# Project sheet

1) **Algorithm**

1) traverse the first list and for each element in the first list.

2) check if that node is also present in the second list.

3) Return the first node found in the first list that is also present in second list.

2)using sorting :

Alogrithm

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.

3) using hashing

Alogrithm

1) Select list that has fewer number of elements.we can get the number of elements,by a single scan on both the list .if both the list have same number of element ,selec any list at random.

2) Create hash tabl using list with fewer elements .

3) Now,traverse the other list and compare the address of each of the node with the value in the hash table

4) using stack:

Alogorithm for merging point

1) Create two stacks for both lists.

2) push all elemenys of both list into their stacks.

3) pop the elements from both stacks at once.

4) till both list are merged ,we will get the same value from both the stacks.

5) As soon as both the stacks return different value ,we known that the last popped element was the merging points of the list.

6) Return the last popped elment from the stack.

5) yes ,there is an other way of solving by using Floyd cycle detection algorithm.

6) yes, we can improve the complexity

Algorithm:

```
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.
```

time complexity -> 0(max(m,n))

space complexity ->O(1)

****************************************************

**Algorithm for finding size of tree:**

```
1) if the tree is empty then return 0
2) else
      i) find size of left subtree
           size(tree->left subtree)
      ii) find size of right subtree
           size (tree->right subtree)
      iii) tree size =size(tree->left subtree) +size (tree->right subtree) + 1

      iv) return tree size
```

a) **without recursion**

```
1)create queue and push root
2)height =0
Loop
```

```
   nodeCount = size of queue
 3) if nodecount is 0
      return height
4) else
      increment height
5) while (node Count>0)
     pop node from front
     push its children to queue
     decrease nodecount
```