Question 1 [25 Marks]

a) Provide a worst-case asymptotic time complexity of the following algorithms by using a suitable asymptotic notation considering the nearest function. Assume that there are no errors/ bugs in the algorithms. Show the meaningful work behind your answer. Marks will be deducted for direct answers. [5+5 marks]

Code	Time Complexity

`#include <iostream>`
`void complexFunction (int n) {`
`for (int i = 1; i <= n; ++i) {`
`for (int j = 1; j <= n; ++j) {`
`for (int k = 1; k <= n; k *= 2) {`
`std::cout << i * j << " ";`
`}}}}`
`int main() {`
`int n = 10;`
`complexFunction(n);`
`return 0; }`
	N^2 logn

`#include <iostream>`
`void complexFunction (int n) {`
`int i = 1;`
`while (i <= n) {`
`for (int j = 1; j <= n; ++j) {`
`for (int k = 1; k <= n; ++k) {`
`if (i % 2 == 0) {`
`std::cout << i * j * k << " ";`
`} else {`
`std::cout << -(i * j * k) << " ";`
`}`
`}`
`}`
`i++;`
`}`
`}`
`int main() {`
`int n = 5;`
`complexFunction(n);`
`return 0;`
`}`
	N^3

b) Please consider the following C++ code [3+2 marks]

```
#include <iostream>
using namespace std;

void magicFunction(int nums[], int n)
    int result[n];

    int j = 0;
    for (int i = 0; i < n ; i++)
        if (nums[i] >= 0 )
            result[j++] = nums[i];
    if (j == n || j == 0)
        return;

    for (int i = 0 ; i < n ; i++)
        if (nums[i] < 0)
            result[j++] = nums[i];

    for (int i = 0; i <= sizeof(result) / sizeof(result[0]) ; i++) {
        nums[i] = result[i];
    }
}
int main()
{
    int nums[] = {0, 9, -7, 2, -12, 11, -20};
    int n = sizeof(nums)/sizeof(nums[0]);
    cout << "Original array: ";
    for (int i=0; i < n; i++)
    cout << nums[i] <<" ";
     magicFunction(nums, n);

    cout<<"\nArray elements after magic function: ";
      for (int i=0; i < n; i++)
      cout << nums[i] <<" ";
        return 0;
}
```

i. What is the output of the above code? 3 marks

```
Original array: 0 9 -7 2 -12 11 -20
Array elements after magic function: 0 9 2 11 -7 -12 -20
```

ii. What is the purpose of the magic function? 2 marks.

Write a C++ program to move all negative elements of an array of integers to the end of the array. This is done without changing the order of the positive and negative elements of the array.

c) Please consider the following C++ code [3+2 marks]

```
#include<iostream>
using namespace std;
void magic_function_2(int nums[], int arr_size)
  {
   int i, n1, n2;

    /* There should be atleast two elements */
    if (arr_size < 2)
    {
        cout<< " Invalid Input ";
        return;
    }

    n1 = n2 = INT_MIN;
    for (i = 0; i < arr_size ; i ++)
    {

        if (nums[i] > n1)
        {
            n2 = n1;
            n1 = nums[i];
        }
        else if (nums[i] > n2 && nums[i] != n1)
        {
            n2 = nums[i];
        }
    }
    if (n2 == INT_MIN)
     {
        cout<< "No suitable output";
     }
    else
     {
        cout<<  "\n  magic_function_2 result is: " <<n2;
     }
}

int main()
{
    int nums[] = {7, 12, 9, 15, 19, 32, 56, 70};
    int               n               = sizeof(nums)/sizeof(nums[0]);
    cout << "Original array: ";
    for (int i=0; i < n; i++)
    cout << nums[i] <<" ";
    magic_function_2(nums, n);
    return 0;
}
```

i. What is the output of the above code? 3 marks

Original array: 7 12 9 15 19 32 56 70
magic_function_2 result is: 56

ii. What is the purpose of the magic function_2? 2 marks.

Write a C++ program to find the second largest element in an array of integers.

d) A double subscripted array declared as int arr [3] [5]. Please answer the following questions [1+4]

i. What is the total number of elements in the arr?

