**Rubric A**

| **SI** | **Category** | **Marks** |
| --- | --- | --- |
| 1 | Proper Method/Function Declaration | 1 |
| 2 | Correct base case list 1 | 2.5 |
| 3 | Correct base case list 2 | 2.5 |
| 4 | Recursive call | 2 |
| 5 | Correctly iterates towards convergence using List\_head.next | 2 |
| 6 | Correctly merge two lists (Sorting with if condition) | 3 |
| 8 | Return the correct list | 2 |
| **Total = 15** | | |

**Solution SET A**

| **Python** | **Java** |
| --- | --- |
| def merge\_sorted\_lists(l1, l2):  if l1 is None:  return l2  if l2 is None:  return l1  if l1.data < l2.data:  l1.next = merge\_sorted\_lists(l1.next, l2)  return l1  else:  l2.next = merge\_sorted\_lists(l1, l2.next)  return l2 | public class Main{  public static Node mergeSortedLists(Node l1, Node l2) {  if (l1 == null)  return l2;  if (l2 == null)  return l1;  if (l1.data < l2.data) {  l1.next = mergeSortedLists(l1.next, l2);  return l1;  } else {  l2.next = mergeSortedLists(l1, l2.next);  return l2;  }  } |

**Rubric B**

| **SI** | **Category** | **Marks** |
| --- | --- | --- |
| 1 | Proper Method/Function Declaration | 1 |
| 2 | Correct base case list 1 | 2.5 |
| 3 | Correct base case list 2 | 2.5 |
| 4 | Recursive call | 2 |
| 5 | Correctly iterates towards convergence using List\_head.next | 2 |
| 6 | Correctly merge two lists (Swap operation with temporary variable) | 3 |
| 8 | Return the correct list | 2 |
| **Total = 15** | | |

**Solution SET B**

| **Python** | **Java** |
| --- | --- |
| def merge\_alternate(l1, l2):  if l1 is None:  return l2  if l2 is None:  return l1  l1\_next = l1.next  l2\_next = l2.next  l1.next = l2  l2.next = merge\_alternate(l1\_next, l2\_next)  return l1 | public class Main{  public static Node mergeAlternate(Node l1, Node l2) {  if (l1 == null)  return l2;  if (l2 == null)  return l1;  Node l1Next = l1.next;  Node l2Next = l2.next;  l1.next = l2;  l2.next = mergeAlternate(l1Next, l2Next);  return l1;  }  } |

**Quiz-05**

**Solve:**

| Python | Java |
| --- | --- |
| def Odd **/** EvenSwap(root):  def helper(node, level):  if not node:  return  ***#Set A***  if level % 2 == 1:  if node.left and node.right and   node.left.val <node.right.val:  node.left, node.right =   node.right, node.left  ***#Set B***  if level % 2 == 0:  if node.left and node.right and   node.right.val > node.left.val:  node.left, node.right =   node.right, node.left    helper(node.left, level + 1)  helper(node.right, level + 1)   helper(root, 0)  return root | public static BTNode Odd/EvenSwap(BTNode root) {  helper(root, 0);  return root;  }  private static void helper(BTNode node, int level) {  if (node == null) return;  ***//Set A***  if (level % 2 == 1) {  if (node.left != null && node.right != null &&   node.left.val < node.right.val) {  BTNode temp = node.left;  node.left = node.right;  node.right = temp;  }  }  ***// Set B***  if (level % 2 == 0) {  if (node.left != null && node.right != null &&   node.left.val < node.right.val) {  BTNode temp = node.left;  node.left = node.right;  node.right = temp;  }  }  helper(node.left, level + 1);  helper(node.right, level + 1);  } |

**Rubric:**

| **Part** | **Marks** |
| --- | --- |
| Correct Parameters main method and helper | 2 |
| Correct Base Case Handling | 2 |
| Checking the Correct Level and Value Comparison | 2 |
| Checking the Existence of Children Nodes | 3 |
| Correctly swapping | 2 |
| Correct Recursive call | 2 |
| Calling helper and return root | 2 |
|  | 15 |