// Question1

public void copyPaste(Queue src, int index) {

int srcCount = (src.last - src.first + src.N) % src.N;

if (srcCount == 0 || index < 0 || index >= N) {

return;

}

int insertPosition = (first + index) % N;

int shiftCount = (last - insertPosition + N) % N;

for (int i = 0; i < shiftCount; i++) {

int targetIndex = (last + srcCount - i - 1 + N) % N;

int sourceIndex = (last - i - 1 + N) % N;

array[targetIndex] = array[sourceIndex];

}

for (int i = 0; i < srcCount; i++) {

int srcIndex = (src.first + i) % src.N;

int destIndex = (insertPosition + i) % N;

array[destIndex] = src.array[srcIndex];

}

last = (last + srcCount) % N;

}

//Question 2

public void Queue(Queue[] queues) {

this.first = null;

this.last = null;

Node currentLast = null;

for (Queue queue : queues) {

if (queue.first != null) {

if (this.first == null) {

this.first = queue.first; // Set the first node of the new merged queue

} else {

currentLast.setNext(queue.first); // Link the last node of the new merged queue to the first node of the current queue

}

currentLast = queue.last; // Update last to be the last node of the current queue

}

}

this.last = currentLast;

}

//Question 3

public void accumulateLeafNodes(Queue queue) {

accumulateLeafNodesRecursive(this, queue);

}

private void accumulateLeafNodesRecursive(TreeNode node, Queue queue) {

if (node == null) {

return;

}

if (node.getLeft() == null && node.getRight() == null) {

Element leafElement = new Element(node.getData());

queue.enqueue(leafElement);

} else {

if (node.getLeft() != null) {

accumulateLeafNodesRecursive(node.getLeft(), queue);

}

if (node.getRight() != null) {

accumulateLeafNodesRecursive(node.getRight(), queue);

}

}

}

Question 4

public double simulateSearch(int N) {

if (root == null || N <= 0) {

return 0.0;

}

TreeNode current = root;

int min = root.getData();

while (current.getLeft() != null) {

current = current.getLeft();

min = current.getData();

}

current = root;

int max = root.getData();

while (current.getRight() != null) {

current = current.getRight();

max = current.getData();

}

Random random = new Random();

int totalVisited = 0;

for (int i = 0; i < N; i++) {

int target = random.nextInt(max - min + 1) + min;

current = root;

int visited = 0;

while (current != null) {

visited++;

if (target < current.getData()) {

current = current.getLeft();

} else if (target > current.getData()) {

current = current.getRight();

} else {

break;

}

}

totalVisited += visited;

}

return (double) totalVisited / N;

}

//Question 5

public static int[] sortByHashing(int[] array) {

int max = Integer.MIN\_VALUE;

for (int num : array) {

if (num > max) {

max = num;

}

}

Hash hashTable = new Hash(2 \* max + 1);

for (int num : array) {

hashTable.insert(num);

}

ArrayList<Integer> sortedList = new ArrayList<>();

for (int i = 0; i < hashTable.table.length; i++) {

if (hashTable.table[i] != null && !hashTable.deleted[i]) {

sortedList.add(hashTable.table[i].getData());

}

}

int[] sortedArray = new int[sortedList.size()];

for (int i = 0; i < sortedList.size(); i++) {

sortedArray[i] = sortedList.get(i);

}

return sortedArray;

}

//Question 6

public int[] intersection(int[] list1, int[] list2) {

Hash tempHash = new Hash(N);

for (int num : list1) {

tempHash.insert(num);

}

int[] tempResult = new int[list1.length];

int index = 0;

for (int num : list2) {

Element found = tempHash.search(num);

if (found != null && !tempHash.deleted[tempHash.hashFunction(num)]) {

tempResult[index++] = num;

tempHash.deleteValue(num);

}

}

int[] result = new int[index];

System.arraycopy(tempResult, 0, result, 0, index);

return result;

}