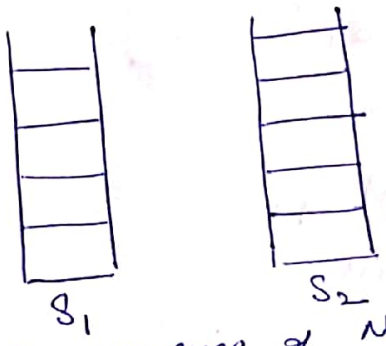
Name: Mohd. Zaid
Enroll: GL3125
Faculty No: 18COB103
S.No: 40

COC2060

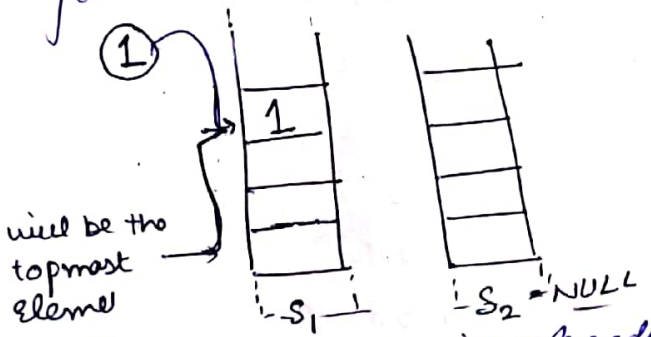
1. Implementation of queue using 2 stacks.

Let us consider 2 stacks, S_1 & S_2 as them of very large sizes.

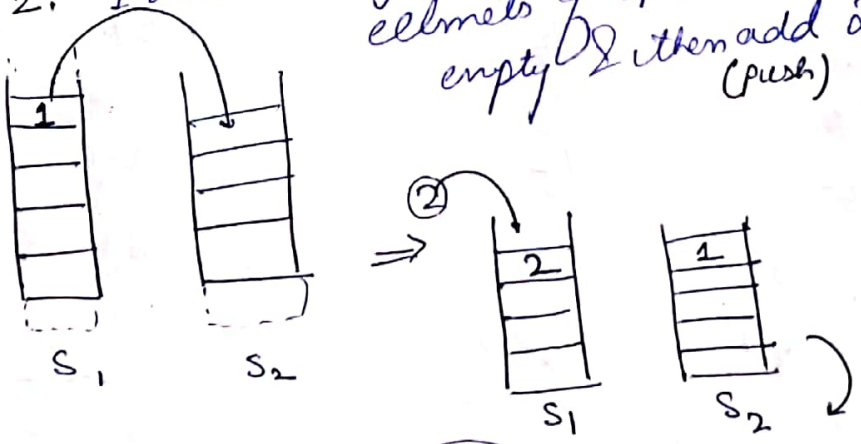


Let us consider enqueuing of No, (1, 2, 3).

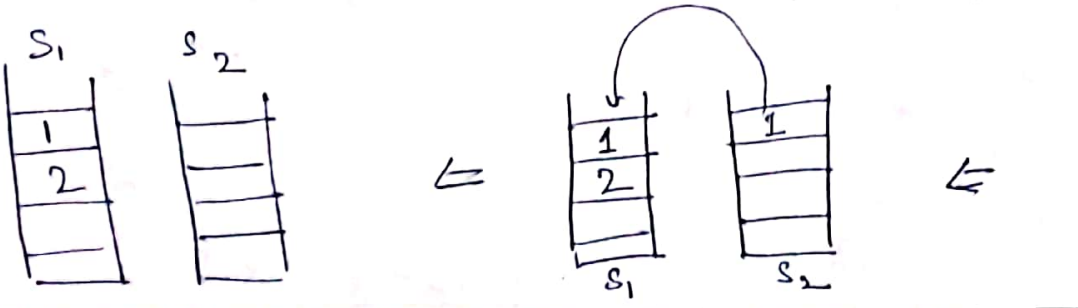
for 1: 1 will be the sole element in the stack.



