Question 1 [20 Marks]

1. Consider the following (partially completed) code snippet for Breath First Traversal (or Level Order Traversal) of a tree and answer the following questions. [7 marks]

```
01: void printLevelOrder(Node* root)
02: {
03:     if (_____)
04:         return;
05:     queue<Node*> q;
06:     q.enqueue(root);

07:     while (_____) {
08:         Node* node = q.front();
09:         cout << node->data << " ";

10:         _____

11:         if (_____)

12:             q.enqueue (_____);

13:         if (_____)

14:             q.enqueue (_____);
15:     }
16: }
17: int main()
18: {
    . . .
19:     printLevelOrder(root);
20:     return 0;
21: }
```

a. Select the correct option for line 03 of the `printLevelOrder` method above. [1 mark]

i. `root = = NULL`

ii. `q.empty( ) == true`

iii. `root != NULL`

iv. `q.empty() == false`

v. None of above

b. Select the correct option for line 07 of the `printLevelOrder` method above. [1 mark]

i. `q.front( ) == root`

ii. `q.empty( ) == true`

iii. `q.dequeue( ) == root`

iv. `root != NULL`

v. `q.empty() == false`

vi. None of above

c. Select the correct option for line 10 of the `printLevelOrder` method above. [1 mark]

i. `q.front( )`

ii. `q.dequeue ( )`

iii. `root = q.dequeue( )`

iv. `delete(node)`

v. `root = NULL`

vi. Not needed

d. Select the correct option for line 11 of the `printLevelOrder` method above. [1 mark]

i. `Node->left = q.front( )`

ii. `node->right != NULL`

iii. `node->left != NULL`

iv. `node != NULL`

v. `root == NULL`

vi. Not needed

e. Select the correct option for line 12 of the `printLevelOrder` method above. [1 mark]

i. `Node->left`

ii. `node->right`

iii. `node`

iv. `root`

v. None of above

f. Select the correct option for line 13 of the `printLevelOrder` method above. [1 mark]

i. `Node->left = q.front( )`

ii. `node->right != NULL`

iii. `node->left != NULL`

iv. `node != NULL`

v. `root == NULL`

vi. Not needed

g. Select the correct option for line 14 of the `printLevelOrder` method above. [1 mark]

i. `Node->left`

ii. `node->right`

iii. `node`

iv. `root`

v. None of above

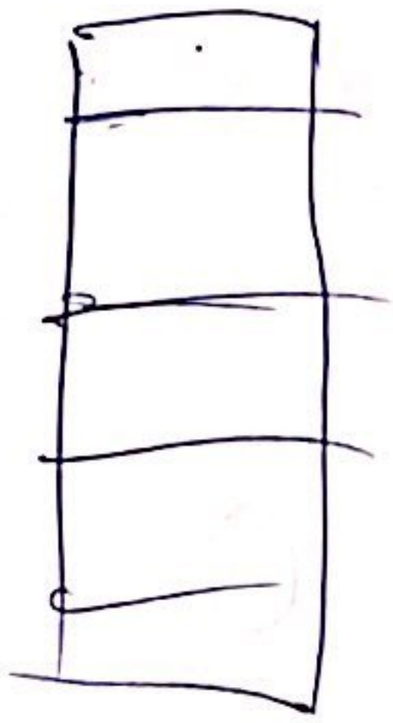
Rough Work (No marks will be awarded without properly understandable rough work):

NO rough work was needed for this question

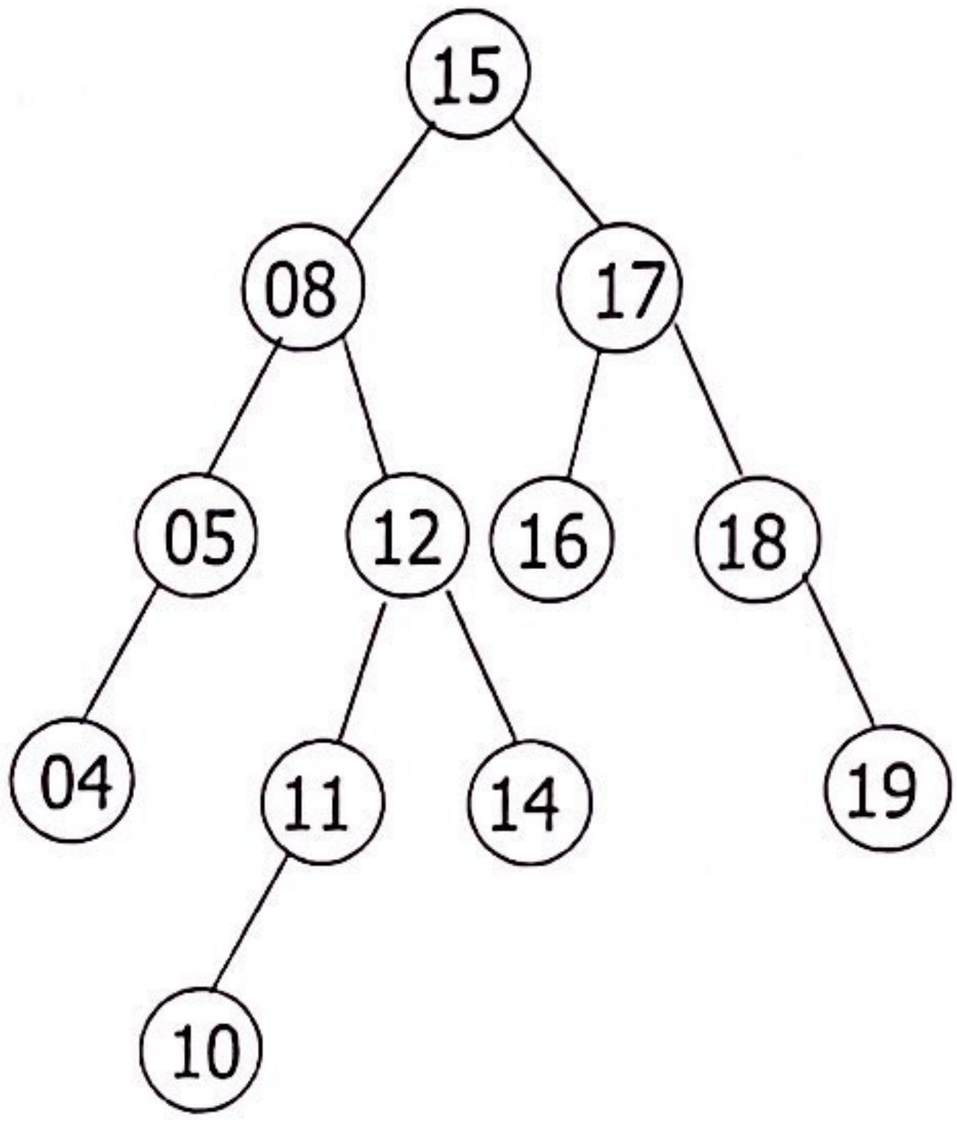
2. Consider the following code snippet and answer the following questions. [5 marks]

```
void Function1 (node* ptr)
{
    int num = Function3(ptr);
    for (int i = 1; i <= num; i++)
        Function2(ptr, i);
}
void Function2 (node* ntr, int var)
{
    if (ntr == NULL)
        return;
    if (var == 1)
        cout << ntr->data << " , ";
    else if (var > 1) {
        Function2(ntr->left, var - 1);
        Function2(ntr->right, var - 1);
    }
}
int Function3(node* node)
{
    if (node == NULL)
        return 0;
    else {
        int lvar = Function3 (node->left);
        int rvar = Function3 (node->right);

        /* use the larger one */
        if (lvar > rvar) {
            return (lvar + 1);
        }
        else {
            return (rvar + 1);
        }
    }
}
int main()
{
    //create tree
    Function1(root);
    return 0;
}
```



