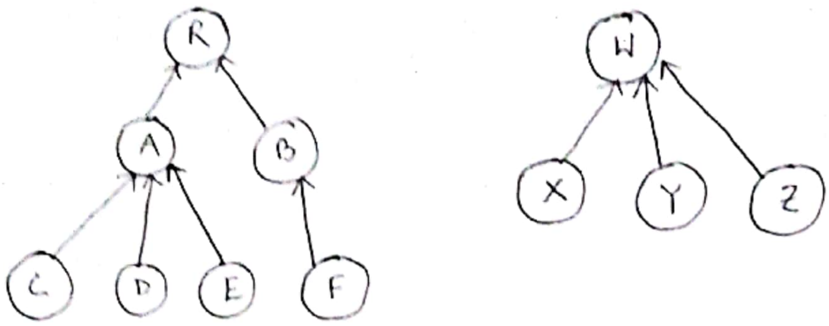
```
void BinaryTree(node *root){
    node *l, *r; char ans;

    if(root != NULL){
        printf("\nDo you want to create Left child(Y/N):=");
        ans= toupper(getche());
        if(ans=='Y'){
            l=(node*)malloc(sizeof(node));
            printf("\nEnter data:=");
            printf("Enter left child of %c", root->data);
            root->left
            l->data = getche();
            l->left = NULL;
            l->right = NULL;
            root->left = l;
            BinaryTree(l);
        }
        printf("\nDo you want to create right child(Y/N):=");
        ans = toupper(getche());
        if(ans=='Y'){
            r=(node*)malloc(sizeof(node));
            printf("Enter right child of %c", root->data);
            r->data = getche();
            r->left = NULL;
            r->right = NULL;
            root->right = r;
            BinaryTree(r);
        }
}
```

Binary Tree construction

Assume,



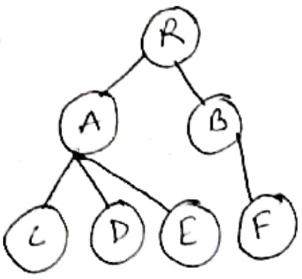
Parent Pointer Implementation

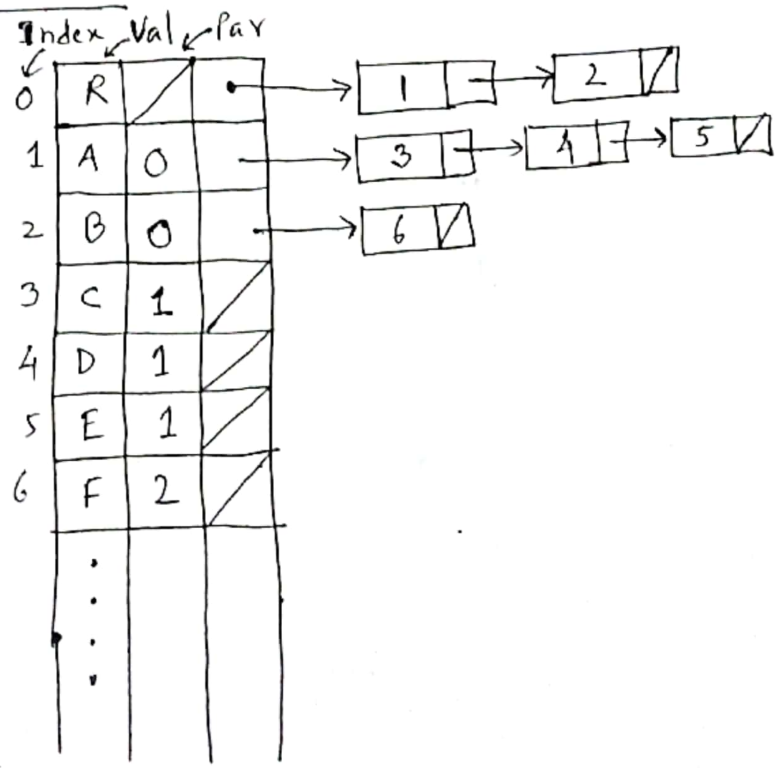


Parent's index	/	0	0	1	1	1	2	/	7	7	7
Label	R	A	B	C	D	E	F	W	X	Y	Z
Node index	0	1	2	3	4	5	6	7	8	9	10

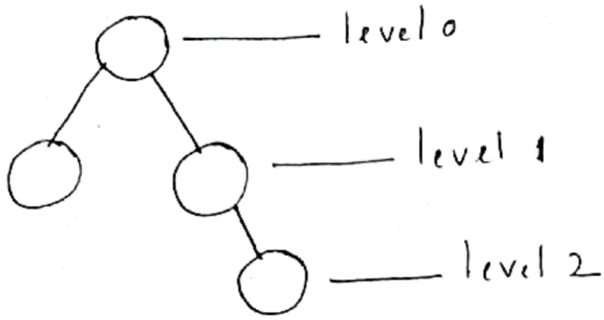
- Easy to find ancestors
- Difficult to find children
- Minimum space

