GREEDY ALGORITHMS

(i) builds a solution step-by-step

(ii) series of step / instruction used to solve a problem

(iii) At each step → makes the choice that gives the highest immediate benefit

(iv) locally optimum choice hoping to find global optimum

(v) Used for optimisation problems.

(vi) A problem can have many feasible solution but only 1 optimal solution

(vii) Greedy is used for optimisation problems but does not give optimal solution always

QUESTION - FRACTIONAL KNAPSACK

Coding Exercise : Fractional Knapsack

Determine how to optimally fill a knapsack with a capacity of W kilograms using a list of N items, where each item is represented by a pair [profit, weight]. In the Fractional Knapsack problem, you can take fractions of items to maximize the total profit in the knapsack.(N will be greater than equal to 1)

Example 1:

Given arr[] = [[70, 10], [90, 20], [150, 30]]

W= 25

Expected output = 145

Example 2:

Given arr[] = [[70, 10], [90, 20], [150, 30]]

W= 45

Expected output = 242.5

CODE 7

```
def fractionalKnapsack(W, arr, n):
    arr.sort(reverse=True, key=lambda x: x[0]/x[1])

    remaning_weight = W

    value = 0

    for i in range(n):

        if remaning_weight == 0:
            break

        weight = min(remaining_weight, arr[i][0])

        remaining_weight -= weight

        value += arr[i][0]/arr[i][1] * weight

    return value
```

QUESTION : Non-Overlapping Intervals

Input : intervals = [[1,2] [2,3], [3,4], [1,3]]

Output : 1

Explanation : [1,3] can be removed and rest of the interval are non-overlapping

Input : [[1,4], [2,5], [5,6], [3,4]]

Output : 2

CODE 7

```
def eraseOverlappingIntervals(intervals):
    intervals.sort(key = lambda x : x[1])
    # sort basis on end time

    res = 0
    end = float('-inf')

    for start, finish in intervals:
        if start >= end :
            end = finish
            # non-overlapping interval

        else :
            res += 1

    return res
```

QUESTION : 55. Jump Game

Example 1:

> **Input:** nums = [2,3,1,1,4]
> **Output:** true
> **Explanation:** Jump 1 step from index 0 to 1, then 3 steps to the last index.

CODE

```
def canJump(nums):
    n = len(nums)
    max_index = 0
    for i in range(n):
        if i > max_index:
            return False
        max_index = max(max_index, i + nums[i])
        if max_index >= n-1
            return True.
```

QUESTION : Minimum number of arrow to burst balloons

Input: points = [[10,16],[2,8],[1,6],[7,12]]
Output: 2
Explanation: The balloons can be burst by 2 arrows:
- Shoot an arrow at x = 6, bursting the balloons [2,8] and [1,6].
- Shoot an arrow at x = 11, bursting the balloons [10,16] and [7,12].

Similar to interval overlapping problem we discussed above,

In this case we arrow += 1 if we find a non-overlapping interval

CODE:

```
def findMinArrowShots(points):
    points.sort(key = lambda x : x[1])
    arrow = 0
    end = float('-inf')
    for start, finish in points:
        if start > end:
            # non-overlapping interval
              found so increase arrow
            arrow += 1
            end = finish
    return arrow
```

QUESTION: Two city Scheduling

Input: costs = [[10,20],[30,200],[400,50],[30,20]]
Output: 110 A B
Explanation:
The first person goes to city A for a cost of 10.
The second person goes to city A for a cost of 30.
The third person goes to city B for a cost of 50.
The fourth person goes to city B for a cost of 20.

There are 2N person and only maximum of n person can travel to a city

CODE :

```
def twoCitySchedCost(costs):
    cost.sort(key = lambda x : x[0] - x[-1])
# this arranges the cost array in such a manner that all low cost of A are arranged before.
    total_cost = 0
    n = len(costs) // 2
# send first n people to 'A' city
    for i in range(n):
        total_cost = cost[i][0]
# now send rest to 'B'
    for i in range(n, 2*n):
        total_cost += costs[i][1]
```

```
return total_cost
```

QUESTION : Boats to save people.