A double subscripted array int arr[3][5] has a total of 3 * 5 = 15 elements.

ii. What is the memory address of the arr [1][3] if the base address is 1010 [in decimal number]. Use formula to find the memory address. Zero marks will be awarded for direct answers.

To calculate the memory address of arr[1][3], we need to consider the size of each element in the array. Assuming each int element occupies 4 bytes of memory, and the base address of the array is 1010, we can calculate the memory address as follows:

The address of arr[1][3] can be calculated as:
Base address + (row index * number of columns + column index) * size of each element

So, the memory address of arr[1][3] would be:
1010 + (1 * 5 + 3) * 4 = 1010 + (8) * 4 = 1010 + 32 = 1042 [in decimal number].

Question 2:
Stacks and Queue : 3 marks

a) Implement a program that checks if a given string is a palindrome. You need to use the existing Stack and Queue Data Sturcture to determine if the string is a palindrome. Write a c++ code. (3 marks)

```
bool isPalindrome(string str) {
   stack<char> stk;
   queue<char> que;

_____________________________________
_
_
_

}
```

b) Describe why it is a bad idea to implement a linked list version of a queue which uses the head of the list as the rear of the queue. Give three reasons (3 Marks)

c) Implement the following stack ADT with the following properties.The push operation checks if the array is full and increases the capacity of the array by creating a new array with twice the capacity and copying the elements to it. The pop operation searches for the maximum element in the array using linear search, swaps it with the last element, and decreases the size of the array. If the size becomes less than or equal to one-fourth of the capacity, the capacity of the array is decreased by creating a new array with half the capacity and copying the elements to it. The empty operation checks if the size of the array is zero or not. (3+3 +1)

```
class Stack {
private: int *arr; int size; int capacity;
 Stack() { size = 0; capacity = 1; arr = new int[capacity]; }
#include <iostream>

class Stack {
private:
        int* arr;
        int size;
        int capacity;

public:
        Stack() {
        size = 0;
        capacity = 1;
        arr = new int[capacity];
```

```
        }

        ~Stack() {
        delete[] arr;
        }

        bool isEmpty();
        void push(int value) ;
        void pop() ;

    };
```

Part a) Solution :

```
bool isPalindrome(string str) {
    stack<char> stk;
    queue<char> que;

    // Pushing characters from the first half into the stack
    for (int i = 0; i < str.length() / 2; i++) {
        stk.push(str[i]);
    }

    // Pushing characters from the second half into the queue
    for (int i = str.length() - 1; i >= str.length() / 2; i--) {
        que.push(str[i]);
    }

    // Comparing characters from the stack and queue
    while (!stk.empty() && !que.empty()) {
        if (stk.top() != que.front()) {
            return false;  // Characters don't match, not a palindrome
        }
        stk.pop();
        que.pop();
    }

    // Stack and queue should be empty if the string is a palindrome
    return (stk.empty() && que.empty());
}
```

Part b) Inefficient enqueue operation: With the head of the list as the rear of the queue, adding elements to the queue (enqueue) requires traversing the entire linked list to reach the end. This operation has a time complexity of O(n), where n is the number of elements in the queue. In

contrast, a typical queue implementation with a rear pointer (tail) allows enqueue operations in constant time O(1).

1. Inefficient dequeue operation: Removing elements from the front of the queue (dequeue) using a linked list implementation with the head as the rear also requires traversing the entire list to find the new front element. This operation also has a time complexity of O(n), making it inefficient compared to O(1) in a standard queue implementation.
2. Increased memory overhead: In the linked list version with the head as the rear, each node in the list requires additional memory overhead for storing the next pointer. This extra overhead per node increases memory consumption compared to a typical queue implementation.
3. Lack of clarity and confusion: Using the head of the linked list as the rear of the queue deviates from the standard queue implementation, which may lead to confusion and difficulties in understanding and maintaining the code. It goes against the common convention and can make the code less readable and harder to debug.
4. Violation of the queue abstraction: The head of the linked list represents the beginning of the list, not the rear of the queue. By using the head as the rear, the implementation violates the conceptual abstraction of a queue, making the code less intuitive and harder to reason about.

