In [ ]:
```
1.temperature conversion
```

In [5]:
```
celsius = 36.50
f =(celsius * 9/5) + 32
print("fahrenheit",f)
```

```
fahrenheit 97.7
```

In [ ]:
```
2.Number of Subarrays With LCM Equal to k
```

In [8]:
```
from math import gcd
def count_sub_lcm(arr, k):
    def lcm(a,b):
        return a*b //gcd(a, b)
    n=len(arr)
    count = 0
    for i in range(n):
        curr_lcm=arr[i]
        for j in range(i, n):
            curr_lcm=lcm(curr_lcm, arr[j])
            if curr_lcm==k:
                count+= 1
    return count
arr= [3]
k=2
print(count_sub_lcm(arr, k))
```

```
0
```

```
3. Minimum Number of Operations to Sort a binary tree
```

In [9]:
```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
def min_binary_tree(root):
    if not root:
        return 0
    queue = [root]
    operations = 0
    while queue:
        next_level = []
        sorted_level = sorted([node.value for node in queue])
        for i, node in enumerate(queue):
            node.value = sorted_level[i]
            if node.left:
                next_level.append(node.left)
            if node.right:
                next_level.append(node.right)
        queue = next_level
        operations += 1
    return operations
root = TreeNode(4)
root.left = TreeNode(2)
root.right = TreeNode(7)
root.left.left = TreeNode(1)
root.left.right = TreeNode(3)
root.right.left = TreeNode(6)
root.right.right = TreeNode(9)
print(min_binary_tree(root))
```

```
3
```

4.Maximum Number of Non-overlapping Palindrome Substring

In [10]:
```
s = "ababa"
n = len(s)
dp = [0] * n
max_palindromes = 0
for i in range(n):
    for j in range(i, -1, -1):
        if s[j:i + 1] == s[j:i + 1][::-1]:
            dp[i] = max(dp[i], dp[j - 1] + 1 if j > 0 else 1)
            max_palindromes = max(max_palindromes, dp[i])
print(max_palindromes)
```

```
5
```

5.Minimum Cost of apples

In [12]:
```
def min_cost_to_buy_apple(n, k):
    if k == 0:
        return [0] * n
    ans = [0] * n
    for i in range(n):
        ans[i] = (i + 1) * k
    return ans
n = 5
k = 2
print(min_cost_to_buy_apple(n, k))
```

```
[2, 4, 6, 8, 10]
```

In [ ]:
```
6.Customers With Strictly Increase
```

In [18]:
```
customers = [
    {"id": 1, "name": "John", "purchases": [100, 120, 150]},
    {"id": 2, "name": "Alice", "purchases": [80, 100, 120]},
    {"id": 3, "name": "Bob", "purchases": [50, 60, 70]}
]

print("ID\tName\tPurchases")
print("-" * 50)
for customer in customers:
    purchases = ", ".join(map(str, customer["purchases"]))
    print(f"{customer['id']}\t{customer['name']}\t{purchases}")
```

```
ID      Name    Purchases
--------------------------------------------------
1       John    100, 120, 150
2       Alice   80, 100, 120
3       Bob     50, 60, 70
```

In [ ]:
```
7. Number of Unequal Triplets in Array
```

In [19]:
```
def count_unequal_triplets(arr):
    n =len(arr)
    count = 0
    for i in range(n):
        for j in range(i+1, n):
            for k in range(j+1, n):
                if arr[i]!= arr[j] and arr[j]!= arr[k] and arr[i]!= arr[k]:
                    count += 1
    return count
arr = [1, 2, 3, 4, 5]
print(count_unequal_triplets(arr))
```

```
10
```

In [ ]:
```
8.Closest Nodes Queries in a Binary search tree
```

In [20]:
```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left= None
        self.right= None
def closest_value(root, target):
    closest = float('inf')
    closest_node = None
    while root:
        if abs(root.value-target)<closest:
            closest= abs(root.value - target)
            closest_node= root.value
        if target<root.value:
            root=root.left
        else:
            root=root.right
    return closest_node
root= TreeNode(10)
root.left=TreeNode(5)
root.right = TreeNode(15)
root.left.left= TreeNode(2)
root.left.right=TreeNode(7)
root.right.left= TreeNode(12)
root.right.right= TreeNode(17)
target=8
print(closest_value(root, target))
```

```
7
```

In [ ]:
```
9.Minimum Fuel Cost to Report capital
```

In [21]:
```
import heapq
def min_fuel_cost_to_report(graph, start):
    n = len(graph)
    distance = [float('inf')] * n
    distance[start]= 0
    pq= [(0, start)]

    while pq:
        cost, node= heapq.heappop(pq)
        if distance[node]< cost:
            continue
        for neighbor, fuel_cost in graph[node]:
            if cost+ fuel_cost<distance[neighbor]:
                distance[neighbor]=cost+ fuel_cost
                heapq.heappush(pq, (distance[neighbor], neighbor))

    return distance
graph = {
    0: [(1, 2), (2, 4)],
    1: [(0, 2), (3, 5)],
    2: [(0, 4), (3, 1)],
    3: [(1, 5), (2, 1)]
}
start=0
print(min_fuel_cost_to_report(graph, start))
```

```
[0, 2, 4, 5]
```

In [1]:
```
def count_beautiful_partitions(n):
    if n==0:
        return 1
    return 2 ** (bin(n).count('1') - 1)
n=10
beautiful_partitions= count_beautiful_partitions(n)
print(beautiful_partitions)
```

```
2
```

In [ ]:
```

```