General Tree Implementations





Height of a binary Search Tree:



Method-1

$$\text{height} = \max(\text{level}_i) = 2$$

Method-2

$$\text{height} = \lceil \log_2 n \rceil = \lceil \log_2 4 \rceil = \lceil \log_2 2^2 \rceil$$

$$= \lceil 2 \log_2 2 \rceil$$

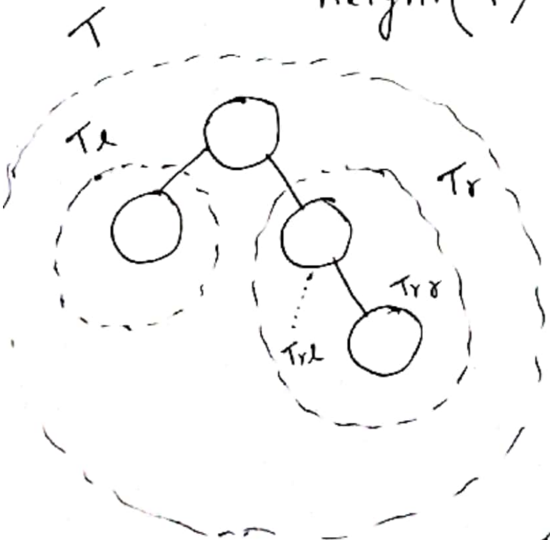
$$= \lceil 2 * 1 \rceil$$

$$= 2$$

Method-3

~~height(T) = max(height(~~

$$\text{height}(T) = \max(\text{height}(T_l), \underline{\text{height}(T_r)}) + 1$$



$$= \max(0, \underline{\max(\text{height}(T_{rl}), \text{height}(T_{rr}))+1}) + 1$$

$$= \max(0, \max(-1, 0)+1)+1$$

$$= \max(0, 0+1)+1$$

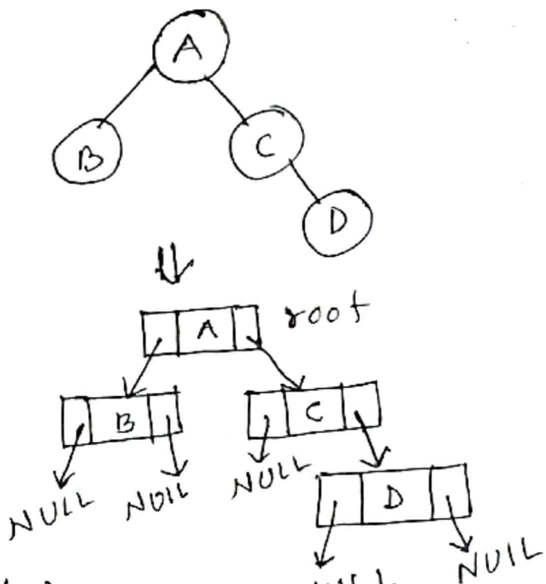
$$= \max(0, 1)+1 = 1+1 = 2$$

Tree	height
NULL	-1
O	Zero
otherwise	$\max(\text{height}(T_l), \text{height}(T_r))+1$