Part d) #include <iostream>

```
class Stack {
private:
        int* arr;
        int size;
        int capacity;

public:
        Stack() {
        size = 0;
        capacity = 1;
        arr = new int[capacity];
        }

        ~Stack() {
        delete[] arr;
        }

        bool isEmpty() {
        return size == 0;
        }

        void push(int value) {
        if (size == capacity) {
```

```
// Create a new array with twice the capacity
capacity *= 2;
int* newArr = new int[capacity];

// Copy elements from the old array to the new array
for (int i = 0; i < size; i++) {
        newArr[i] = arr[i];
}

// Delete the old array
delete[] arr;

// Point the arr pointer to the new array
arr = newArr;
}

arr[size] = value;
size++;
}

void pop() {
if (isEmpty()) {
std::cout << "Stack is empty. Cannot pop." << std::endl;
return;
}

// Find the maximum element and its index
int maxElement = arr[0];
int maxIndex = 0;

for (int i = 1; i < size; i++) {
if (arr[i] > maxElement) {
        maxElement = arr[i];
        maxIndex = i;
}
}

// Swap the maximum element with the last element
int temp = arr[maxIndex];
arr[maxIndex] = arr[size - 1];
arr[size - 1] = temp;

// Decrease the size of the array
size--;
```

```
        // Check if the size is less than or equal to one-fourth of the capacity
        if (size <= capacity / 4) {
        // Create a new array with half the capacity
        capacity /= 2;
        int* newArr = new int[capacity];

        // Copy elements from the old array to the new array
        for (int i = 0; i < size; i++) {
                newArr[i] = arr[i];
        }

        // Delete the old array
        delete[] arr;

        // Point the arr pointer to the new array
        arr = newArr;
        }
        }
};

int main() {
        Stack stack;
        std::cout << "Is the stack empty? " << (stack.isEmpty() ? "Yes" : "No") << std::endl;

        stack.push(10);
        stack.push(5);
        stack.push(15);
        stack.push(8);

        std::cout << "Is the stack empty? " << (stack.isEmpty() ? "Yes" : "No") << std::endl;

        stack.pop();
        stack.pop();

        std::cout << "Is the stack empty? " << (stack.isEmpty() ? "Yes" : "No") << std::endl;

        return 0;
}
```

Question 3:

Complete the following code that rotate the singly linked list counter-clockwise by k nodes. Where k is a given positive integer. For example, if the given linked list is:

10->20->30->40->50->60->NULL and k is 4,

the list should be modified to 50->60->10->20->30->40->NULL.

```
01:   struct Node{
02:         int data;
03:         Node* next;
04:   };
05:   void rotate(Node*&head, int k)
06:   {
07:         _____________________________________;
08:         Node* current=head;
09:         while(_____________________________)
10:               current = current -> next;
11:          _____________________________;
12:         current = head;
13:         for(int count = 0; count < k - 1 ; count++)
14:                     current = current -> next;
15:         _____________________________________;
16:         current -> next = NULL;
17:   }
```

Line 07:

i. if(k == 0) return

ii. if(k <= 0 && head == NULL) return

iii. if(k <= 0 || head == NULL) return

iv. None of these

Line 09:

i. current -> next != NULL

ii. current

iii. current -> next == NULL

iv. None of these

Line 11:

i. current -> next = head

ii. head = current -> next

iii. There is no need of Line 11

iv. None of these

Line 15:

i. head = current

ii. current -> next = head

iii. head -> next = current -> next

iv. None of these

Space for rough work:

Part-B [5 marks]

Given the code fragment. [2mark]

```
struct NodeType{
        int data;
        NodeType* next;
}
        NodeType* p;
        NodeType* q;
        p = new NodeType;
        p->data = 12;
        p->next = NULL;
        q = new NodeType;
        q ->data = 5;
        q ->next = p;
```

Which of the following expressions has the value NULL?
a. p
b. q
c. q->next
d. q->next->next

