| Set A | |
| --- | --- |
| Python | Java |
| def sum\_k\_smallest(root, k):  if not root or k == 0:  return 0, k    left\_sum, k = sum\_k\_smallest(root.left, k)  if k == 0:  return left\_sum, k    k -= 1  current\_sum = left\_sum + root.val    if k == 0:  return current\_sum, k    right\_sum, k = sum\_k\_smallest(root.right, k)  return current\_sum + right\_sum, k  def smallest\_sum(root, k):  total\_sum, \_ = sum\_k\_smallest(root, k)  return total\_sum | public static int[] sumKSmallest(TreeNode root, int k) {  if (root == null || k == 0) {  return new int[]{0, k}; // {sum, remaining k}  }    int[] leftResult = sumKSmallest(root.left, k);  int leftSum = leftResult[0];  k = leftResult[1];    if (k == 0) {  return new int[]{leftSum, 0};  }    k--;  int currentSum = leftSum + root.val;    if (k == 0) {  return new int[]{currentSum, 0};  }    int[] rightResult = sumKSmallest(root.right, k);  int rightSum = rightResult[0];  k = rightResult[1];    return new int[]{currentSum + rightSum, k};  }  public static int smallestSum(TreeNode root, int k) {  return sumKSmallest(root, k)[0];  } |
| **Set B** | |
| **Replace + with \*** | |

| **Marking Rubric** | | |
| --- | --- | --- |
| **Criteria** | **Description** | **Marks** |
| **1. Correct recursive structure** | Proper use of recursion to traverse the BST without loops or iteration. | **3 marks** |
| **2. In-order traversal logic** | Traverses left → root → right correctly to ensure smallest elements come first. | **3 marks** |
| **3. Proper k management** | Correct decrementing and passing of k during recursion to stop at the right time. | **3 marks** |
| **4. Correct sum calculation** | Accurately adds the node values to compute the sum of the first k nodes. | **3 marks** |
| **5. No extra data structures used** | No arrays, lists, stacks, or other structures for traversal or storage (except tiny return arrays). | **1 mark** |
| **6. Edge cases handled (k == 0 or root == null)** | Correct behavior when k = 0 or tree is empty. | **2 marks** |