1.algorithm for merging linked list

1.Initialize two pointers to the start of both the lists.
2.Initialize a sentinel node with a Integer min value as a dummy node. This value serves as a placeholder to start our merged list. We would discard this later.
3.Compare the values at both the pointers of the 2 lists and pick the smaller one.
4.Append this smaller value to the next sentinel node and move the pointer one step forward from the list that we got the number.
5.Again compare both the values at the two pointers and keep appending the numbers to the list.
6.Stop when we reach the end of one of the list. This would mean that all the elements of this list are finished and the rest of the elements in the second list are greater. So just append all the remaining elements to the sorted list.
7.Return the pointer next to the sentinel node that we defined in step 2.

2.hash table:

1.Select a List, that has fewer number of elements. We can get the number of elements, by a single scan on both the lists. If both the lists have same number of elements, select any list at random.
2.Create a hash table using the list with fewer elements. Creating a hash table means storing the address of each of the nodes of the smaller list in a separate data structure such as an array.
3.Now, traverse the other list and compare the address of each of the node with the values in the hash table.
4.If there exists an intersection point, certainly we will find a match in the hash table and we will obtain the intersection point.

3.using stack:

1.Create 2 different stacks for both the lists.
2.Push all the elements of both the lists in the 2 stacks.
3.Now start POPing the elements from both the stacks at once.
4.Till both the lists are merged, we will get the same value from both the stacks.
5.As soon as both the stacks return different value, we know that the last popped element was the merging point of the lists.
6.Return the last popped element from the stack.

4.sorting:

1.Create an array and store all the addresses of the nodes.
2.Now sort this array.

3.For each element in the second list, from the beginning search for the address in the array. We can use a very efficient search algorithm like Binary Search which gives us the result in O(log n).
4.If we find a same memory address, that means that is the merging point of the 2 lists.

5.other ways

1.Simply use two loops
2.Mark Visited Nodes
3.Using difference of node counts
4.Make a circle in first list
5.Reverse the first list and make equations
6.Traverse both lists and compare addresses of last nodes
7.Use Hashing
8.2-pointer technique

6.complexity

1.Find the length of both the lists. Let ‘m’ be the length of List 1 and ‘n’ be the length of List 2.
2.Find the difference in length of both the lists. d = m – n
3.Move ahead ‘d’ steps in the longer list.
4.This means that we have reached a point after which, both of the lists have same number of nodes till the end.
5.Now move ahead in both the lists in parallel, till the ‘NEXT’ of both the lists is not the same.
6.The NEXT at which both the lists are same is the merging point of both the lists.

- level order traversing

1) Create an empty queue q
2) temp_node = root
3) Loop while temp_node is not NULL
   a) Enqueue temp_node’s children to q
   b) Increase count with every enqueuing.
   c) Dequeue a node from q and assign it’s value to temp_node

```
int sizeof tree(Node *root)
{
  if(root == NULL)
        return 0;
      queue<Node *> q;
    int count = 1;
    q.push(root);
       while(!q.empty())
```

```
   {
      Node *temp = q.front();
      if(temp->left)
      q.push(temp->left);
 count++;
      }
     if(temp->right)
      {
          q.push(temp->right);
          count++;
      }
      q.pop();
   }
     return count;
}
```