The following code, which builds a linked list with the numbers 18 and 32 as its components, has amissing statement. What should the statement be? [2 marks]

```
struct NodeType{
        int data;
        NodeType* next;
}
        NodeType* p;
        NodeType* q;
        p = new NodeType;
        p-> data = 18;
        q = new NodeType;
        q-> data = 32;
                        < - - missing statement
        q->next = NULL;
```

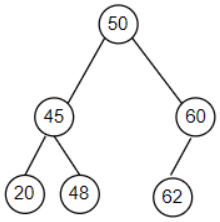
a. p = q
b. p->next =new NodeType;
c. p->next = q->next;
d. p->next = q;

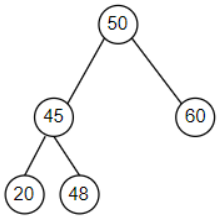
Given the code fragment, what is the data type of the expression ptr->link->link->volume. [1 mark]

```
struct ListNode{
        float volume;
        ListNode *link;
}
        ListNode *ptr;
```

a. ListNode
b. *ListNode
c. Expression is invalid
d. float

Find sec max





Case 1 Case 2

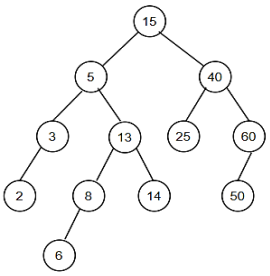
[25 marks] Consider the two cases of a given tree. You have to find the second largest value using **recursion**. For the case 1, the second largest is supposed to be 60, so the function must return 60. Write a function to find second largest value, considering both of the given cases

```
Node getSecondLargest(Node node)
{
Case 1:
if (node.right.right == null && node.rught.left != null) {

return node.right
}
-----------------------------------------------------------------------------------------------------
Case 2:
if (node.right != null &&   node.right.left == null &&  node.right.right == null) {
return node;
}
-----------------------------------------------------------------------------------------------------

  return getSecondLargest(node.right);
}
```

Given a tree



Write a **recursive** code to count the internal nodes of a tree and return the resultant count.

int countInternalNodes(Node* root)
{

```
int countNonleaf(struct Node* root)
{
                                                                  // Base cases.
    if (root == NULL || (root->left == NULL &&
                         root->right == NULL))
        return 0;


    return 1 + countNonleaf(root->left) +
               countNonleaf(root->right);      // If root is Not NULL
                                                      and its one of its

                                               // child is also not NULL

}
```

///5 mrks for basecase
///5 mrks for rest

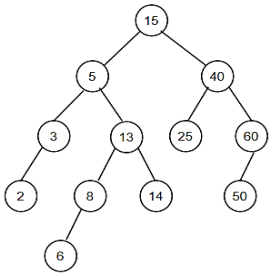
Sol 2

```
1. int internalnodes(struct node *newnode)
2.          {
3.              if(newnode != NULL)
4.              {
5.                  internalnodes(newnode->left);
6.           if((newnode->left != NULL) || (newnode->right != NULL))
7.                  {
8.                      count++;
9.                  }
10.                 internalnodes(newnode->right);
11.             }
12.             return count;
13.         }
```