a. Given the following tree, what will be the output of the above code snippet. No marks will be awarded without proper dry run indicating the status of the system stack. [3 marks]



i. 15 , 08 , 05 , 04 , 12 , 11 , 10 , 14 , 17 , 16 , 18 , 19

ii. 04 , 05 , 08 , 10 , 11 , 12 , 14 , 15 , 16 , 17 , 18 , 19

iii. 15 , 08 , 17 , 05 , 12 , 16 , 18 , 04 , 11 , 14 , 19 , 10

iv. 04 , 05 , 10 , 11 , 14 , 12 , 08 , 16 , 19 , 18 , 17 , 15

v. 15 , 17 , 08 , 18 , 16 , 12 , 05 , 19 , 14 , 11 , 04 , 10

vi. None of above

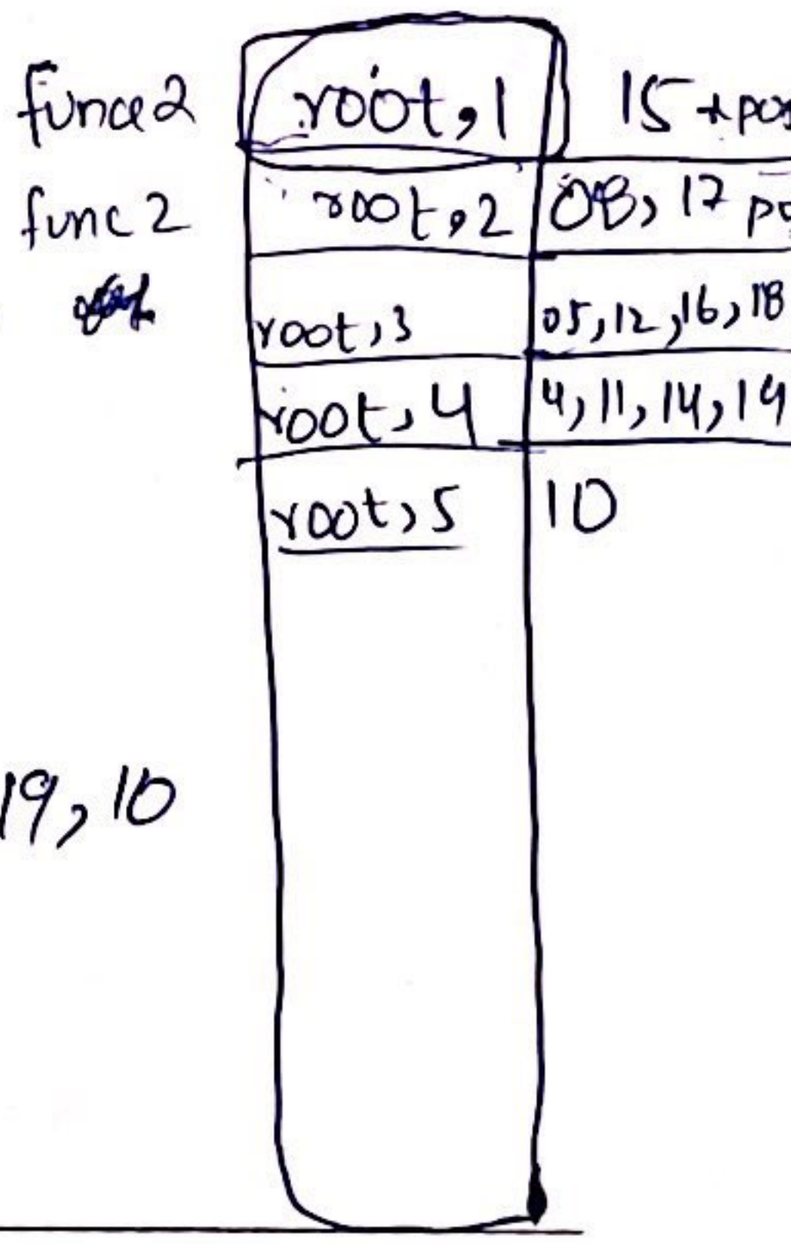
Dry run:

