```
def convert_temperature(celsius):
    kelvin = celsius + 273.15
    fahrenheit = celsius * 1.80 + 32.00
    return [round(kelvin, 5), round(fahrenheit, 5)]
celsius = 36.50
print(convert_temperature(celsius))



import math
from functools import reduce


def lcm(x, y):
    return abs(x * y) // math.gcd(x, y)


def lcm_of_list(lst):
    return reduce(lcm, lst, 1)


def subarrays_with_lcm(nums, k):
    count = 0
    for i in range(len(nums)):
        for j in range(i, len(nums)):
            subarray_lcm = lcm_of_list(nums[i:j+1])
            if subarray_lcm == k:
                count += 1
    return count
nums = [3, 6, 2, 7, 1]
```

```
k = 6
print(subarrays_with_lcm(nums, k))


class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def min_operations_to_sort_levels(root):
    from collections import deque

    if not root:
        return 0

    def level_order_traversal(root):
        levels = []
        queue = deque([root])
        while queue:
            level_size = len(queue)
            level = []
            for _ in range(level_size):
                node = queue.popleft()
                level.append(node.val)
                if node.left:
```

```
            queue.append(node.left)
          if node.right:
            queue.append(node.right)
        levels.append(level)
    return levels

def min_swaps_to_sort(arr):
    n = len(arr)
    sorted_arr = sorted(arr)
    index_map = {value: idx for idx, value in enumerate(arr)}
    swaps = 0
    for i in range(n):
        if arr[i] != sorted_arr[i]:
            swaps += 1
            swap_idx = index_map[sorted_arr[i]]
            index_map[arr[i]] = swap_idx
            arr[i], arr[swap_idx] = arr[swap_idx], arr[i]
    return swaps

levels = level_order_traversal(root)
total_swaps = 0
for level in levels:
    total_swaps += min_swaps_to_sort(level)

return total_swaps
```

```python
root = TreeNode(1)
root.left = TreeNode(4)
root.right = TreeNode(3)
root.left.left = TreeNode(7)
root.left.right = TreeNode(6)
root.right.left = TreeNode(8)
root.right.right = TreeNode(5)
root.right.left.left = TreeNode(9)
root.right.left.right = TreeNode(10)
print(min_operations_to_sort_levels(root))
```

```python
def max_palindrome_substrings(s, k):
    def is_palindrome(sub):
        return sub == sub[::-1]

    n = len(s)
    dp = [[0] * (n + 1) for _ in range(n + 1)]

    for length in range(k, n + 1):
        for i in range(n - length + 1):
            j = i + length
            if is_palindrome(s[i:j]):
                for x in range(i + k, j + 1):
                    dp[j][x] = max(dp[j][x], dp[i][x - k] + 1)
```

```
    return max(max(row) for row in dp)
s = "abaccdbbd"
k = 3
print(max_palindrome_substrings(s, k))
```

```
import heapq
def min_cost_to_buy_apples(n, roads, appleCost, k):
    graph = {i: [] for i in range(1, n + 1)}
    for u, v, cost in roads:
        graph[u].append((v, cost))
        graph[v].append((u, cost))

    def dijkstra(start):
        heap = [(0, start)]
        dist = {i: float('inf') for i in range(1, n + 1)}
        dist[start] = 0
        while heap:
            current_dist, node = heapq.heappop(heap)
            if current_dist > dist[node]:
                continue
            for neighbor, weight in graph[node]:
                distance = current_dist + weight
                if distance < dist[neighbor]:
```

```python
                dist[neighbor] = distance
                heapq.heappush(heap, (distance, neighbor))
        return dist

    min_cost = []
    for i in range(1, n + 1):
        dist = dijkstra(i)
        min_cost_i = float('inf')
        for j in range(1, n + 1):
            if i != j:
                min_cost_i = min(min_cost_i, dist[j] + appleCost[j - 1] + dist[j] * k)
            else:
                min_cost_i = min(min_cost_i, appleCost[j - 1])
        min_cost.append(min_cost_i)

    return min_cost
n = 4
roads = [[1, 2, 4], [2, 3, 2], [2, 4, 5], [3, 4, 1], [1, 3, 4]]
appleCost = [56, 42, 102, 301]
k = 2
print(min_cost_to_buy_apples(n, roads, appleCost, k))




def count_unequal_triplets(nums):
```

```python
    count = 0
    n = len(nums)
    for i in range(n - 2):
        for j in range(i + 1, n - 1):
            for k in range(j + 1, n):
                if nums[i] != nums[j] and nums[i] != nums[k] and nums[j] != nums[k]:
                    count += 1
    return count
nums = [4, 4, 2, 4, 3]
print(count_unequal_triplets(nums))
```

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def inorder_traversal(root):
    result = []
    stack = []
    current = root
    while stack or current:
        while current:
            stack.append(current)
```

```python
            current = current.left
        current = stack.pop()
        result.append(current.val)
        current = current.right
    return result

def find_closest_values(sorted_values, query):
    left, right = 0, len(sorted_values) - 1
    min_val, max_val = -1, -1
    while left <= right:
        mid = (left + right) // 2
        if sorted_values[mid] >= query:
            max_val = sorted_values[mid]
            right = mid - 1
        else:
            left = mid + 1
    left, right = 0, len(sorted_values) - 1
    while left <= right:
        mid = (left + right) // 2
        if sorted_values[mid] <= query:
            min_val = sorted_values[mid]
            left = mid + 1
        else:
            right = mid - 1

    return [min_val, max_val]
```

```
def closest_nodes(root, queries):
    sorted_values = inorder_traversal(root)
    return [find_closest_values(sorted_values, query) for query in queries]

root = TreeNode(6)
root.left = TreeNode(2)
root.right = TreeNode(13)
root.left.left = TreeNode(1)
root.left.right = TreeNode(4)
root.right.left = TreeNode(9)
root.right.right = TreeNode(15)
root.right.right.left = TreeNode(14)

queries = [2, 5, 16]
print(closest_nodes(root, queries))
```

```
from collections import defaultdict
import math

def minimumFuelCost(roads, seats):
    n = len(roads) + 1
    tree = defaultdict(list)
```

```
    for a, b in roads:
        tree[a].append(b)
        tree[b].append(a)

    def dfs(node, parent):
        total_representatives = 1
        fuel = 0

        for neighbor in tree[node]:
            if neighbor != parent:
                sub_representatives, sub_fuel = dfs(neighbor, node)
                total_representatives += sub_representatives
                fuel += sub_fuel
        if node != 0:
            cars_needed = math.ceil(total_representatives / seats)
            fuel += cars_needed
        return total_representatives, fuel
    _, total_fuel = dfs(0, -1)
    return total_fuel
print(minimumFuelCost([[0, 1], [0, 2], [0, 3]], 5))
```

```
MOD = 10**9 + 7
```

```
def is_prime_digit(ch):
    return ch in {'2', '3', '5', '7'}
def num_beautiful_partitions(s, k, minLength):
    n = len(s)
    dp = [[0] * (k + 1) for _ in range(n + 1)]
    dp[0][0] = 1
    for i in range(1, n + 1):
        for j in range(1, k + 1):
            for l in range(minLength, i + 1):
                if is_prime_digit(s[i - l]) and not is_prime_digit(s[i - 1]):
                    dp[i][j] = (dp[i][j] + dp[i - l][j - 1]) % MOD

    return dp[n][k]
print(num_beautiful_partitions("23542185131", 3, 2))
```