1. 1184. Distance Between Bus Stops
   We will get the distance from start to end and compare it with the other distance. Return the minimum one. TC is O(n)
   ```
   class Solution:
       def distanceBetweenBusStops(self, distance: List[int], start: int, destination: int) -> int:
           if start > destination:
             start, destination = destination, start
           one_sum = sum(distance[start:destination])
           return min(one_sum, sum(distance) - one_sum)
   ```

2. 1185. Day of the Week
   We will use datetime to get the week day. TC is O(1)
   ```
   import datetime
   class Solution:
       def dayOfTheWeek(self, day: int, month: int, year: int) -> str:
           weekdays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
           return weekdays[datetime.date(year, month, day).weekday()]
   ```

3. 876. Middle of the Linked List
   We will use two nodes, the slower one will move one node every step, and the fast one will move two nodes every step. We will repeat the previous progress until the fast one cannot move. TC is O(n).
   ```
   class Solution:
       def middleNode(self, head: ListNode) -> ListNode:
           dummy = ListNode(0)
           dummy.next = head
           slow, fast = dummy, dummy
           while fast:
             slow = slow.next
             if not fast.next or not fast.next.next:
               return slow
             else:
               fast = fast.next.next
   ```
4. Check If a Number Is Majority Element in a Sorted Array
   We will use binary search to get our starting index and ending index of our target number. Then compare their difference with our len / 2 and return the result. TC is O(logn)
   ```
   from bisect import *
   class Solution:
       def isMajorityElement(self, nums: List[int], target: int) -> bool:
           return bisect(nums, target) - bisect_left(nums, target) > len(nums) / 2
   ```

5. Increasing Order Search Tree
   We will use in-order traverse to get all nodes we need. Then we will reconnect them one by one. TC is O(n)

```
class Solution:
    def increasingBST(self, root: TreeNode) -> TreeNode:
        nodes = []
        self.traverse(root, nodes)
        dummy = TreeNode(0)
        dummy_mem = dummy
        for node in nodes:
          node.left = None
          dummy.right = node
          dummy = dummy.right
        dummy.right = None
        dummy.left = None
        return dummy_mem.right

    def traverse(self, root, nodes):
      if not root:
        return None
      if root.left:
        self.traverse(root.left, nodes)
      nodes.append(root)
      if root.right:
        self.traverse(root.right, nodes)
```