**DS & Algo Interview Questions**

**1- Sort 0's and 1's**

Step 1: Let there be two counters – left and right. left will be pointing at the beginning and right will be pointing at the end of inputArray.

Step 2: If inputArray[left] is equal to 1, swap inputArray[left] with inputArray[right] and decrement right count.

Step 3: If inputArray[left] is not equal to 1, don’t swap. Just increment left counter.

Step 4: Follow step 2 and step 3 until left < right. At the end, you will get sorted array.

public class Array0s1sProgram {  
 private static void sortBinaryArray(int[] inputArray) {  
 int left = 0;  
 int right = inputArray.length - 1;  
  
 System.*out*.println("Input Array Before Sorting : " + Arrays.*toString*(inputArray));  
  
 while (left < right) {  
 if (inputArray[left] == 1) {  
 //Swapping  
 inputArray[right] = inputArray[right] + inputArray[left];  
 inputArray[left] = inputArray[right] - inputArray[left];  
 inputArray[right] = inputArray[right] - inputArray[left];  
  
 right--;  
 } else {  
 left++;  
 }  
 }  
 System.*out*.println("Input Array After Sorting : " + Arrays.*toString*(inputArray));  
 }  
  
 public static void main(String[] args) {  
 *sortBinaryArray*(new int[]{1, 0, 1, 1, 0, 1, 0, 0});  
 System.*out*.println("============================");  
 *sortBinaryArray*(new int[]{1, 1, 1, 1, 0, 0, 0, 0});  
 }  
}

Output:

Input Array Before Sorting : [1, 0, 1, 1, 0, 1, 0, 0]

Input Array After Sorting : [0, 0, 0, 0, 1, 1, 1, 1]

============================

Input Array Before Sorting : [1, 1, 1, 1, 0, 0, 0, 0]

Input Array After Sorting : [0, 0, 0, 0, 1, 1, 1, 1]

**2- Sort Absolutely sorted LinkedList**

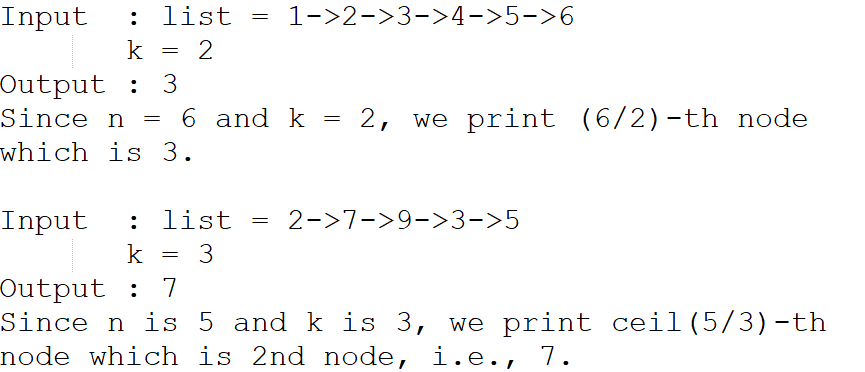
Input : 1 -> -2 -> -3 -> 4 -> -5

output: -5 -> -3 -> -2 -> 1 -> 4

* An efficient solution can work in O(n) time.
* An important observation is, all negative elements are present in reverse order.
* So, we traverse the list, whenever we find an element that is out of order, we move it to the front of the linked list.

public class SortAbsolutelySortedLL {  
 Node sortAbsolutelySortedLL(Node head) {  
 if (head == null) {  
 return null;  
 }  
 Node prev = head;  
 Node curr = head.next;  
  
 while (curr != null) {  
 // If curr is smaller than prev, then it must be moved to head  
 if (curr.data < prev.data) {  
 // Detach curr from linked list  
 prev.next = curr.next;  
  
 // Move current node to beginning  
 curr.next = head;  
 head = curr;  
  
 // Update current  
 curr = prev;  
 } else {  
 // Nothing to do if current element is at right place  
 prev = curr;  
 }  
  
 // Move current  
 curr = curr.next;  
 }  
 return head;  
 }  
}

**3- Find the fractional (or n/k – th) node in linked list**

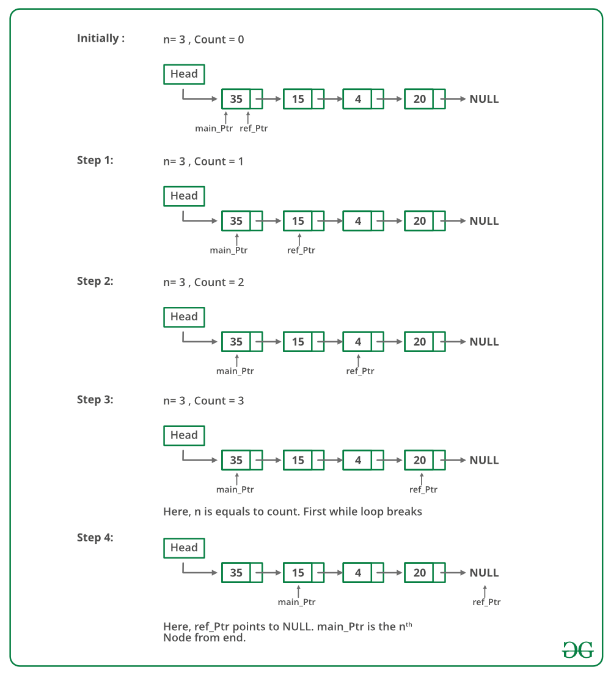


**Solution:**

1. Take two pointers temp and fractionalNode and initialize them with null and head respectively.
2. For every k jumps of the temp pointer, make one jump of the fractionalNode pointer.

**4- nth node from end**

* Maintain two pointers – reference pointer and main pointer.
* Initialize both reference and main pointers to head.
* First, move the reference pointer to n nodes from head.
* Now move both pointers one by one until the reference pointer reaches the end.
* Now the main pointer will point to nth node from the end.
* Return the main pointer.



**5- Sort array on frequency**

1. Iterate over array, maintain HashMap that maintains count of numbers present in array.
2. Sort based on the count.