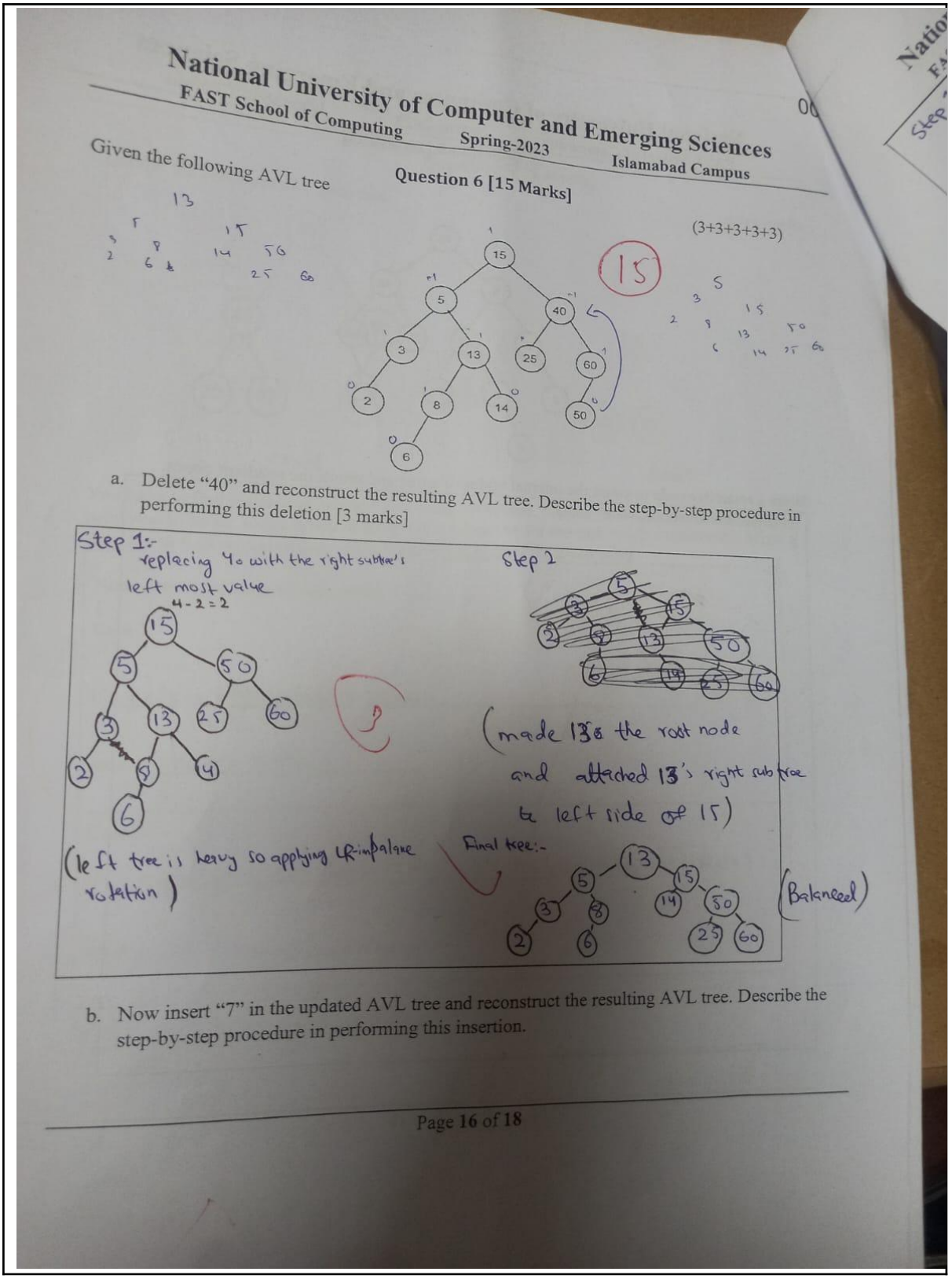
}

AVL [15 marks]