(Function 3) returns height

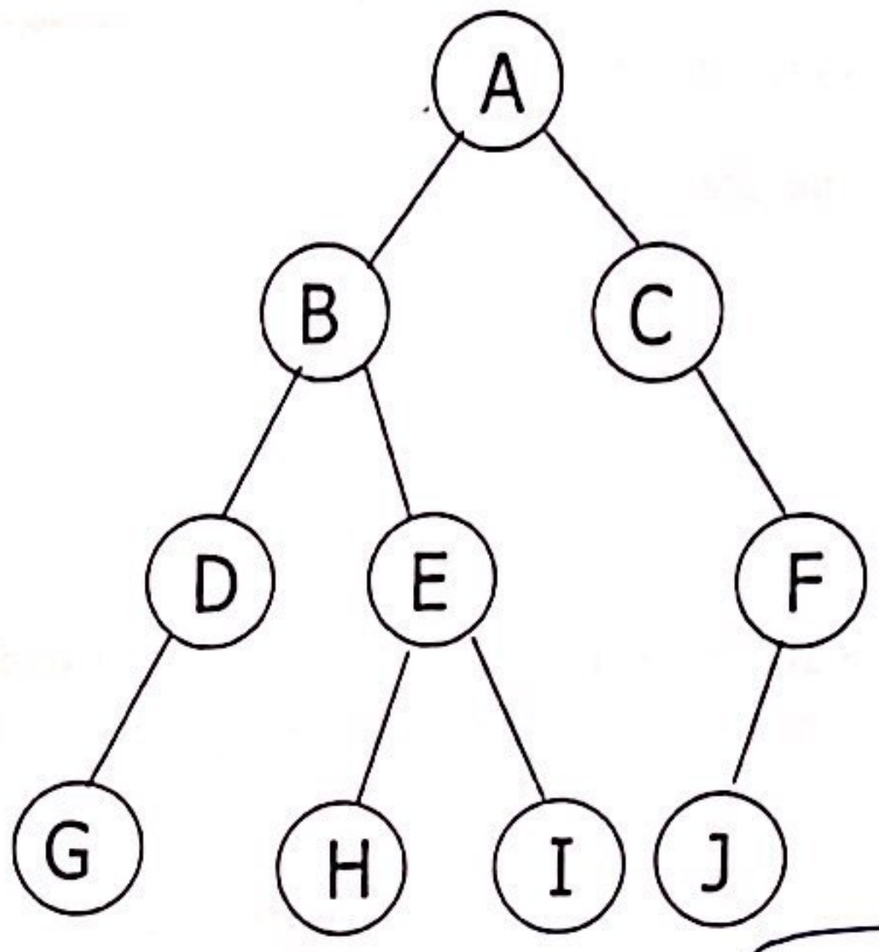
function 2 prints node on a level.

So on output window we get.

15, 08, 17, 05, 12, 16, 18, 04, 11, 14, 19, 10



b. Given the following tree, what will be the output of the above code snippet. No marks will be awarded without proper dry run indicating the status of the system stack. [2 marks]



i. A, B, D, G, E, H, I, C, F, J

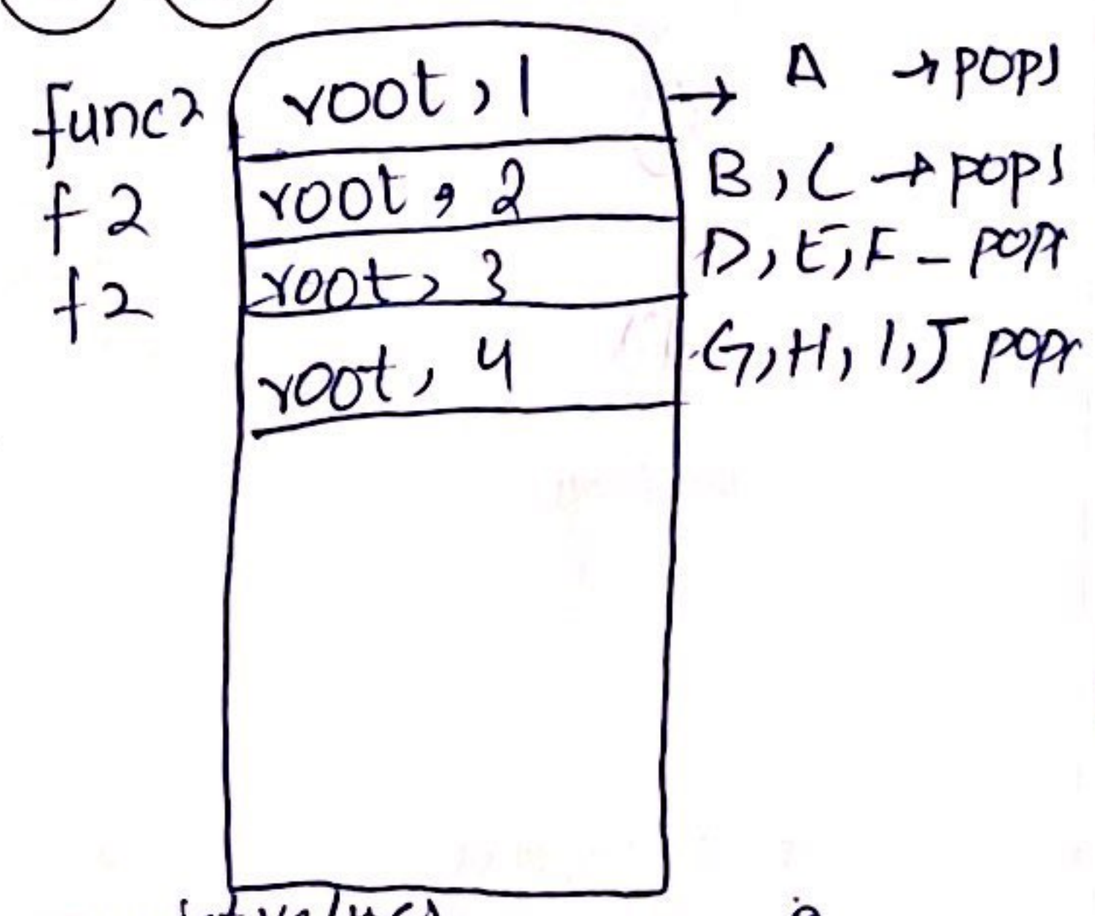
ii. G, D, H, I, E, B, J, F, C, A

iii. G, D, B, H, E, I, A, J, F, C

2 iv. A, B, C, D, E, F, G, H, I, J

v. A, C, B, F, E, D, J, I, H, G

vi. None of above



Dry run:

function 3 returns height.

func2 called in a loop will ~~return~~ print values

A, B, C, D, E, E, G, H, I, J.