Input : people = [1, 2] , limit = 3

Output : 1

Explanation - 1 boat (1,2)

```
def numRescueBoats(people, limit):
    people.sort()
    boats = 0

    lightest, heaviest = 0, len(people) - 1

    while lightest <= heaviest :
        boats += 1
        if people[lightest] + people[heaviest] <=
                                              limit

            lightest += 1
            # lightest person is included only
              if condition is met true

        heaviest -= 1  # only heaviest person
                         is assigned boat is
                         limit is excuded

    return boats
```

QUESTION - TASK SCHEDULER

Input : tasks = ['A', 'A', 'A', 'B', 'B', 'B'] , n = 2

Output : 8

Explanation :

A , B, idle, A, B, idle , A, B

	HF AAABC	HF HF AAABBBC #=2
parts (P)	HF-1 ↓ 3-1 = 2	HF-1 ↓ 3-1 = 2
slot in Parts (S)	P x n(given)	P x [n - (#-1)]
tasks to be placed (T)	5 - 3x1	7 - 3x2
empty slots/idle	S-T	S-T

return empty slots + len(tasks) if empty slots > 0
else len(tasks)

CODE :

```
def leastInterval(tasks, n):
    count = [0]*26
    max_freq = 0

    number_max_freq = 0

    for task in tasks:
```

```
        index = ord(task) - ord('A')
        count[index] += 1

        if max_freq < count[index]

                max_freq = count[index]
                number_max_freq = 1

        if max_freq == count[index]

                number_max_freq += 1

    parts = max_freq - 1

    slots_per_part = n - (number_max_freq - 1)

    total_slot_in_part = parts * slot_per_part

    task_remaining = len(task) - max_freq *
                                 number_max_freq

    idles = max(0, total_slot_in_part - task_
                                 remaining)

    return len(tasks) + idles
```

QUESTION : LARGEST NUMBER

Input : [2, 10]

Output : 210

Input : [10, 5, 9]
Output : 9510

CODE :

```
def largestNumber(nums):
    for i, n in enumerate(nums):
        nums[i] = str(n)

    def compare(n1, n2):
        if n1+n2 > n2+n1
            return -1
        else:
            1

    nums = sorted(nums, key = cmp_to_key(compare))
    return str(int("".join(nums)))
```

QUESTION : 134. Gas Station.

There are `n` gas stations along a circular route, where the amount of gas at the `i^th` station is `gas[i]`.

You have a car with an unlimited gas tank and it costs `cost[i]` of gas to travel from the `i^th` station to its next `(i + 1)^th` station. You begin the journey with an empty tank at one of the gas stations.

Given two integer arrays `gas` and `cost`, return *the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return* `-1`. If there exists a solution, it is **guaranteed** to be **unique**

Example 1:

> **Input:** gas = [1,2,3,4,5], cost = [3,4,5,1,2]
> **Output:** 3
> **Explanation:**
> Start at station 3 (index 3) and fill up with 4 unit of gas. Your tank = 0 + 4 = 4
> Travel to station 4. Your tank = 4 - 1 + 5 = 8
> Travel to station 0. Your tank = 8 - 2 + 1 = 7
> Travel to station 1. Your tank = 7 - 3 + 2 = 6

```
Travel to station 2. Your tank = 6 - 4 + 3 = 5
Travel to station 3. The cost is 5. Your gas is just enough to travel back to station 3.
Therefore, return 3 as the starting index.
```

CODE :

```
def canCompleteCircuit(gas, cost):
    if sum(gas) < sum(cost):
        return -1

    for i in range(len(gas)):
        total += gas[i] - cost[i]

        if total < 0:
            total = 0
            res = i + 1

    return res
```

QUESTION : JUMP GAME II

Input: nums = [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

CODE :

```
def jump(nums):

    jumps = 0
    current_end = 0
    farthest = 0
```

```
for i in range(n-1):  # we dont need to jump from the last element.

    #always try to jump to farthest position
    farthest = max(farthest, nums[i] + i)
    if i == current_end:
        jump += 1
        current_end = farthest
        if current_end >= n-1
            break
return jumps
```

ARRAY QUESTIONS

QUESTION : Rotate Array

Input : nums = [1, 2, 3, 4, 5, 6, 7]
Output : [5, 6, 7, 1, 2, 3, 4]

Reverse the num array

[7, 6, 5, 4, 3, 2, 1,]

Now reverse the 2 sub array formed

[5,6,7,1,2,3,4]

(1) Reverse (2) Reverse (0, k-1)

Reverse (k, length-1)

CODE:

```
def rotate(nums):
    if len(nums) == 0:
        return 0

    k = k % len(nums)

    temp = nums[-k:]  # create temp array of last k elements

    # now shift the front elment to last

    for i in reversed(range(0, len(nums)-k)):
        nums[i+k] = nums[i]

    # now append the 'temp' array at the beginning

    for i in range(len(temp)):
        nums[i] = temp[i]
```