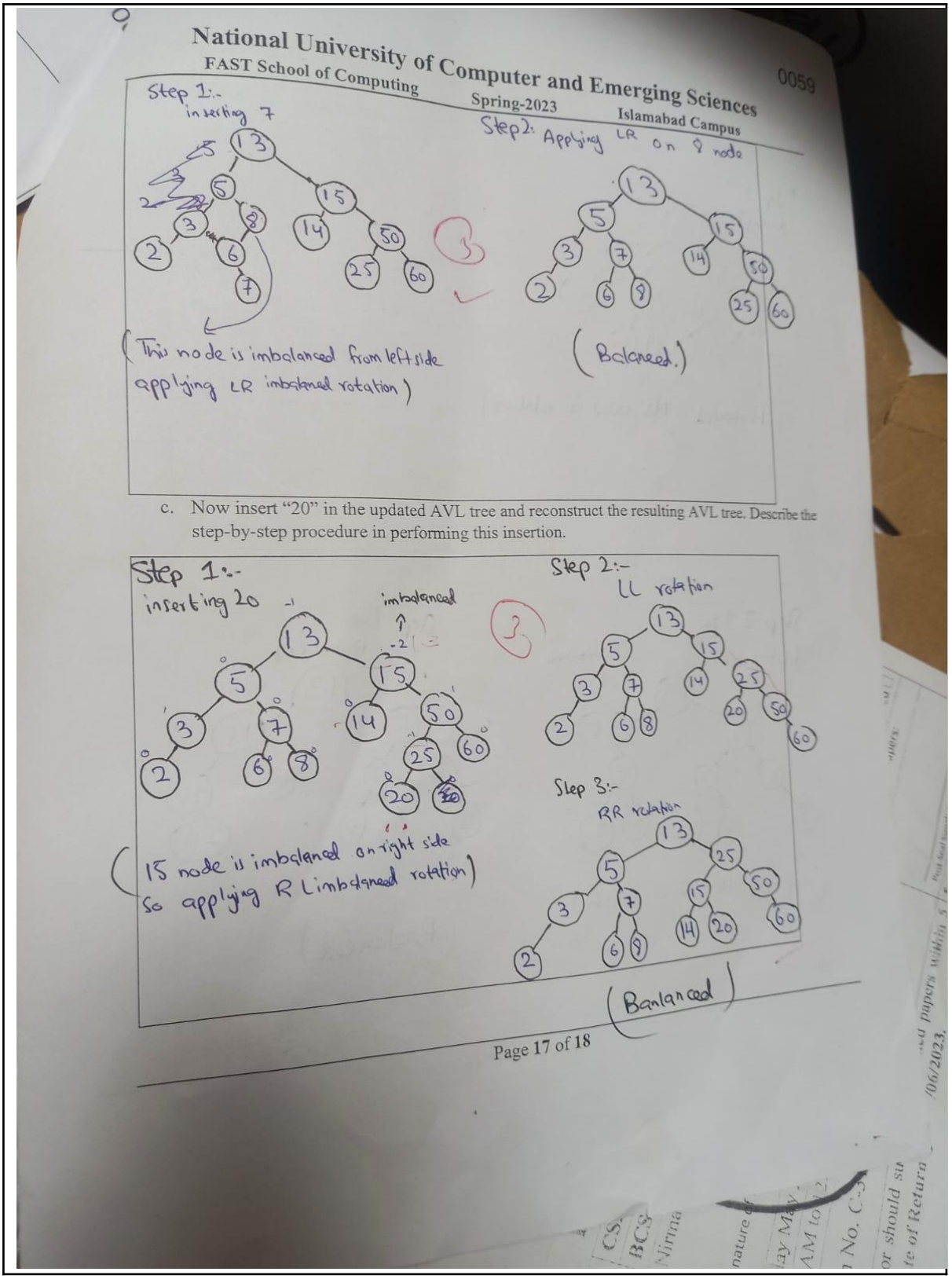
Given the following AVL tree



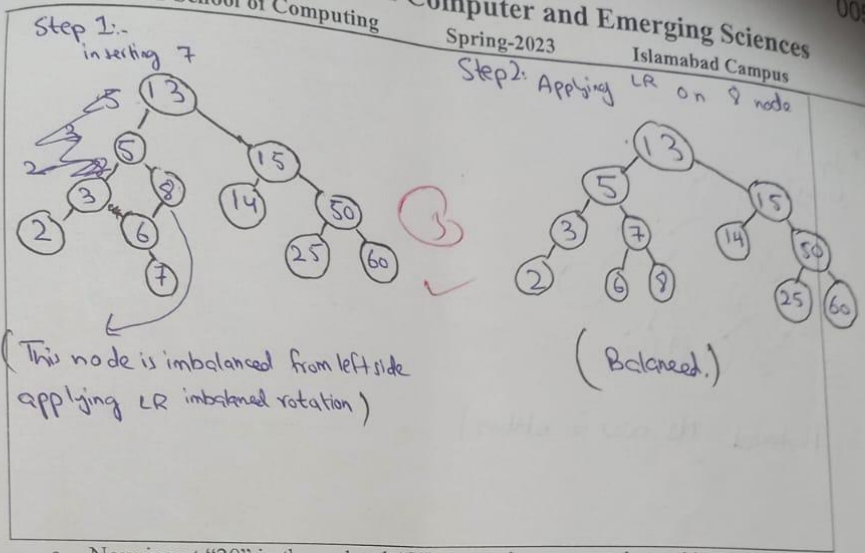
a. Delete “40” and reconstruct the resulting AVL tree. Describe the step-by-step procedure in performing this deletion [3 marks]