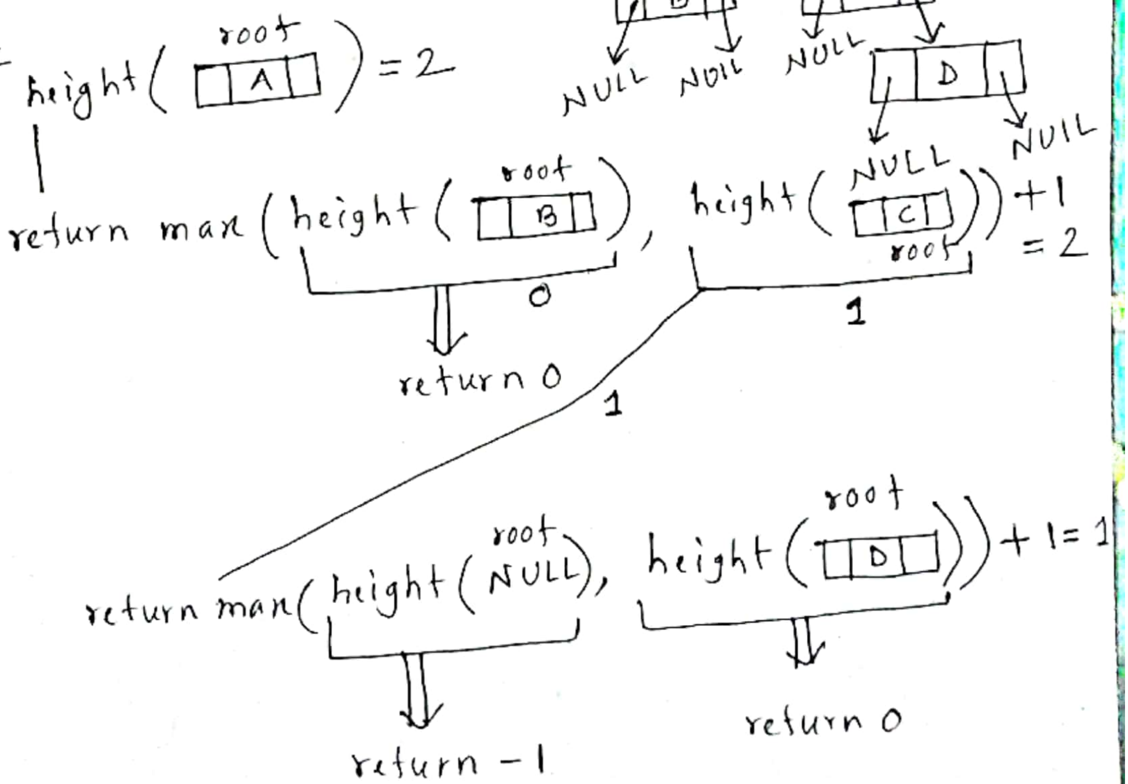
Algorithm of height calculation (Method)

```
int height(node *root){
    if (root == NULL)
        return -1;
    else if((root->left == NULL) && (root->right == NULL))
        return 0;

    else
        return max(height(root->left), height(root->right))
                      +1;

}
```

```
int max(int x, int y){
    if (x>y)  return x;
    else  return y;
}
```

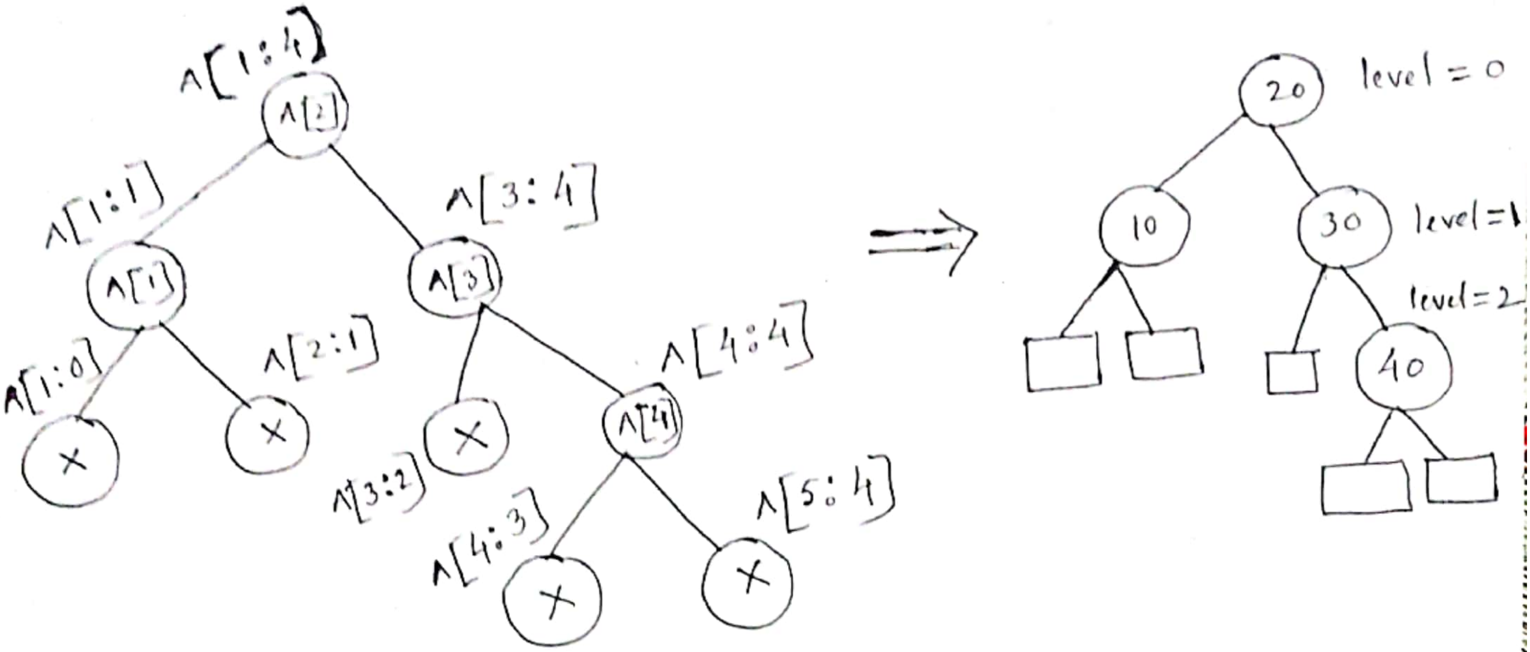


Recursion Tree



Average Number of Successful and Unsuccessful comparison:

