1.

import java.util.Arrays;

public class ShipmentAllocator {

public static int minContainers(int[] shipments, int[] containerLimits) {

Arrays.sort(shipments);

reverseArray(shipments);

Arrays.sort(containerLimits);

reverseArray(containerLimits);

int[] containersUsed = new int[containerLimits.length];

int usedCount = 0;

for (int shipment : shipments) {

boolean allocated = false;

for (int j = 0; j < containerLimits.length; j++) {

if (containersUsed[j] + shipment <= containerLimits[j]) {

containersUsed[j] += shipment;

allocated = true;

if (usedCount <= j) {

usedCount = j + 1;

}

break;

}

}

if (!allocated) {

return -1;

}

}

return usedCount;

}

private static void reverseArray(int[] array) {

int left = 0, right = array.length - 1;

while (left < right) {

int temp = array[left];

array[left] = array[right];

array[right] = temp;

left++;

right--;

}

}

public static void main(String[] args) {

int[] shipments = {10, 20, 30};

int[] containerLimits = {15, 15, 20, 10};

int result = minContainers(shipments, containerLimits);

System.out.println("Minimum number of containers needed: " + result);

}

}

2

class ListNode {

int val;

ListNode next;

ListNode(int x) {

val = x;

next = null;

}

}

public class Solution {

public boolean hasCycle(ListNode head) {

if (head == null || head.next == null) {

return false; // If the list is empty or has only one node, it can't have a cycle

}

ListNode slow = head;

ListNode fast = head;

while (fast != null && fast.next != null) {

slow = slow.next;

fast = fast.next.next;

if (slow == fast) {

return true;

}

}

return false;

}

public static void main(String[] args) {

ListNode node1 = new ListNode(20);

ListNode node2 = new ListNode(30);

ListNode node3 = new ListNode(40);

ListNode node4 = new ListNode(60);

ListNode node5 = new ListNode(80);

node1.next = node2;

node2.next = node3;

node3.next = node4;

node4.next = node5;

node5.next = node3;

Solution solution = new Solution();

System.out.println("Has cycle: " + solution.hasCycle(node1));

ListNode node6 = new ListNode(6);

ListNode node7 = new ListNode(4);

ListNode node8 = new ListNode(2);

ListNode node9 = new ListNode(8);

node6.next = node7;

node7.next = node8;

node8.next = node9;

System.out.println("Has cycle: " + solution.hasCycle(node6));

}

}