1. Merge Two Sorted Lists:-

Code:-

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def mergeTwoLists(list1, list2):
    dummy = ListNode()
    current = dummy

    while list1 and list2:
        if list1.val < list2.val:
            current.next = list1
            list1 = list1.next
        else:
            current.next = list2
            list2 = list2.next
        current = current.next

    if list1:
        current.next = list1
    elif list2:
        current.next = list2
```

```
        return dummy.next
```

Time complexity: O(n+m)
Space complexity: O(1)

2. Merge k Sorted Lists:-

Code:-

```
from heapq import heappop, heappush

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def mergeKLists(lists):
    min_heap = []
    for index, node in enumerate(lists):
        if node:
            heappush(min_heap, (node.val, index, node))
```

```
    dummy = ListNode()
    current = dummy

    while min_heap:
        val, index, node = heappop(min_heap)
        current.next = node
        current = current.next
        if node.next:
            heappush(min_heap, (node.next.val, index,
node.next))

    return dummy.next

# Example usage:
# lists = [[1,4,5],[1,3,4],[2,6]]
# Result: 1 -> 1 -> 2 -> 3 -> 4 -> 4 -> 5 -> 6
```

Time complexity: O(N log k)

Space complexity: O(k)

3. Remove Duplicates from Sorted Array:-

Code:-

```
def removeDuplicates(nums):
if not nums:
    return 0

slow = 0
for fast in range(1, len(nums)):
    if nums[fast] != nums[slow]:
        slow += 1
        nums[slow] = nums[fast]

return slow + 1
```

Time complexity: O(log n)

Space complexity: O(1)

4. Search in Rotated Sorted Array:-

Code:-

```
def search(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid

        if nums[left] <= nums[mid]:
            if nums[left] <= target < nums[mid]:
                right = mid - 1
            else:
                left = mid + 1
        else:
```

```
            if nums[mid] < target <=
        nums[right]:
                left = mid + 1
            else:
                right = mid - 1

        return -1
```

Time complexity: O(log n)

Space complexity: O(1)

5. Find First and Last Position of Element in Sorted Array:-

Code:-

```
def searchRange(nums, target):
    def findBound(isFirst):
        left, right = 0, len(nums) - 1
        bound = -1
```

```
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            bound = mid
            if isFirst:
                right = mid - 1
            else:
                left = mid + 1
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return bound

first = findBound(True)
if first == -1:
    return [-1, -1]
last = findBound(False)
return [first, last]
```

Time complexity: O(log n)

Space complexity: O(1)

6. Find Minimum in Rotated Sorted Array:-

code:-

```
    def findMin(nums):
left, right = 0, len(nums) - 1

while left < right:
    mid = (left + right) // 2
    if nums[mid] > nums[right]:
        left = mid + 1
    else:
        right = mid

return nums[left]
```

Time complexity: O(log n)

Space complexity: O(1)

7. Valid Parentheses:-

Code:-

```
def isValid(s):
    stack = []
    mapping = {')': '(', '}': '{', ']': '['}

    for char in s:
        if char in mapping:
            top_element = stack.pop() if stack else '#'
            if mapping[char] != top_element:
                return False
        else:
            stack.append(char)

    return not stack
```

Time complexity: O(n)

Space complexity: O(n)

8. Implement strStr():-

Code:-

```
def strStr(haystack, needle):
    if not needle:
        return 0

    lps = [0] * len(needle)
    j = 0

    # Compute LPS array
    for i in range(1, len(needle)):
        while j > 0 and needle[i] != needle[j]:
            j = lps[j - 1]
        if needle[i] == needle[j]:
            j += 1
```

```
            lps[i] = j

    j = 0
    for i in range(len(haystack)):
        while j > 0 and haystack[i] != needle[j]:
            j = lps[j - 1]
        if haystack[i] == needle[j]:
            j += 1
        if j == len(needle):
            return i - j + 1

    return -1
```

Time complexity: O(n+m)

Space complexity: O(m)

9. Longest Common Prefix:-

Code:-

```
def longestCommonPrefix(strs):
```

```
        if not strs:
            return ""

        prefix = strs[0]
        for s in strs[1:]:
            while s[:len(prefix)] != prefix and prefix:
                prefix = prefix[:-1]
        return prefix
```

Time complexity: O(s)

Space complexity: O(1)

10. Pow(x, n):-

Code:-

```
def myPow(x, n):
if n == 0:
    return 1
if n < 0:
    x = 1 / x
```

```
        n = -n

    result = 1
    while n:
        if n % 2:
            result *= x
        x *= x
        n //= 2

    return result
```

Time Complexity: O(log n)O(\log n)O(logn)
Space Complexity: O(1)O(1)O(1).