A	10	20	30	40
	1	2	3	4



Average Number of Successful Comparison:

[Use internal node ○ level]

For 20 element Comparison = 1

For 10 " " = 2

For 30 " " = 2

For 40 " " = 3

$$\text{Average Comparison} = \frac{1+2+2+3}{4} = \frac{8}{4} = 2$$

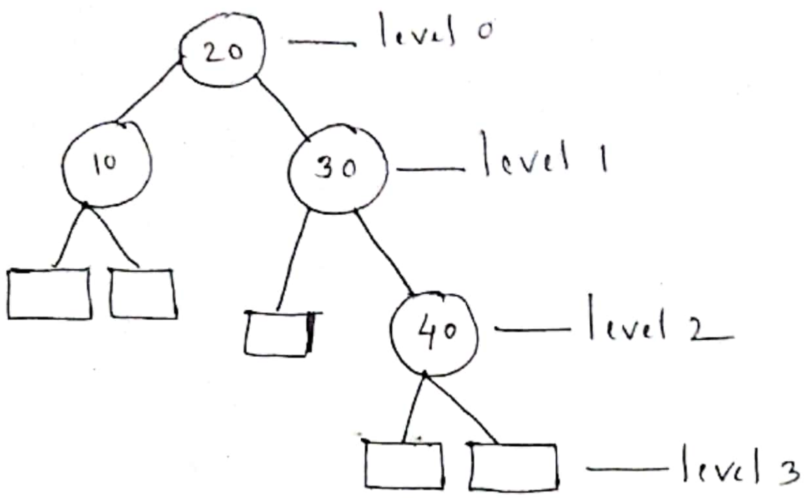
Using Formula:

$$\text{Average Comparison} = \frac{\sum(l_i+1)}{n}$$

$$= \frac{(0+1)+(1+1)+(1+1)+(2+1)}{4}$$

$$= 2$$

Average Number of Unsuccessful Comparison



Use external node level

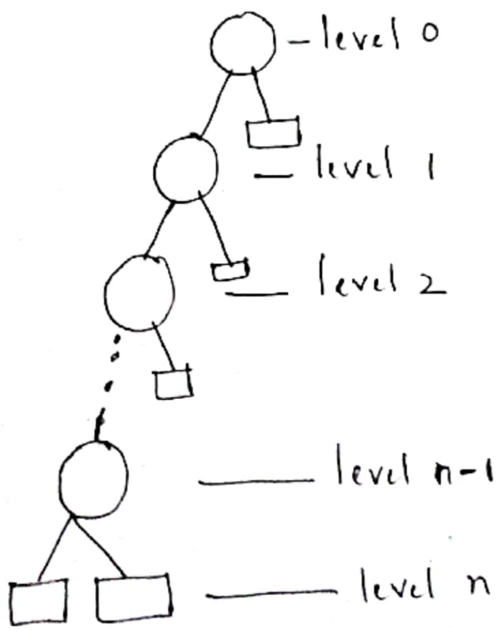
$$\text{Average comparison} = \frac{\sum l_i}{n+1}$$

$$= \frac{2+2+2+3+3}{5}$$

$$= \frac{12}{5}$$

$$= 2.4$$

Internal and External Path length calculation for one sided binary Search Tree:



$I(T)$ = Internal Path length

$$= 0+1+2+3+\cdots+\overline{n-1}$$

$$= \frac{(n-1)n}{2} \quad \ldots \text{eq } 1$$

$$\left[1+2+3+\cdots+n = \frac{n(n+1)}{2}\right]$$

$E(T)$ = External Path length

$$= \underline{\underline{1+2+3+\cdots+n}}+n$$

$$= \frac{n(n+1)}{2}+n = n\left\{\frac{n+1}{2}+1\right\}$$

$$= \frac{n(n+3)}{2} \quad \ldots \text{eq } 2$$

$$E(T)-I(T)$$

$$= \frac{n(n+3)}{2} - \frac{(n-1)n}{2}$$

$$= \frac{n}{2}\left\{(n+3)-(n-1)\right\} = \frac{n}{2}\times 4 = 2n$$

$$\boxed{E(T)-I(T)=2n}$$