1.Height of Binary Tree After Subtree Removal Queries

```
class Solution:
 def treeQueries(self, root: Optional[TreeNode], q : List[int]) -> List[int]:
 lvl = defaultdict(int)
 mx = 0
 def level(node):
 nonlocal mx
 if not node:
 return 0
 l = level(node.left)
 r = level(node.right)
 lvl[node.val] = max(l,r) +1
 mx = max(mx, lvl[node.val])
 return lvl[node.val]
 level(root)
 n = len(lvl) + 3
 ans = [mx-1 for i in range(n+1)]
 curmx = 0
 def helper(node, cur):
```

```
if not node:
return
nonlocal curmx
# print(node.val)
l,r = 0,0
if node.left:
l = lvl[node.left.val]
if node.right:
r = lvl[node.right.val]
if r > l:
right = node.right.val
curmx = max(curmx, cur + l)
ans[right] = curmx
helper(node.right, cur + 1)
elif l > r:
left = node.left.val
curmx = max(curmx, cur + r)
```

2.Sort Array by Moving Items to Empty Space

```
from typing import List
class Solution:
 def sortArray(self, nums: List[int]) -> int:
 def f(nums, k):
 vis = [False] * n
 cnt = 0
 for i, v in enumerate(nums):
 if i == v or vis[i]:
 continue
 cnt += 1
 j = i
 while not vis[j]:
 vis[j] = True
cnt += 1
j = nums[j]
 return cnt - 2 * (nums[k] != k)
 n = len(nums)
 a = f(nums, 0)
 b = f([(v - 1 + n) % n for v in nums], n - 1)
 return min(a, b)
if _name_ == "_main_":
 solution = Solution()
 nums = [2, 0, 1, 4, 3]
 result = solution.sortArray(nums)
 print("Minimum swaps needed:", result)
```

3. Apply Operations to an Array

```
class Solution:
 def applyOperations(self, nums: List[int]) -> List[int]:
 zeros=0
 i=0
 while(i<(len(nums)-1)):
 if(nums[i]==nums[i+1]):
 nums[i]*=2
 nums[i+1]=0
 i+=1
 i+=1
 # print(nums)
 zeros=nums.count(0)
 nums = [i for i in nums if i != 0]
 return nums+([0]*zeros)
```

4. Maximum SumofDistinct Subarrays With Length K

Code:

```
class Solution:
 def maximumSubarraySum(self, nums: List[int], k: int) -> int:
```

```
l, r =0, 0
mx, total = 0, 0
visit = set()
while r<len(nums):
while nums[r] in visit:
total -= nums[l]
visit.remove(nums[l])
l+=1
total += nums[r]
visit.add(nums[r])
if (r-l+1) == k:
mx = max(mx, total)
total -= nums[l]
visit.remove(nums[l])
l+=1
r+=1
return mx
```

5. Total Cost to Hire K Workers

Code:

```
class Solution:
 def totalCost(self, costs: List[int], k: int, candidates: int) -> int:
```

```
 q = costs[:candidates]
 qq = costs[max(candidates, len(costs)-candidates):]
 heapify(q)
 heapify(qq)
 ans = 0
 i, ii = candidates, len(costs)-candidates-1
 for _ in range(k):
 if not qq or q and q[0] <= qq[0]:
 ans += heappop(q)
 if i <= ii:
 heappush(q, costs[i])
i += 1
 else:
 ans += heappop(qq)
 if i <= ii:
 heappush(qq, costs[ii])
ii -= 1
 return ans
```

6. Minimum Total Distance Traveled

Code:

```
class Solution:
 def minimumTotalDistance(self, robot: List[int], factory:
List[List[int]]) -> int:
 robot.sort()
 factory.sort()
 m, n = len(robot), len(factory)
 dp = [[0]*(n+1) for _ in range(m+1)]
 for i in range(m): dp[i][-1] = inf
 for j in range(n-1, -1, -1):
 prefix = 0
 qq = deque([(m, 0)])
 for i in range(m-1, -1, -1):
 prefix += abs(robot[i] - factory[j][0])
 if qq[0][0] > i+factory[j][1]: qq.popleft()
 while qq and qq[-1][1] >= dp[i][j+1] - prefix: qq.pop()
 qq.append((i, dp[i][j+1] - prefix))
 dp[i][j] = qq[0][1] + prefix
 return dp[0][0]
```

7. Minimum Sub arrays in a Valid Split

Code:

```
from typing import List
from math import gcd, inf
from functools import cache
class Solution:
 def validSubarraySplit(self, nums: List[int]) -> int:
def dfs(i):
 if i >= n:
 return 0
 ans = inf
 for j in range(i, n):
 if gcd(nums[i], nums[j]) > 1:
 ans = min(ans, 1 + dfs(j + 1))
 return ans
 n = len(nums)
 ans = dfs(0)
 dfs.cache_clear()
 return ans if ans < inf else -1
if _name_ == "_main_":
 solution = Solution()
 nums = [2, 3, 4, 9, 6]
 result = solution.validSubarraySplit(nums)
 print("Minimum subarray splits needed:", result)
```

8. Number of Distinct Averages

Code:

```
class Solution:
 def distinctAverages(self, nums: List[int]) -> int:
 a=[]
 for i in range(len(nums)//2):
 a.append((max(nums)+min(nums))/2)
 nums.remove(max(nums))
 nums.remove(min(nums))
 b=set(a)
 print(a)
 print(b)
 return len(b)
```

9. Count Ways To Build Good Strings

Code:

```
class Solution:
 def countGoodStrings(self, l: int, h: int, zero: int, one: int) ->
int:
 return (f := cache(lambda n: n <= h and ((n >= l) + f(n + zero)
+
f(n + one)) % 1_000_000_007))(0)
```

10. Most Profitable Path in a Tree

Code:

```
class Solution:
 def mostProfitablePath(self, edges: List[List[int]], bob: int, amount: List[int]) -> int:
 graph = defaultdict(list)
 for u,v in edges:
 graph[u].append(v)
 graph[v].append(u)
 n = len(graph)
 vis = set()
 parent = [0]*n
 dist = [0]*n
 def dfs(node,p,d):
 if node in vis:
 return
 vis.add(node)
 parent[node] = p
 dist[node] = d
 for c in graph[node]:
 dfs(c,node,d+1)
 dfs(0,-1,0)
 time = 0
```

```
while bob!=0:
if time<dist[bob]:
amount[bob] = 0
elif time == dist[bob]:
amount[bob]//=2
time+=1
bob = parent[bob]
p = set(parent)
self.ans = float('-inf')
visited = set()
def fun(node,x):
if node in visited:
return
if node not in p:
self.ans = max(self.ans,x)
return
visited.add(node)
for c in graph[node]:
fun(c,x+amount[c])
fun(0,amount[0])
return self.ans
```