3. Suppose we have numbers between 1 and 1000 in a binary search tree and want to search for the number 363. Which of the following sequence could not be the sequence of the node examined? No marks will be awarded without the proper justification. [1 mark]

a. 2, 252, 401, 398, 330, 344, 397, 363

b. 924, 220, 911, 244, 898, 258, 362, 363

c. 925, 202, 911, 240, 912, 245, 258, 363

d. 2, 399, 387, 219, 266, 382, 381, 278, 36

Justification by drawing the tree:

4. In full binary search tree every internal node has exactly two children. If there are 100 leaf nodes in the tree, how many internal nodes are there in the tree? No marks will be awarded without the proper justification. [1 mark]

a. 25

b. 49

c. 99

d. 101

Justification:

5. The height of a BST is given as h. The maximum no. of nodes possible in the tree is? No marks will be awarded without the proper justification. [1 mark]

a. $2^{h-1} - 1$

b. $2^{h+1} - 1$

c. h + 1

d. h -1

e. $2^h + 1$

f. $2^{h-1} + 1$

g. $2^h + 1$

Justification by drawing the tree:

As on each level there can be 2^h nodes

So $\sum_{i=1}^{h} 2^h = 2^{h+1} - 1$

6. The height of a BST is given as h. The minimum no. of nodes possible in the tree is? No marks will be awarded without the proper justification. [1 mark]

a. $2^{h-1} - 1$

b. $2^{h+1} - 1$

(c.) h + 1

d. h -1

e. $2^h + 1$

f. $2^{h-1} + 1$

g. $2^h + 1$

Justification by drawing the tree:

if the heigh is 0 then there can be 1 node.
" " " 1 then there can be 2 nodes.
" " " 2 then there can be 3 nodes
so nodes = h+1

7. In a full binary tree, every internal node has exactly two children. A full binary tree with 2n+1 = $2^h + 1$ nodes contains? No marks will be awarded without the proper justification. [1 mark]

(a.) n leaf node

b. n internal nodes

c. n-1 leaf nodes

d. n-1 internal nodes

Justification:

8. The run time for traversing all the nodes of a binary search tree with n nodes and printing them in an order is: No marks will be awarded without the proper justification. [1 mark]

a. O(n log(n))

(b.) O(n)

c. O(√n)

d. O(log(n))

Justification:

As there are n elements and each element is visited once so O(n).

9. If n elements are sorted in a binary search tree. What would be the asymptotic complexity to search a key in the tree? No marks will be awarded without the proper justification. [1 mark]

a. O(1)

b. O(log n)

c. O(n)

d. O(nlogn)

Justification:

To search we only need to go down until height. in worst case. h ≥ log n.
The worst case is that tree takes the shape of linked list. So O(n).

10. If n elements are sorted in a balanced binary search tree. What would be the asymptotic complexity to search a key in the tree? No marks will be awarded without the proper justification. [1 mark]

a. O(1)

b. O(logn)

c. O(n)

d. O(nlogn)

Justification

In a balanced binary tree we only go down till the height ⇒ h ≥ log n

Question 2 [20 Marks]

a. Consider the following code of an array-based stack class and main to answer the following questions.

```
class Stack
 {
private:
char *stackArray;
int stackSize;
int top;

public:
Stack(int size)
{
stackArray = new char[size];
stackSize = size;
top = -1;
}

void push(char val)
{
if (isFull())
cout << "The stack is full.\n";
else
{
top++;
stackArray[top] = val;
}
}

void pop(char &val)
{
if (isEmpty())
 cout << "The stack is empty.\n";
else
{
val = stackArray[top];
top--;
}
}

bool isFull()
{
return (top == stackSize - 1);

}

bool isEmpty()
{
return (top == -1);
}
```

```
void makeNull()
{
top = -1;
}
};
```

```
1. void main()
2. {
3. double num;
4. double num2;
5. char ch;
6. Stack stack(10);
7.
8. ifstream infile;
9. infile.open("data.txt");
10. if (!infile)
11. {
12. cout << "The input file does not exist." << endl;
13. return;
14. }
15. //reads file line by line and puts the first value in num and
16. //second in ch
17. infile >> num >> ch;
18.
19. num2 = num;
20. //loop till number of lines in file
21. while (infile) //Loop 1
22. {.
23. if (num > num2)
24. {
25. stack.makeNull();
26. if (!stack.isFull()) {
27. stack.push(ch); }
28. num2 = num;
29. }
30. else if (num == num2) {
31. if (!stack.isFull()){
32. stack.push(ch);
33. }
34. else
35. {
36. cout << "Stack overflows "<< endl;
37. return;
38. }
39. }
40. //reads next file line and puts the first
41. //value in num and second in ch
42. infile >> num >> ch;
43. }
44.
45. cout << num2 << endl;
```

```
46. while (!stack.isEmpty()) //Loop 2
47. {
48. char v;
49. stack.pop(v);
50. cout << v << " "<< num2 << endl;
51. }
52. cout << endl;
53. return;
54. }
```

data.txt input file is shown below :

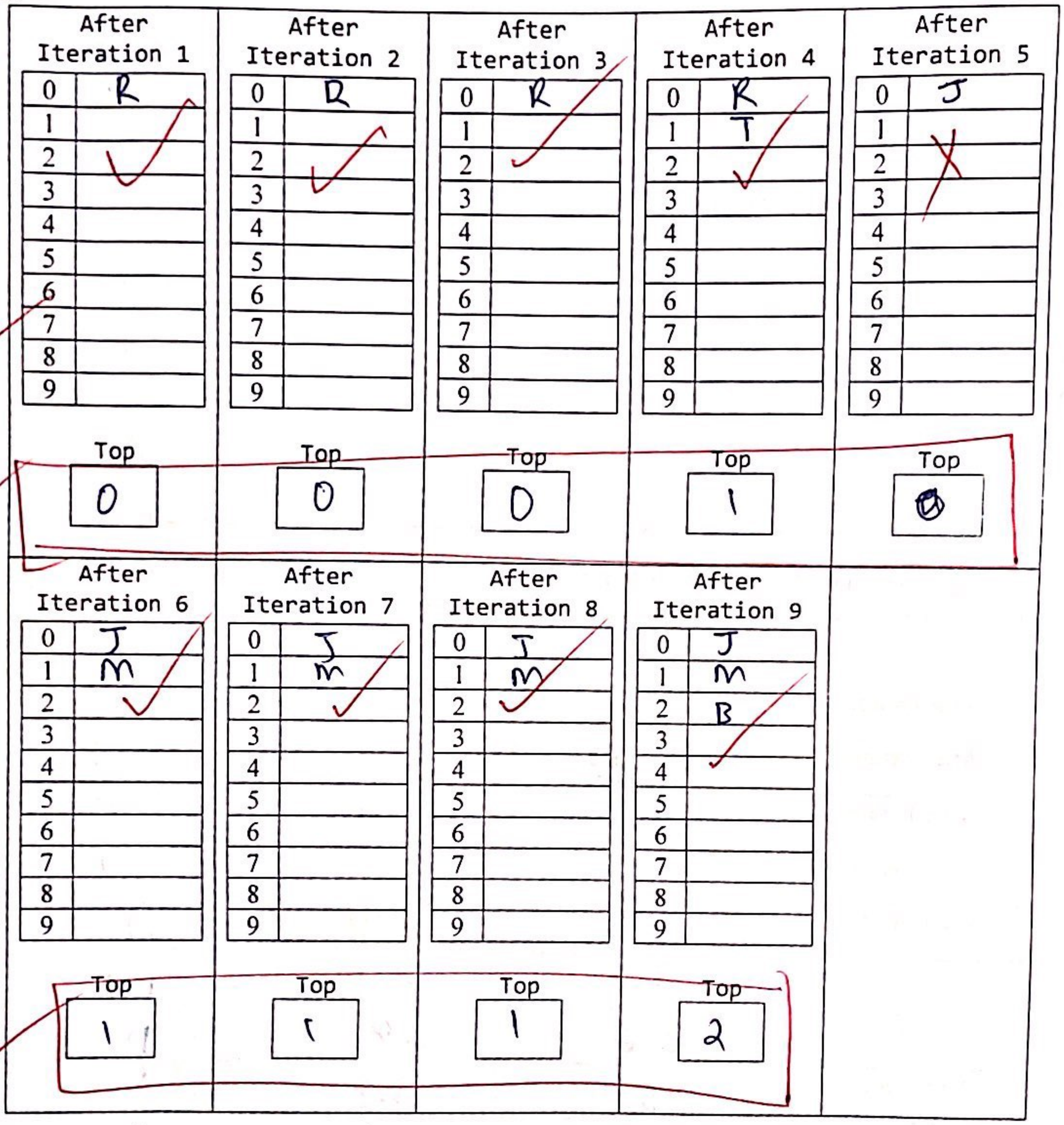
```
3.4 R
3.2 K
2.5 C
3.4 T
3.8 J
3.8 M
3.6 P
3.5 D
3.8 B
```