for adding 2: 1 will already be present. Now shift all the elements of S_1 to S_2 until S_2 is empty & then add 2 to S_1 (push)

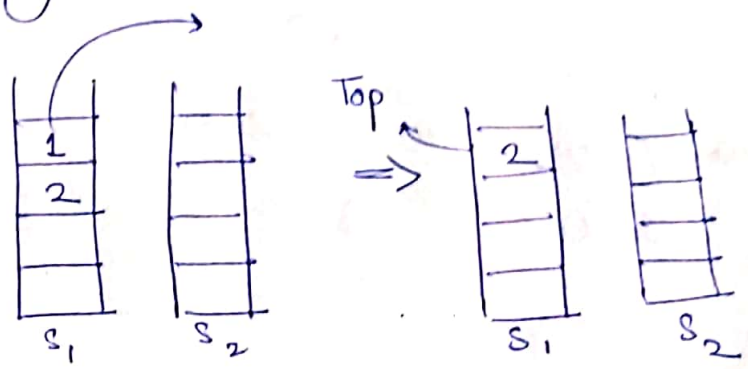


Now popping 1 from the S_2 & pushing into S_1



Thus inserting an element using 2 stacks implementation is O(n).

- For deleting an element takes constant time ie O(1) simply poping the top element from stack.



Q2 Trasform.

a) (A+B)* (c-D)^E*F

b) (A+B)*(c^(A(D-E)+F)-G

to infin and Post fin.

a) (A+B)* (c-D)^E^F

Scanned	Stack	Postfin
(	(	
(A	(	A
(A+	(+	A
(A+B	(+	AB
(A+B)	~~(+)~~	AB+
(A+B)*	*	AB+
(A+B)*(	*(	AB+
(A+B)*(c	*(	AB+C
(A+B)*(c-	*(-	AB+C
(A+B)*(c-D	*(-	AB+CD
(A+B)*(c-D)	~~*(-)~~	AB+CD-
(A+B)*(c-D)^	*^	AB+CD-
(A+B)*(c-D)^E	*^	AB+CD-E
(A+B)*(c-D)^E*	*	AB+CD-E^*
(A+B)*(c-D)^E^F	*	AB+CD-E^*F

∴ Postfin = AB+CD-E^*F*

b) (A+B)*(C^(D-E)+F)-G

Scanned	Stack	Postfix
(	(	
(A	(	A
(A+	(+	A
(A+B	(+	AB
(A+B)	~~(+)~~	AB+
(A+B)*	*	AB+
(A+B)*(	*(	AB+
(A+B)*(C	*(	AB+C
(A+B)*(C^	*(^	AB+C
(A+B)*(C^(	*(^(	AB+C
(A+B)*(C^(D	*(^(	AB+CD
(A+B)*(C^(D-	*(^(-	AB+CD
(A+B)*(C^(D-E	*(^(-	AB+CDE
(A+B)*(C^(D-E)	*(^	AB+CDE-
~~(A+B)*(C^(D-E)~~	~~*(+~~	~~AB+CDE-^+~~
~~(A+B)*(C^(D-E)+~~	~~*(+~~	~~AB+CDE-^F~~
~~(A+B)*(C^(D-E)+F~~	~~*~~	~~AB+CDE-^F+*~~
(A+B)*(C^(D-E)+	*(+	AB+CDE-^
(A+B)*(C^(D-E)+F	*(+	AB+CDE-^F
(A+B)*(C^(D-E)+F)	*(+)	AB+CDE-^F+
(A+B)*(C^(D-E)+F)-	*-	AB+CDE-^F+*
(A+B)*(C^(D-E)+F)-G	-	AB+CDE-^F+*G-

Ans AB+CDE-^F+*G-

a) $(A+B)*(C-D)\wedge E*F$

to Prefix.

