Qui#02 – Solution

Question#01

|  |
| --- |
| vector<int> getRecentCalls(const vector<int>& callsHistory, int start, int end, int size, int k, int capacity) {  vector<int> recentCalls;  // Ensure we don't try to access more calls than are stored  if (k > size) k = size;  // Start from the most recent call at `end - 1` and move backwards  int index = (end - 1 + capacity) % capacity;  for (int i = 0; i < k; i++) {  recentCalls.push\_back(callsHistory[index]);  index = (index - 1 + capacity) % capacity;  // we are using circular array (queue), Move backward in circular manner  }  return recentCalls;  } |

Question#02

|  |  |
| --- | --- |
| What is the predecessor of node 33 | 25 (The left child of 33 is 25, which has no right children. This makes 25 the largest value smaller than 33 in the BST.) |
| what is the successor of node 1 | 1 (Node 2 is its parent and is the smallest value greater than 1 in the tree, as there are no nodes between 1 and 2) |

|  |
| --- |
| What will be the pre-order sequence of the above BST? |
| 2 1 33 25 11 7 12 13 40 34 36 |

What is the height of this given BST ? How does the height affect the performance of search, insert, and delete operations?

The height of this BST is 5. In a skewed tree (all nodes on one side), the height equals the number of nodes, resulting in a time complexity of O(n) for search, insert, and delete operations. A balanced BST, with height O(log n), performs these operations more efficiently.

|  |
| --- |
| Node\* insert(Node\* root, int value) {  if (!root) return new Node(value);  if (value < root->data) {  root->left = insert(root->left, value);  } else {  root->right = insert(root->right, value);  }  return root;  }  // In-order traversal to extract odd numbers from a BST and store in a vector  void extractOdds(Node\* root, vector<int>& oddNumbers) {  if (!root) return;  extractOdds(root->left, oddNumbers);  if (root->data % 2 != 0) {  oddNumbers.push\_back(root->data);  }  extractOdds(root->right, oddNumbers);  }  Node\* createBSTFromOdds(const vector<int>& oddNumbers) {  Node\* newBST = nullptr;  for (int number : oddNumbers) {  newBST = insert(newBST, number);  }  return newBST;  } |