1. Convert the Temperature

```
def con(celsius):
    k = round(celsius + 273.15, )
    f = round(celsius * 1.8 + 32.0, 5)
    return [k, f]
c = 36.50
ans = con(c)
print(ans)
```

2. Number of Subarrays With LCM Equal to K

```
import math
from collections import defaultdict

def count_subarrays(nums, k):
    def gcd(a, b):
        while b:
            a, b = b, a % b
        return a

    def lcm(a, b):
        return a * b // gcd(a, b)

    count = 0
    n = len(nums)
    prefix_gcd = defaultdict(int)
```

```
    prefix_gcd[1] = 1

    left = 0
    current_gcd = 0

    for right in range(n):
        current_gcd = gcd(current_gcd, nums[right])

        while current_gcd % k != 0:
            left += 1
            current_gcd = prefix_gcd[left]

        count += right - left + 1
        prefix_gcd[right + 1] = current_gcd

    return count
nums = [3, 6, 2, 7, 1]
k = 6
result = count_subarrays(nums, k)
print(result)
```

3. Minimum Number of Operations to Sort a Binary Tree by Level

```
class Solution:
 def minimumOperations(self, root:
Optional[TreeNode]) -> int:
 def cost(a):
 b = sorted(a)
```

```
c = 0
d = defaultdict(int)
for idx,e in enumerate(a):
d[e]=idx
for idx,e in enumerate(b):
if a[idx]==e:
continue
pos = d[e]
a[idx],a[pos] = a[pos],a[idx]
d[a[pos]] = pos
c+=1
return c
d = defaultdict(list)
dfs = [(root,1)]
while dfs:
node,h = dfs.pop()
d[h].append(node.val)
if node.right:
dfs.append((node.right,h+1))
if node.left:
dfs.append((node.left,h+1))
ans = 0
for m in d.values():
ans += cost(m)
return ans
```

4. Maximum Number of Non-overlapping Palindrome Substrings

```
class Solution:
 def maxPalindromes(self, s: str, k: int) -> int:
 n = len(s)
 dp = [0] * (n + 1)
 for i in range(k, n + 1):
 dp[i] = dp[i - 1]
 for length in range(k, k + 2):
 j = i - length
 if j < 0:
 break
 if self.isPalindrome(s, j, i):
 dp[i] = max(dp[i], 1 + dp[j])
 return dp[-1]
 def isPalindrome(self, s, j, i):
 left, right = j, i - 1
 while left < right:
 if s[left] != s[right]:
 return False
 left += 1
 right -= 1
 return True
```

5. Minimum Cost to Buy Apples

```
import heapq
from collections import defaultdict
from typing import List
import sys
inf = sys.maxsize
heappop = heapq.heappop
```

```
heappush = heapq.heappush
class Solution:
 def minCost(
 self, n: int, roads: List[List[int]], appleCost: List[int], k:
int
 ) -> List[int]:
 def dijkstra(i):
 q = [(0, i)]
 dist = [inf] * n
 dist[i] = 0
 ans = inf
 while q:
 d, u = heappop(q)
 ans = min(ans, appleCost[u] + d * (k + 1))
 for v, w in g[u]:
 if dist[v] > dist[u] + w:
 dist[v] = dist[u] + w
heappush(q, (dist[v], v))
 return ans
 g = defaultdict(list)
 for a, b, c in roads:
 a, b = a - 1, b - 1
 g[a].append((b, c))
 g[b].append((a, c))
 return [dijkstra(i) for i in range(n)]
def main():
 n = 5
 roads = [[1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5, 6], [1, 5, 2]]
 appleCost = [10, 20, 30, 40, 50]
```

```
 k = 2
 solution = Solution()
 result = solution.minCost(n, roads, appleCost, k)
 print(result)
if _name_ == "_main_":
 main()
```