Reversing the Expression:

F*E^)D-C(*)B+A(

Scan	Stack	Prefix
F		F
F*	*	F
F*E	*	FE
F*E^	*^	FE
F*E^)	*^)	FE
F*E^)D	*^)	FED
F*E^)D-	*^)-	FED
F*E^)D-C	*^)-	FEDC
F*E^)D-C(	*^)-	FEDC-
F*E^)D-C(*	*^)*^	FEDC-**
F*E^)D-C(*)	*^)*	FEDC-^*
F*E^)D-C(*)B	*^)*	FEDC-^*B
F*E^)D-C(*)B+	*^)+	FEDC-^*B
F*E^)D-C(*)B+A	*^)+	FEDC-^*BA
F*E^)D-C(*)B+A(	*^)+	FEDC-^*BA+

∴ FEDC-^*BA+* reversed

+AB^-CDEF

Ans.

D) $(A+B)*(C\hat{}(D-E)+F)-G$

Reversing $G-)F+)E-D(\hat{}C(*)B+A)$

Scan	Stack	Prefix
G		G
G-	-	G
G-)	-)	G
G-)F	-)	GF
G-)F+	-)+	GF
G-)F+)	-)+)	GF
G-)F+)E	-)+)	GFE
G-)F+)E-	-)+)-	GFE
G-)F+)E-D	-)+)-	GFED
G-)F+)E-D(	-)+)-(	GFED-
G-)F+)E-D(^	-)+)-(^	GFED-
G-)F+)E-D(^C	-)+)-(^	GFED-C
G-)F+)E-D(^C(	-)+)-(^(	GFED-C+^
G-)F+)E-D(^C(*	-)+)-(^(*	GFED-C+^
G-)F+)E-D(^C(*)	-)+)-(^(*)	GFED-C+^
G-)F+)E-D(^C(*)B	-)+)-(^(*)	GFED-C+^B
G-)F+)E-D(^C(*)B+	-)+)-(^(*)+	GFED-C+^B
G-)F+)E-D(^C(*)B+A	-)+)-(^(*)+	GFED-C+^BA
	-)+)-(^(*)+	GFED-C+^BA+
	-)+)-(^(*)+	GFED-C+^BA+*-

∴ reversed = GFED-C+^BA+*-

∴ Ans = $+*-AB\hat{}+C-DEFG$

Here are your random numbers:

17	62	3	61	55
58	4	84	8	81
80	28			

Timestamp: 2020-11-30 14:39:40 UTC

Again! Go Back

Note: The numbers are generated left to right, i.e., across columns.

Follow us: Twitter | Facebook
Terms and Conditions
About Us

Q3

Generate 12 random No from <https://...>

1. Sort using.
   a) Radix Sort
   b) QuickSort.
   c) MergeSort.

No generated are:

17, 62, 03, 61, 55, 58, 04, 84, 08, 81, 80, 28

a) Radix Sort

(1) Considering the one's place:

Input array: 17 62 03 61 55 58 04 84 08 81 80 28

Arranging No acc^n to one's place:-

80, 61, 81, 62, 03, 04, 84, 55, 17, 58, 08, 28
(0) (1) (1) (2) (3) (4) (4) (5) (7) (8) (8)

since 61 comes before 81 similarly as 58 comes 08 in the input array

(2) Considering the 10's place of the No's after step 1;

80 61 81 62 03 04 84 55 17 58 08 28

Arranging No's acc^n to the 10's place:

58

03 04 08 17 28 55 61 62 80 81 84

Completed Sorted array.

| 03 04 08 17 28 55 58 61 62 80 81 84 |

Quick Sort:

b) 17 62 3 61 55 58 4 84 8 81 80 28

0 1 2 3 4 5 6 7 8 9 10 11

↑ pivot

Selecting Pivot

Move pivot to end

17 62 3 61 55 28 4 84 8 81 80 58

Pick from the array

Move from Left to right until we find value equal to pivot.

