

Amazon OA2

Cut off Rank Easy, Time: O(N)

```
def countLevelUpPlayers(cutoffRank, num, scores):
    d = {}
    for s in scores:
        if s not in d:
            d[s] = 0
            d[s] += 1

    buffer = []
    for s, num in d.items():
        hq.heappush(buffer, (-s, num))

    rank = 1
    count = 0
    while buffer:
        s, num = hq.heappop(buffer)
        if s == 0:
            break
        if rank <= cutoffRank:
            count += num
            rank += num
        else:
            break

    return count
```

```
def fill_the_truck(num, boxes, unitSize, unitsPerBox, truckSize):
    buffer = []

    for idx, num_box in enumerate(boxes):
        num_units = unitsPerBox[idx]
        for _ in range(num_box):
            hq.heappush(buffer, -1*num_units)

    print(buffer)
    count = 0
    for _ in range(truckSize):
        if len(buffer) > 0:
            count += hq.heappop(buffer)

    return count

print(fill_the_truck(3, [1,2,3], 3, [3,2,1], 3))
```

Fill The Truck EASY - Greedy

239. Sliding Window Maximum

You are given an array of integers `nums`, there is a sliding window of size `k` which is moving from the very left of the array to the very right. You can only see the `k` numbers in the window. Each time the sliding window moves right by one position.

Return the max sliding window.

Example 1:

Input: `nums = [1,3,-1,-3,5,3,6,7], k = 3`

Output: `[3,3,5,6,7]`

Explanation:

Window position	Max
[1, 3, -1] →	3
[3, -1, -3] →	3
[1, 3, -1] →	5
[1, 3, -1] →	5
[1, 3, -1] →	6
[1, 3, -1] →	7

Example 2:

Input: `nums = [1], k = 1`

Output: `[1]`

```
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        n = len(nums)
        ans = []
        def clear():
            while q and nums[q[-1]] < nums[i]:
                q.pop()
        q = deque()
        for i in range(n):
            clear()
            q.append(i)
            ans.append(nums[q[0]])
            if i - k + 1 < 0:
                continue
            if i - k + 1 == q[0]:
                q.popleft()
            ans.append(nums[q[0]])
        return ans
```

Disk Space Analysis

Subtree with Maximum Average

Easy lah

```
class Solution:
    def breakPalindrome(self, palindrome: str) -> str:
        if len(palindrome)<2:
            return ""
        lst=list(palindrome)
        for i in range((len(palindrome)//2)):
            if palindrome[i]!="a":
                lst[i]="a"
                return "".join(lst)
        lst[-1]="b"
        return "".join(lst)
```

Break a Palindrome

Smallest Negative Balance

Dict and then items() to get the smallest

Find The Highest Profit

Desc counter

Ways to Split String Into Prime Numbers

DP, get prime number first by another DP

Fetch Items to Display

Create

Packaging Automation

Greedy, recursive

Divide and conquer, cut it in half and calculate the minDist in two parts, and then calculate the middle part, pick points that has xDist and yDist shorter than current minDist