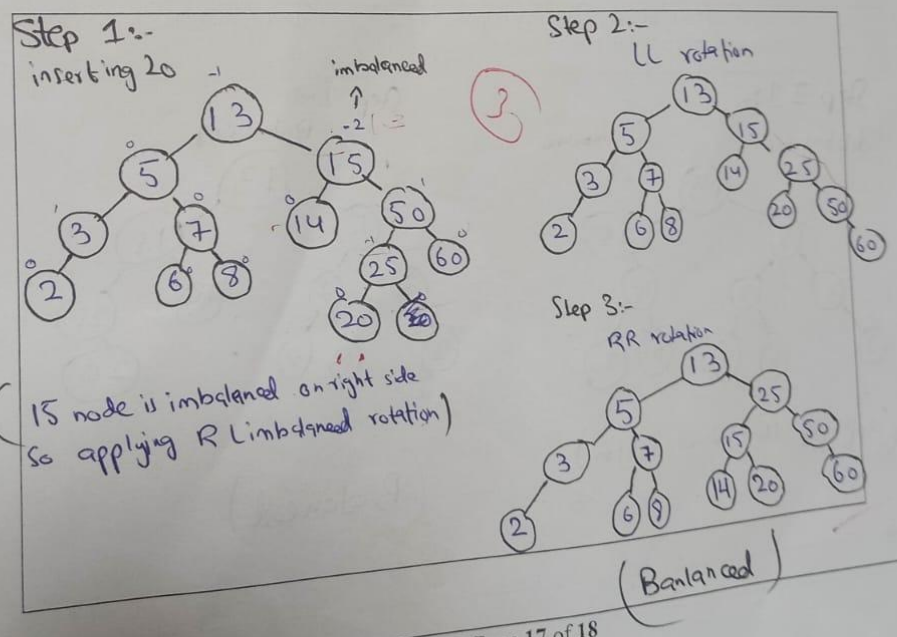
b. Now insert "7" in the updated AVL tree and reconstruct the resulting AVL tree. Describe the step-by-step procedure in performing this insertion. [3 Marks]



c. Now insert "20" in the updated AVL tree and reconstruct the resulting AVL tree. Describe the step-by-step procedure in performing this insertion. [3 Marks]

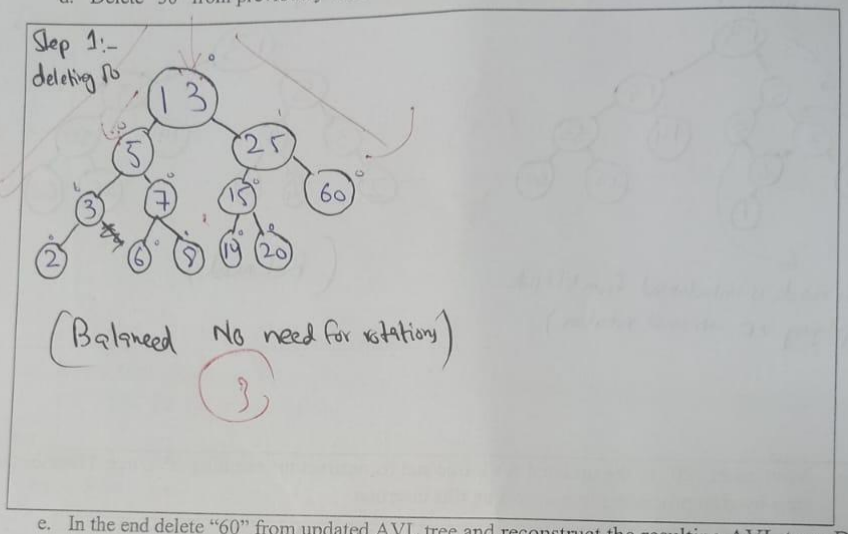


c. Now insert "20" in the updated AVL tree and reconstruct the resulting AVL tree. Describe the step-by-step procedure in performing this insertion.