Move right Bound until it crosses the left bound or find value less than pivot

17 8 3 61 55 28 4 84 62 81 80 58

interchange

17 8 3 4 55 28 61 84 62 81 80 58

when pivots crosses:

17 8 3 4 55 28 58 84 62 81 80 61

Calling quick Sort Now on left list

17 8 28 4 55 3 58

3 8 28 4 55 17 58

↑ pivot

3 8 28 17 55 4

3 4 28 17 55 8

3 4 28 8 55 17

3 4 8 28 55 17

3 4 8 17 55 28

3 4 8 17 28 55 58 84 62 81 80 61

Scanned with CamScanner

3 4 8 17 28 55 | 58 | 84 62 61 80 | 81 |

Sorting in right sublist

- - - - - - | 58 | 80 62 61 84 | 81

- - - - | 58 | 80 62 61 | 81 | 84

| 58 | 80 61 62 | 81 | 84

| 58 | | 61 | 80 | 62 | 81 | 84

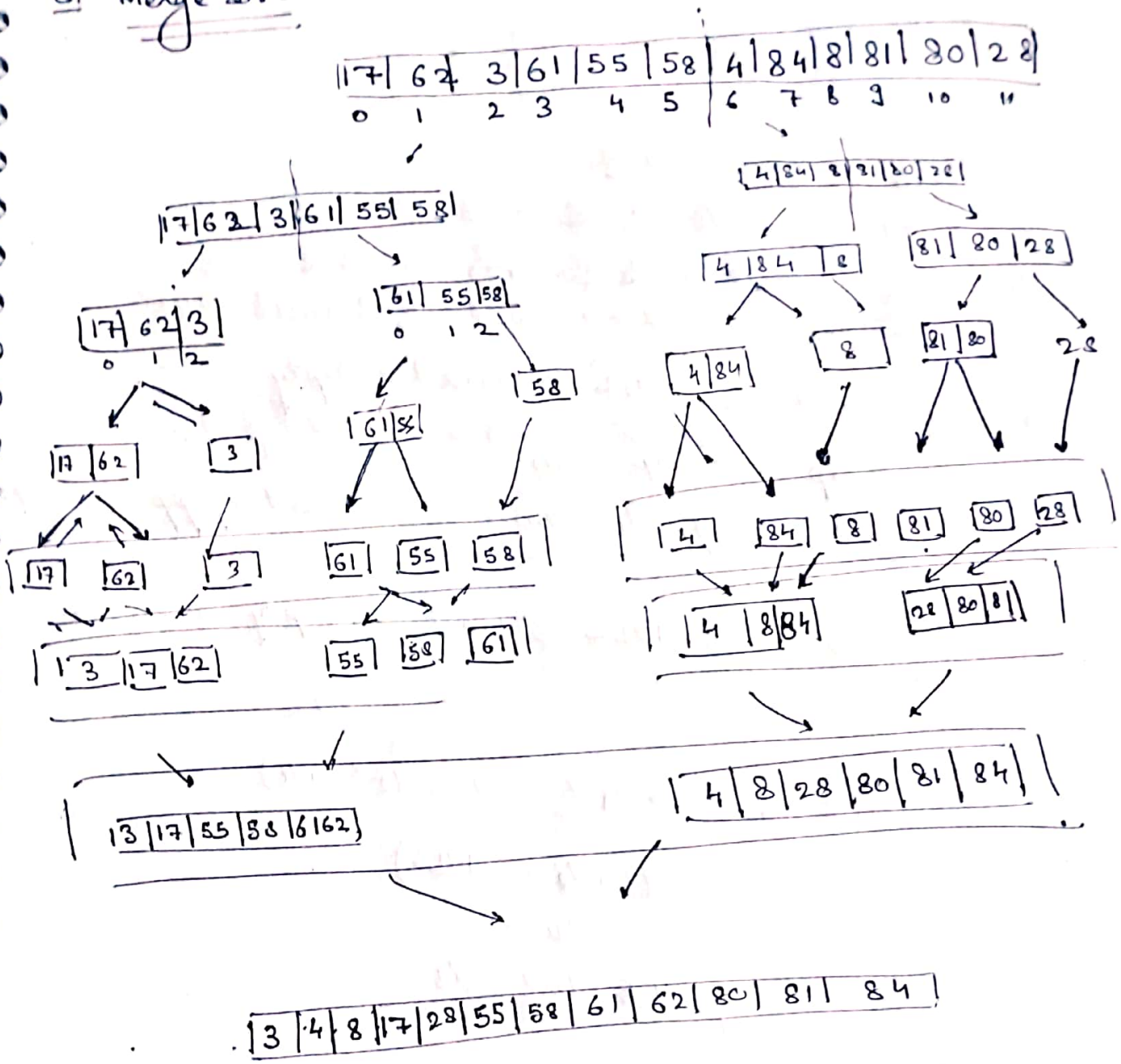
| 58 | | 61 | | 62 | 80 | 81 | | 84 |

∴ Sorting finished as right sublist have only one element.

Sorted array:

| 03 | 04 | 8 | 17 | 28 | 55 | 58 | 61 | 62 | 80 | 81 | 84 |

a. Merge Sort



3 4 8 17 28 55 58 61 62 80 81 84

Inorder Traversal:

Algo:

1. Create an Empty stack.
2. Initialize current node as the root.
3. Push current node contents to stack and set the current → current → left until NULL.
4. If current is NULL & stack != Empty
   a) Pop the top item.
   b) Print popped item. & set current ⇒ popped → right.
   Repeat (3).
5. If current is NULL & stack is Empty we are done.

void printInorder (struct node* node)

Past order using stack :

1. Push root to first stack
2. while first ! Empty
   push node from first to second.
   push left node &
   push right node children
   and pop node to first stack.
3. Print contents.

Code :

```
//used stack stl
void printPastorder(struct node *node)
{
    if (node == NULL)
        return;
    stack<node*> stack; node* current = node;
    while (current != NULL || stack.empty() == false)
    {
```

```
while (current != NULL)
    {
    if (current -> right)  // if it exist
        {
        stack.push (current -> right);
        }
        Stack.push (current);
        current = current -> left;
    }

    node* current = stack.pop();
if (current -> right && stack.top() == current ->
                                        right)
        stack.pop();
        Stack.push();
        current = current -> right;
    }
        { cout << current -> data;
          current = NULL;
        }
    }
}
```

Inorder Code

```
void printInorder(node* node)
{
    if(node == NULL)
        return;
    Stack <node*> stack;
    node *current = root;
while(current != NULL || stack.empty() == false)
{   while(current != NULL)
    {   stack.push(current);
        current = current -> left();
    }
    current = stack.top();
    stack.pop();
    cout << current -> data;
    current = current -> right;
}
```