i. For the given data.txt file, what will be the values in the stack and what will be the values of num, num2 and ch after each iteration of Loop 1 of main(). [9 marks]

Iteration number	num	num2	ch
After Iteration 1	3.2	3.4	K
After Iteration 2	2.5	3.4	C
After Iteration 3	3.4	3.4	T
After Iteration 4	3.8	~~3.4~~ 3.8	J
After Iteration 5	3.8	3.8	M
After Iteration 6	3.6	3.8	P
After Iteration 7	3.5	3.8	D
After Iteration 8	3.8	3.8	B
After Iteration 9			

4.5

stack:

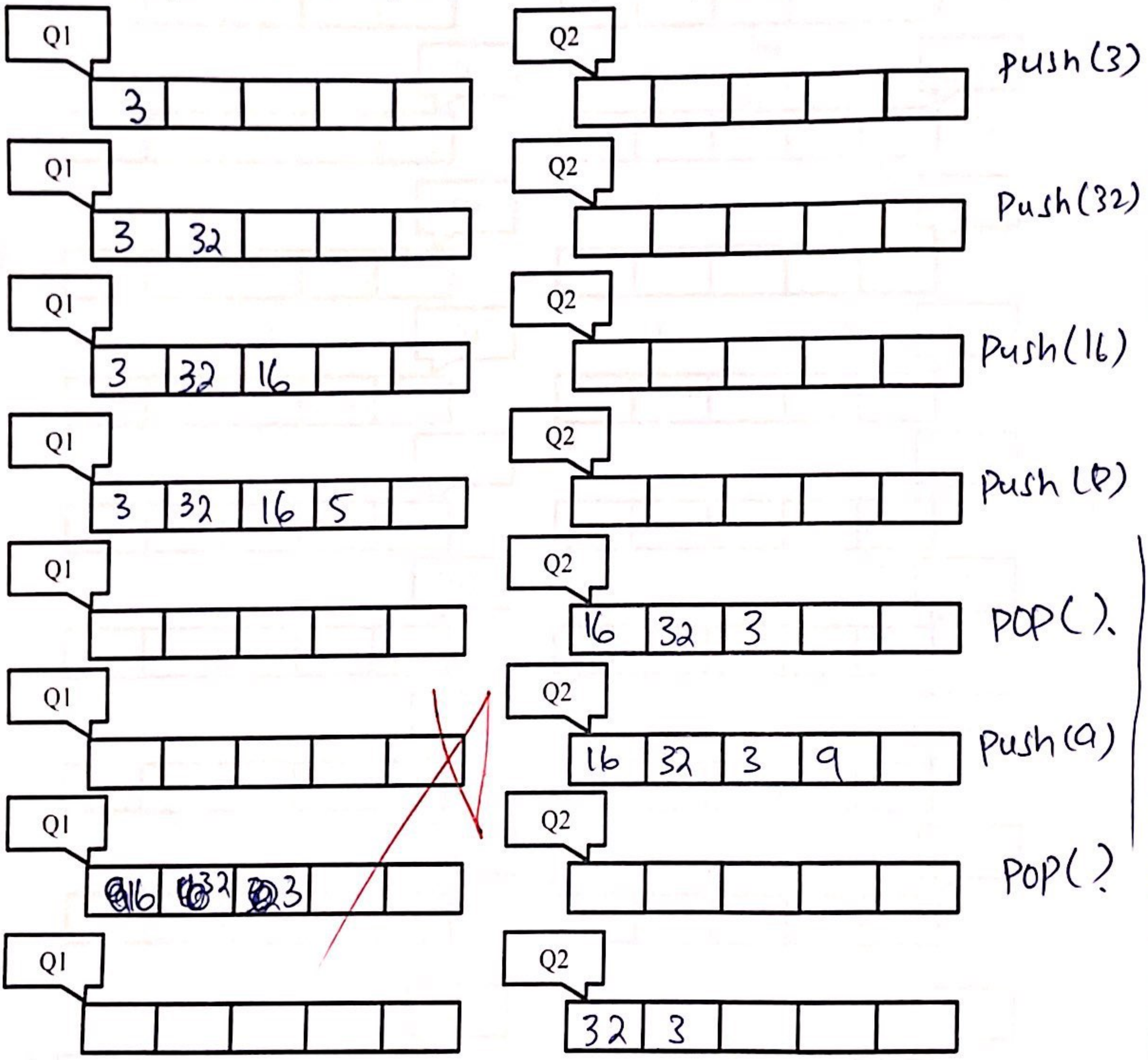


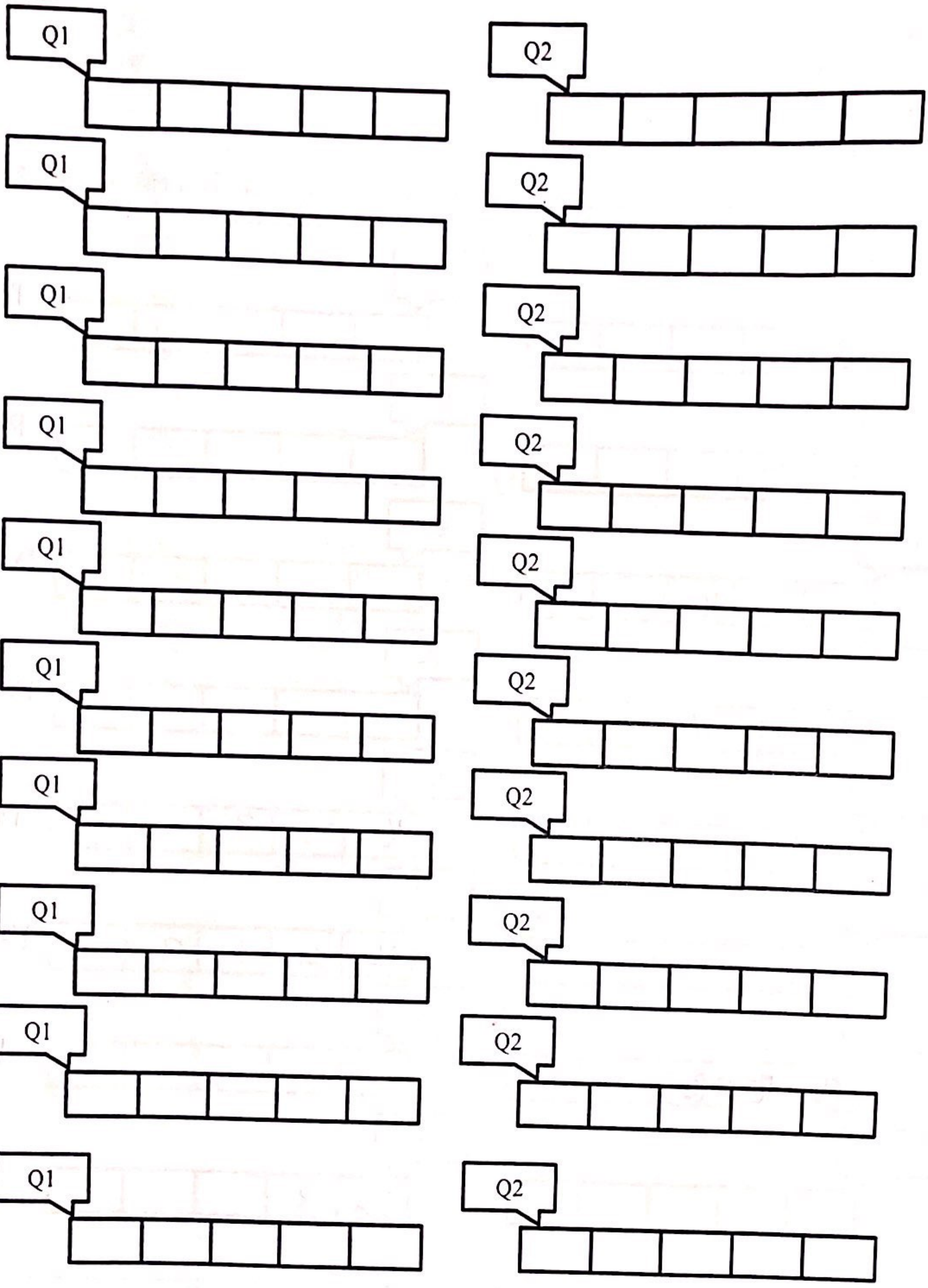
ii. What is the output of Loop 2 of main()? (3)