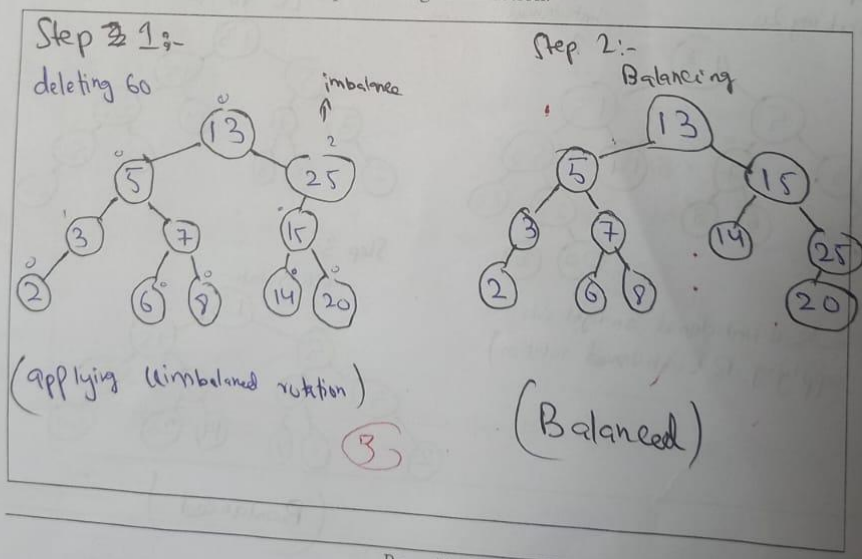


d. Delete "50" from previously constructed AVL Tree and reconstruct the resulting AVL tree. [3 Marks]

d. Delete "50" from previously constructed AVL Tree and reconstruct the resulting AVL tree.

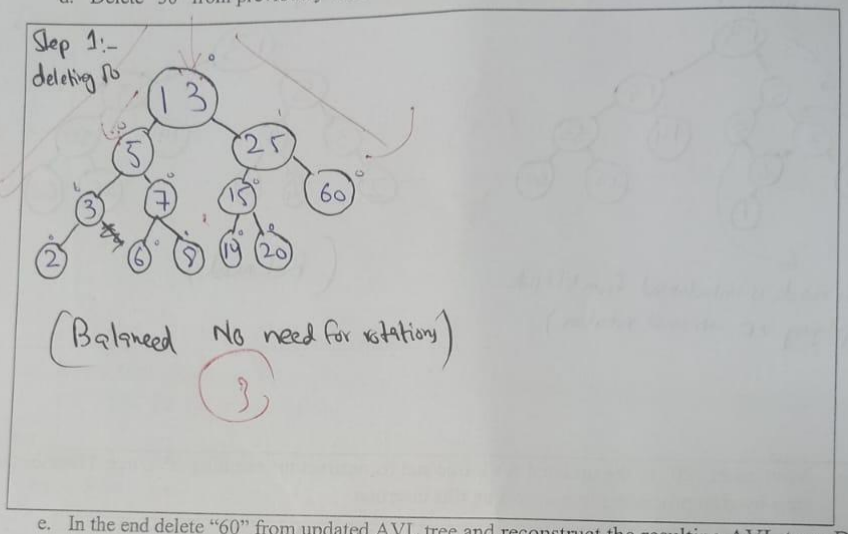


e. In the end delete "60" from updated AVL tree and reconstruct the resulting AVL tree. Describe the step-by-step procedure in performing this deletion.



e. In the end delete "60" from updated AVL tree and reconstruct the resulting AVL tree. Describe the step-by-step procedure in performing this deletion. [3 Marks]

d. Delete "50" from previously constructed AVL Tree and reconstruct the resulting AVL tree.



e. In the end delete "60" from updated AVL tree and reconstruct the resulting AVL tree. Describe the step-by-step procedure in performing this deletion.