Preorder

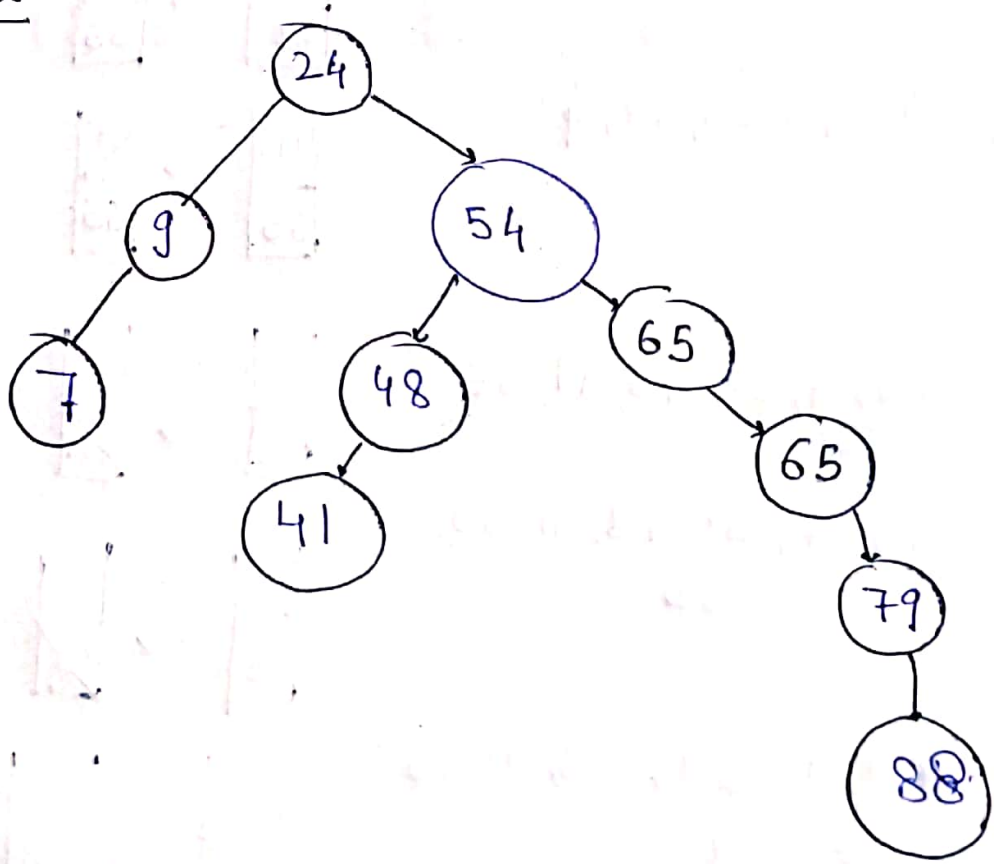
```
void printPreorder(node* node)
{   if(node == NULL)
        return;
    Stack <node*> stack;
    stack.push(node);
    while(stack.empty() == false)
    {
```

```
Node * node1 = stack.top();
cout << node1->data;
if (node1->right != NULL)
    stack.push(node1->right);
if (node1->left != NULL)
    stack.push(node->left);
}
```

Random no generated

24 54 65 48 9
65 79 7 88 41

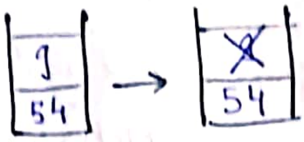
Binary Tree Generated



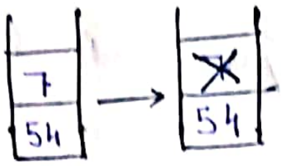
Preorder:

24

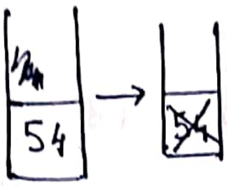
24, 9



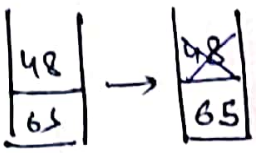
24, 9, 7



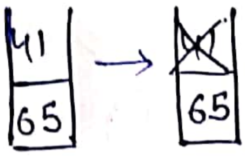
24, 9, 7, 54, ~~78~~



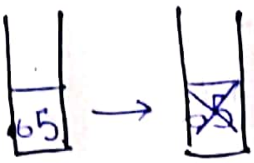
24, 9, 7, 54, 48



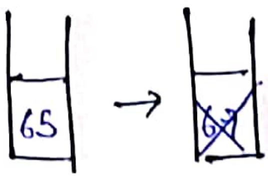
24, 9, 7, 54, 48, 41



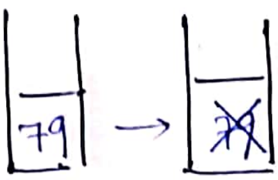
24, 9, 7, 54, 48, 41, 65



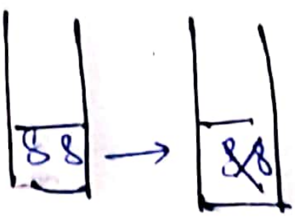
24, 9, 7, 54, 48, 41, 65,
65



24, 9, 7, 54, 48, 41, 65
65, 79



24, 9, 7, 54, 48, 41, 65,
65, 79, 79, 88



Inorder:

