**Solve (Set#A)**

| **Python** | |
| --- | --- |
| def max\_path\_sum(root):  if root is None:  return 0  left\_sum = max\_path\_sum(root.left)  right\_sum = max\_path\_sum(root.right)  if left\_sum > right\_sum:  best = left\_sum  else:  best = right\_sum  return root.value + best | |
| **Java** | |
| **public class Solution {**  **public int maxPathSum(TreeNode root) {**  **if (root == null) {**  **return 0;**  **}**  **int leftSum = maxPathSum(root.left);**  **int rightSum = maxPathSum(root.right);**  **if (leftSum > rightSum) {**  **best = leftSum;**  **} else {**  **best = rightSum;**  **}**  **return root.value + best;**  **}**  **}** | |

**Rubric (Set#A)**

| **SI** | **Category** | **Marks** |
| --- | --- | --- |
| 1 | Correct base case: if the node is None | 2 |
| 2 | Calling Left node recursively | 3 |
| 3 | Calling Right node recursively | 3 |
| 4 | Comparing and finding the maximum path from a node | 3 |
| 5 | Calculating both left and right and returning the final maximum | 4 |
| **Total = 15** | | |

**Solve (Set # B)**

| **Python** | |
| --- | --- |
| def min\_path\_sum(root):  if root is None:  return 0  leftSum = root.elem+min\_path\_sum(root.left)  rightSum = root.elem+min\_path\_sum(root.right)  if leftSum < rightSum:  return leftSum  else:  return rightSum | |
| **Java** | |
| **public static int min\_path\_sum(Node root) {**  **if (root == null) {**  **return 0;**  **}**  **int leftSum = root.elem + min\_path\_sum(root.left);**  **int rightSum = root.elem + min\_path\_sum(root.right);**  **if (leftSum < rightSum) {**  **return leftSum;**  **} else {**  **return rightSum;**  **}**  **}** | |

**Rubric (Set#B)**

| **SI** | **Category** | **Marks** |
| --- | --- | --- |
| 1 | Correct base case: if the node is None | 2 |
| 2 | Calling Left node recursively and adding root | 4 |
| 3 | Calling Right node recursively and adding root | 4 |
| 4 | Comparing and finding the minimum path from a node | 3 |
| 5 | Returning the final minimal path correctly | 2 |
| **Total = 15** | | |