Min Distance Between Robots

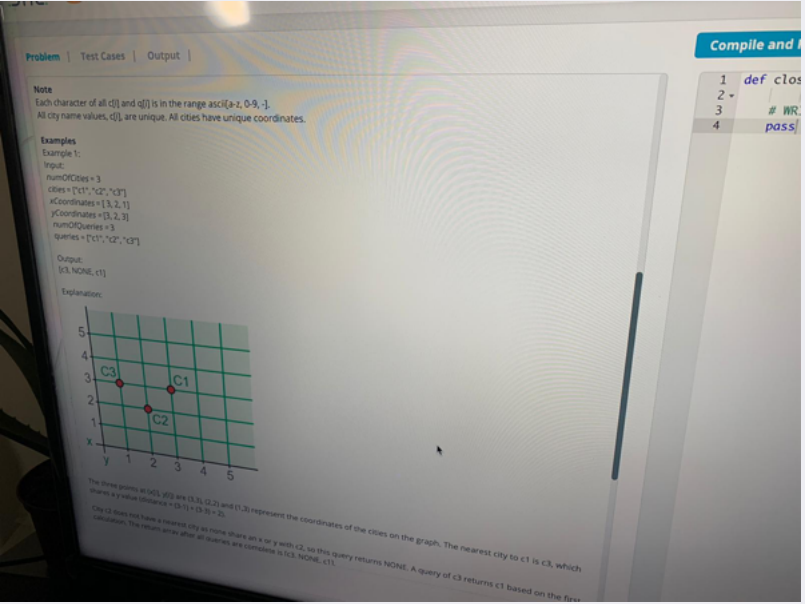
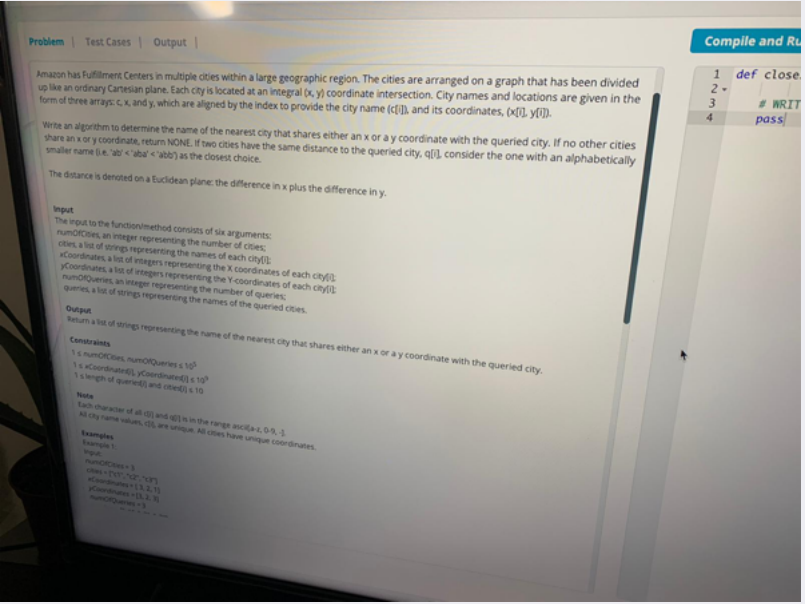
Union and find

Power Grid

Combination,

Count Teams

$${}_nC_r = \frac{n!}{r!(n-r)!}$$



```
def findNearestCities(numOfCities, cities, xCoordinates, yCoordinates, numQueries, queries):
    xd = {}
    yd = {}
    for idx, x in enumerate(xCoordinates):
        if x not in xd:
            xd[x] = []
            xd[x].append((yCoordinates[idx], idx))
    for idx, y in enumerate(yCoordinates):
        if y not in yd:
            yd[y] = []
            yd[y].append((xCoordinates[idx], idx))

    nameToIdx = {}
    for idx, city in enumerate(cities):
        nameToIdx[city] = idx
    ans = []
    for city in queries:
        res = None
        idx = nameToIdx[city]
        x, y = xCoordinates[idx], yCoordinates[idx]
        xNeighbors, yNeighbors = xd[x], yd[y]
        for xn in xNeighbors:
            xDist, xName = abs(xn[0] - x), cities[xn[1]]
            if xName == city:
                continue
            if not res or xDist < res[1] or (xDist==res[1] and xName<res[0]):
                res = (xName, xDist)
        for yn in yNeighbors:
            yDist, yName = abs(yn[0] - y), cities[yn[1]]
            if yName == city:
                continue
            if not res or yDist < res[1] or (yDist==res[1] and yName<res[0]):
                res = (yName, yDist)
        if res:
            ans.append(res[0])
        else:
            ans.append('NONE')
    return ans
```

Nearest City