7

7 9

7 9 24

7 9 24 41

7 9 24 41 48

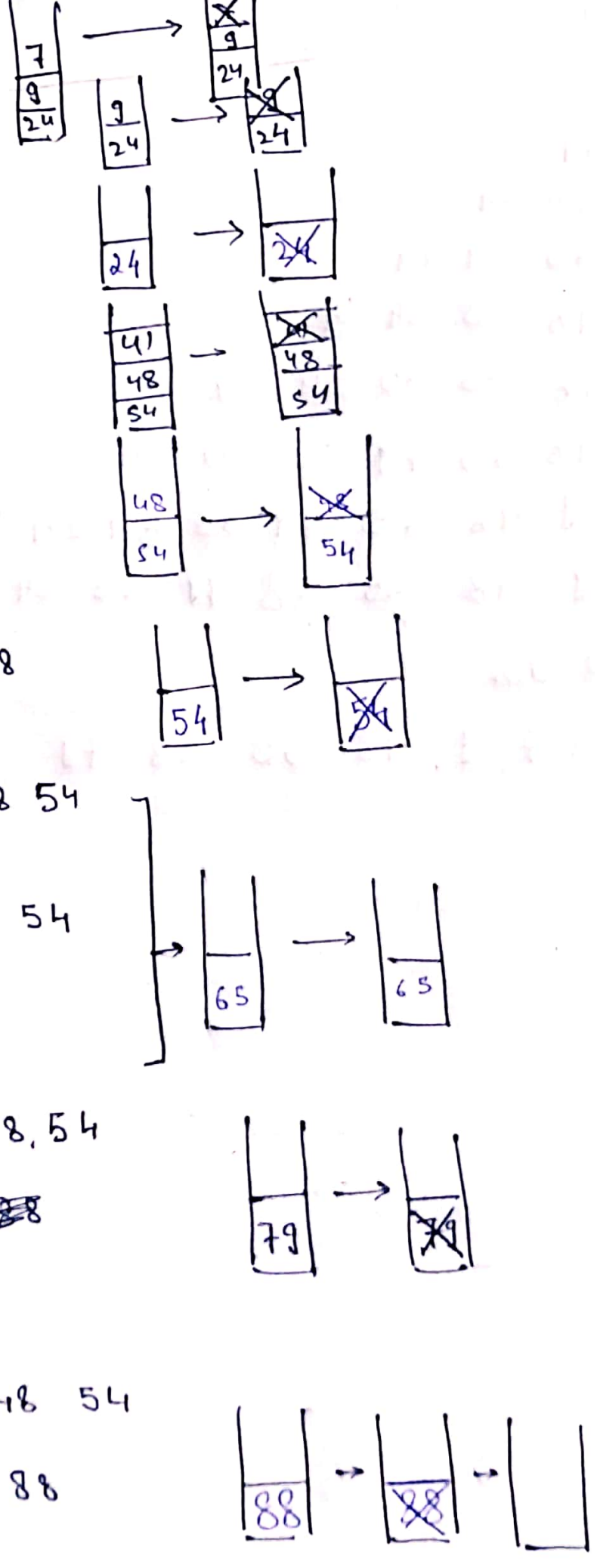
7 9 24 41 48
54

7 9 24 41 48 54
65

7 9 24 41 48 54
65 65

7 9 24 41 48 54
65, 65, 79, ~~88~~

7 9 24 41 48 54
65 65 79 88



Postorder:

24
54 24
65 54 24
48 65 54 24
9 48 65 54 24
9 48 65 65 54 24
9 48 65 79 65 54 24
7 9 48 65 79 65 54 24
7 9 48 65 88 79 65 54 24.

∴ Postorder

| 7 9 48 65 88 79 65 54 24 |

Here are your random numbers:

24	54	65	48	9
65	79	7	88	41

Timestamp: 2020-11-30 16:58:14 UTC

Again! Go Back

Note: The numbers are generated left to right, i.e., across columns.

Follow us: Twitter | Facebook
Terms and Conditions
About Us