2nd Way : More Code Intuitive

```
def rotate(nums):
    # define a reverse function to reverse the nums
    def reverse(arr, start, end):
        while start < end:
            arr[start], arr[end] = arr[end], arr[start]
    # reversing the whole array
    reverse(nums, 0, len(nums) - 1)
    # reversing the first k element
    reverse(nums, 0, k-1)
    # reversing the rest k element
    reverse(nums, k, len(nums) - 1)
```

QUESTION : ISOMORPHIC STRINGS

Input: s = 'egg' t = 'add'
Output: true

CODE :

```
def isIsomorphic(s,t):
```

```
if len(s) != len(t):
        return False

s_hash, t_hash = {}, {}

for i in range(len(s)):

    char_s = s[i]
    char_t = t[i]

    if char_s not in s_hash:

        s_hash[char_s] = char_t

    if char_t not in t_hash:

        t_hash[char_t] = char_s

    if s_hash[char_s] != char_t or
       t_hash[char_t] != char_s

        return False

return True
```

Question : 3. Longest Substring Without Repeating Character

Input : s = "abcabcbb"
Output : 3

Explanation : 'abc' = 3

CODE :

```
def max_unique_length(s):
    max_len = 0
    start = 0
    seen = {}
    for i in range(len(s)):
        char = s[i]
        if char in seen:
            start = max(start, seen[char]+1)
        max_len = (max_len, i - start +1)
        seen[char] = i
    return max_len
```

Question : 49. Group Anagrams

Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

CODE

```
def group_anagrams (strs):
```

```
    anagrams = {}
    for string in strs:
        sorted_string = ''.join(sorted(string))
        if sorted_string in anagrams:
            anagram[sorted_string] += [string]
        else:
            anagrams[sorted_string] = [string]
    return anagram.values()
```

Searching Algorithms Questions

33. Search in a rotated array

CODE:

```
def search_rotated_sorted_array(nums, target):
    #check if the array is empty
    if not nums:
```

```
    return -1
left, right = 0, len(nums) - 1
while left <= right:
    mid = (left + right) // 2
    if nums[mid] == target
        return mid
    # Check if the left half is sorted
    if nums[left] < nums[mid]:
        #Check if the target is in the left half
        if nums[left] <= target < nums[mid]:
            right = mid - 1 # check the left half
        else:
            left = mid + 1 # check the right half
    # The right half is sorted
    else:
        # if the target is in the right half
        if nums[mid] < target <= nums[right]:
            left = mid + 1 # check the right half
        else:
            right = mid - 1 # check the
```

left half

```
return -1
```

34. Find First and Last Position of Elements in Sorted Array

Input = nums = [5, 7, 7, 8, 8, 10]
target = 8
Output = [3, 4]

CODE:

```
def search_for_range(array, target):
    def search(nums, target, findstartIndex):
        ans = -1
        start = 0
        end = len(nums) - 1
        while start <= end:
            mid = start + (end-start)//2 #modified version for finding min
            if target < nums[mid]:
                end = mid - 1
            elif target > nums[mid]:
```

```
            start = mid + 1

        else:
            # target found
            ans = mid
            if findStartIndex:
                # if searching for start index
                  continue searching left
                end = mid - 1

            else:
                # if searching for end index
                  continue searching right
                start = mid + 1

        return ans

    # initialize res array with default values
    ans = [-1, -1]

    # find the start index of the target
    ans[0] = search(array, target, True)

    if ans[0] != -1:
        ans[1] = search(array, target, False)

    return ans
```

74. Search a 2D Matrix

Input : [[1,3,5,7],[10,11,16,20],[23,30,34,60]]
target = 3

Output : True

We will consider the 2D matrix as 1D and apply the binary search, with some modification

We will convert 1D Index to 2D index on fly.

CODE:

```
def searchMatrix(self, matrix, target):
    #check if the matrix is empty
    if not matrix or not matrix[0]:
        return False

    #Get the dimension of the matrix
    row : m   , column = n
    m, n = len(matrix), len(matrix[0])

    #initialize pointers for binary search
    #Treat the 2D matrix as a 1D sorted array
    left, right = 0, m*n - 1
    while left <= right:
        mid = (left+ right) //2

        #Formula to convert 2D matrix to 1D
        row, col = mid //n,  mid %n

        #Get the value of mid
```

```
value = matrix[row][col]

if value == target:
    return True

elif value < target:
    left = mid + 1

else:
    right = mid - 1

return False.
```