b. Following data must be processed through a stack in the given sequence

- Push(3)
- Push(32)
- Push(16)
- Push(5)
- Pop()
- Push(9)
- Pop()
- pop()

However, you only have two queues available (each of size=5) and no stack. Your task is to use these two queues to implement stack. Show your queues at each push and pop attempt. (8 marks)





NOTE: if you still need extra queues to show your working, you can extend your working on rough sheet at the end. Use same representations of Queue 1 and 2

National University of Computer and Emerging Sciences

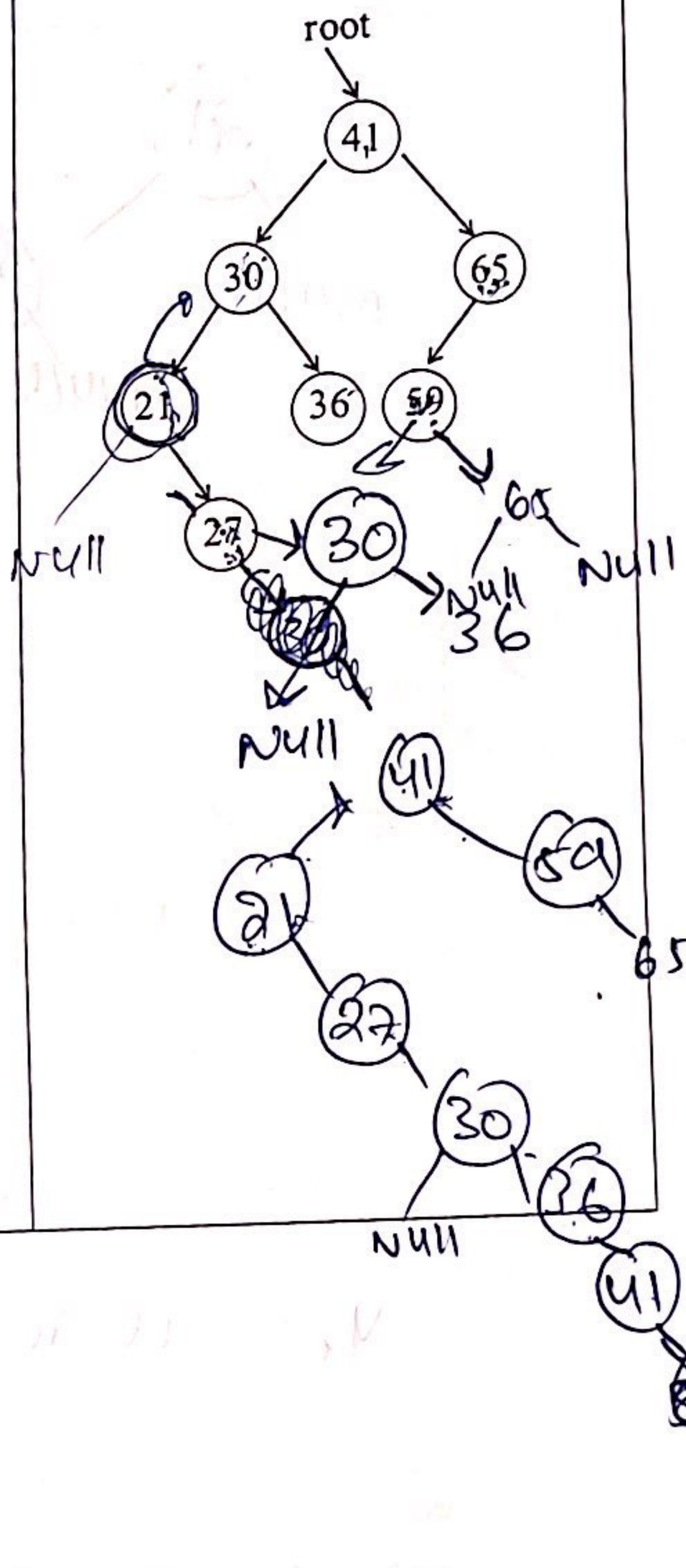
Question 3 [20 Marks]

Given a code of BST.

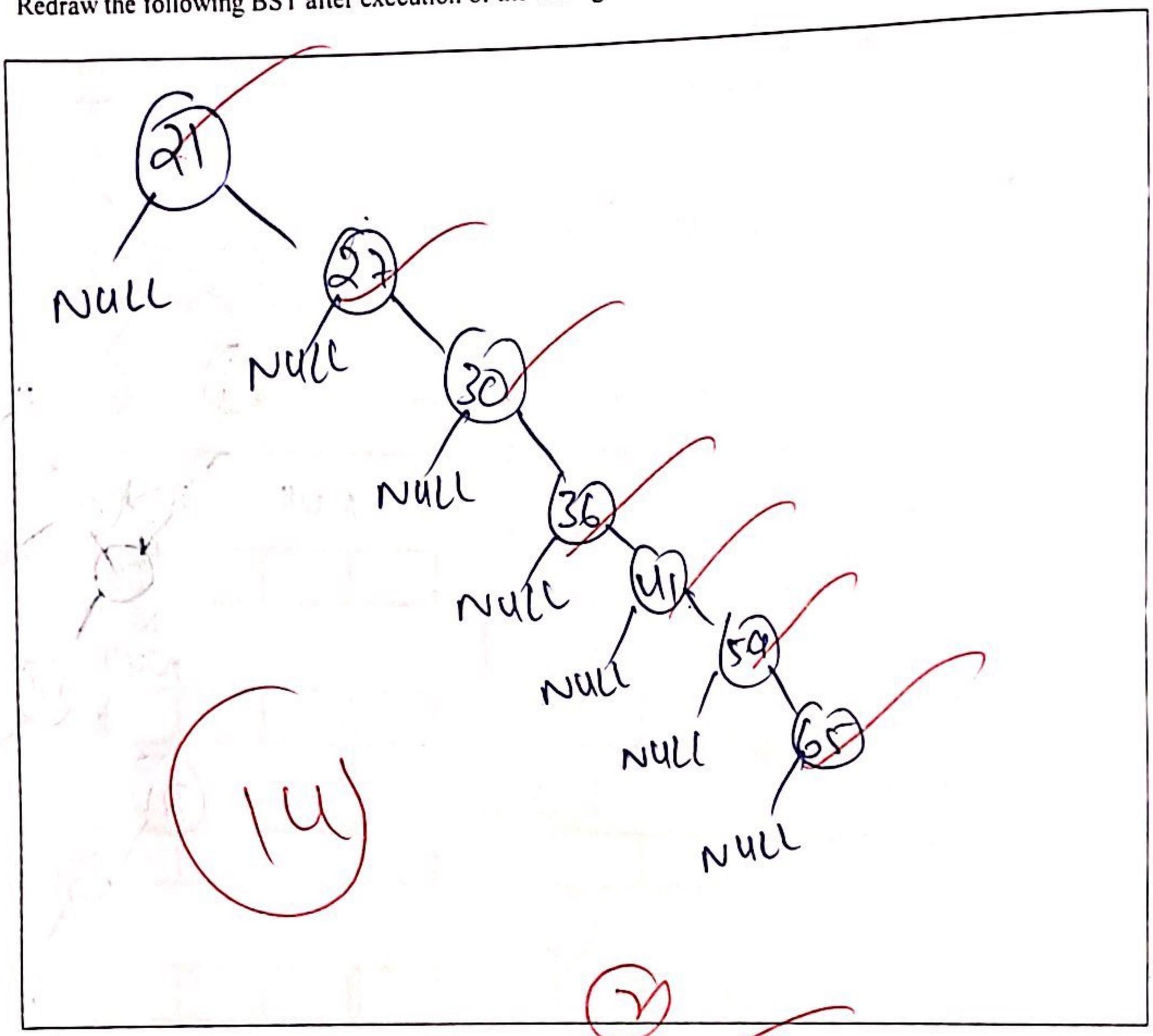
```
void UpdateTree() {
    if (root != NULL) {
        node* pre = NULL;
        root = UpdateTree(root, pre);
    }
}

node* UpdateTree(node* curr, node* pre) {
    if (curr->left == NULL && curr->right == NULL)
        return curr;

    node* r1 = NULL;
    node* r2 = NULL;
    if (curr->left != NULL)
        r1 = UpdateTree(curr->left, curr);
    if (curr->right != NULL)
        r2 = UpdateTree(curr->right, curr);
    if (r1 == NULL)
        return curr;
    if (r1->left != NULL)
    {
        r1->left->right = r1;
        r1 = r1->left->right;
        r1->left = NULL;
    }
    node* tmp = r1;
    while (tmp->right != NULL)
        tmp = tmp->right;
    tmp->right = curr;
    curr->right = r2;
    curr->left = NULL;
    return r1;
}
```



1. Redraw the following BST after execution of the code given below. [14 Marks]



2. Is the tree still BST? [2 Mark]

Yes it is still a BST.

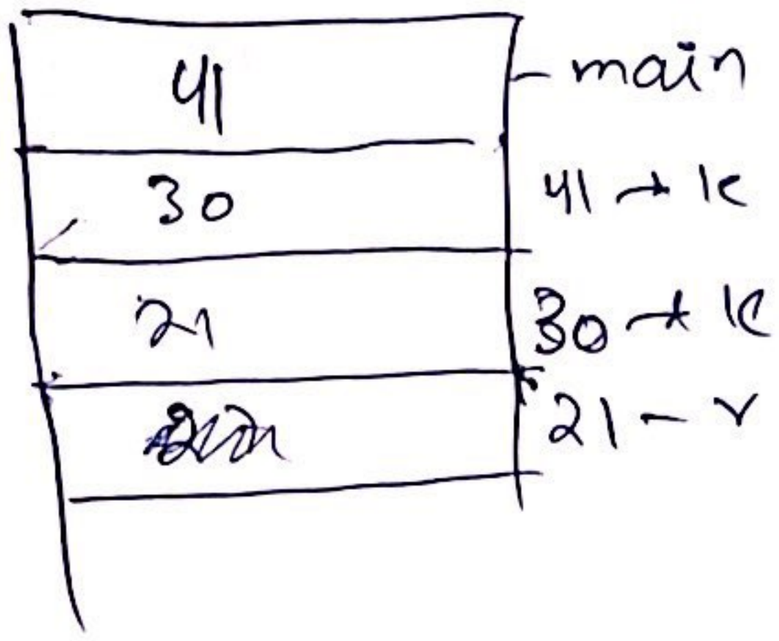
3. If there are N number of nodes, what will be the worst time complexity to search an item in updated tree? [2 Mark]

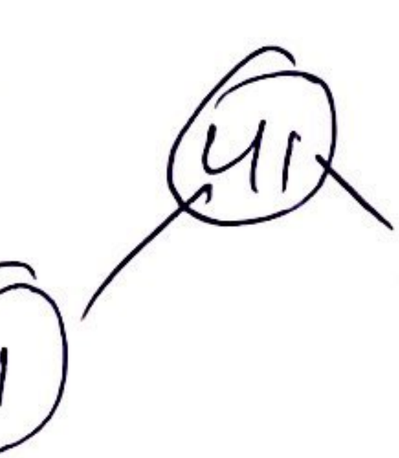
Worst complexity = O(N).

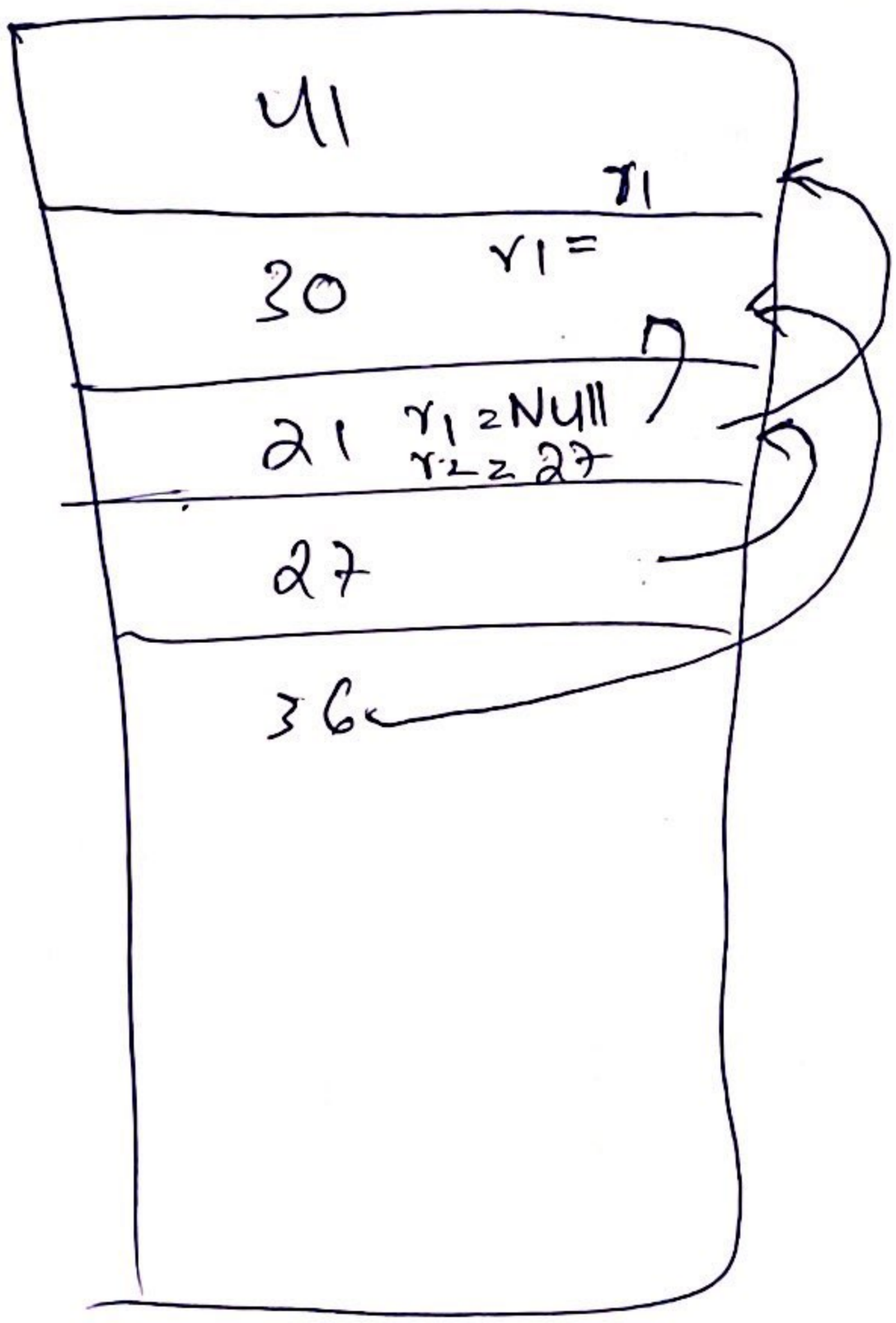
4. What is the worst time complexity of the above code? Justify your answer. [2 Mark]

The worst time complexity would be O(N)
We visit all N nodes. But some nodes are worst

Rough Work


41
30
21
main
41 → ic
30 → ic
21 → r


21
27


41
r1
r1 =
30
21
r1 = Null
r2 = 27
27
36


41
r1 = 21
30

41
21
27
30
NULL
36