1. To Implement the Median of Medians algorithm ensures that you handle the worst-case time complexity efficiently while finding the k-th smallest element in an unsorted array. arr = [12, 3, 5, 7, 19] k = 2 Expected Output:5 arr = [12, 3, 5, 7, 4, 19, 26] k = 3 Expected Output:5 arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6 Expected Output:6

def partition(arr, pivot):

low = [x for x in arr if x < pivot]

high = [x for x in arr if x > pivot]

return low, pivot, high

def select\_pivot(arr):

if len(arr) <= 5:

return sorted(arr)[len(arr) // 2]

sublists = [arr[i:i+5] for i in range(0, len(arr), 5)]

medians = [sorted(sublist)[len(sublist) // 2] for sublist in sublists]

return median\_of\_medians(medians, len(medians) // 2)

def median\_of\_medians(arr, k):

pivot = select\_pivot(arr)

low, pivot\_value, high = partition(arr, pivot)

if k < len(low):

return median\_of\_medians(low, k)

elif k == len(low):

return pivot\_value

else:

return median\_of\_medians(high, k - len(low) - 1)

arr1 = [12, 3, 5, 7, 19]

k1 = 2

print(f"{k1}-th smallest element:", median\_of\_medians(arr1, k1 - 1))

arr2 = [12, 3, 5, 7, 4, 19, 26]

k2 = 3

print(f"{k2}-th smallest element:", median\_of\_medians(arr2, k2 - 1))

arr3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

k3 = 6

print(f"{k3}-th smallest element:", median\_of\_medians(arr3, k3 - 1))

2. To Implement a function median\_of\_medians(arr, k) that takes an unsorted array arr and an integer k, and returns the k-th smallest element in the array. arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6 arr = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27] k = 5 Output: An integer representing the k-th smallest element in the array.

def partition(arr, pivot):

low = [x for x in arr if x < pivot]

high = [x for x in arr if x > pivot]

return low, pivot, high

def select\_pivot(arr):

if len(arr) <= 5:

return sorted(arr)[len(arr) // 2]

sublists = [arr[i:i+5] for i in range(0, len(arr), 5)]

medians = [sorted(sublist)[len(sublist) // 2] for sublist in sublists]

return median\_of\_medians(medians, len(medians) // 2)

def median\_of\_medians(arr, k):

if len(arr) <= 5:

return sorted(arr)[k]

pivot = select\_pivot(arr)

low, pivot\_value, high = partition(arr, pivot)

if k < len(low):

return median\_of\_medians(low, k)

elif k == len(low):

return pivot\_value

else:

return median\_of\_medians(high, k - len(low) - 1)

arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

k1 = 6

print(f"The {k1}-th smallest element is:", median\_of\_medians(arr1, k1 - 1))

arr2 = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27]

k2 = 5

print(f"The {k2}-th smallest element is:", median\_of\_medians(arr2, k2 - 1))

3. Write a program to implement Meet in the Middle Technique. Given an array of integers and a target sum, find the subset whose sum is closest to the target. You will use the Meet in the Middle technique to efficiently find this subset. a) Set[] = {45, 34, 4, 12, 5, 2} Target Sum : 42 b) Set[]= {1, 3, 2, 7, 4, 6} Target sum = 10:

from itertools import combinations

def meet\_in\_the\_middle(arr, target):

n = len(arr)

left\_part = arr[:n//2]

right\_part = arr[n//2:]

left\_sums = {sum(combo): combo for r in range(len(left\_part) + 1) for combo in combinations(left\_part, r)}

right\_sums = {sum(combo): combo for r in range(len(right\_part) + 1) for combo in combinations(right\_part, r)}

closest\_sum = float('inf')

closest\_subset = []

for l\_sum, l\_combo in left\_sums.items():

for r\_sum, r\_combo in right\_sums.items():

current\_sum = l\_sum + r\_sum

if abs(current\_sum - target) < abs(closest\_sum - target):

closest\_sum = current\_sum

closest\_subset = l\_combo + r\_combo

return closest\_subset, closest\_sum

arr1 = [45, 34, 4, 12, 5, 2]

target1 = 42

print(f"Closest subset for target {target1}:", meet\_in\_the\_middle(arr1, target1))

arr2 = [1, 3, 2, 7, 4, 6]

target2 = 10

print(f"Closest subset for target {target2}:", meet\_in\_the\_middle(arr2, target2))

4. Write a program to implement Meet in the Middle Technique. Given a large array of integers and an exact sum E, determine if there is any subset that sums exactly to E. Utilize the Meet in the Middle technique to handle the potentially large size of the array. Return true if there is a subset that sums exactly to E, otherwise return false. a) E = {1, 3, 9, 2, 7, 12} exact Sum = 15 b) E = {3, 34, 4, 12, 5, 2} exact Sum = 15

from itertools import combinations

def meet\_in\_the\_middle\_exact\_sum(arr, exact\_sum):

n = len(arr)

left\_part = arr[:n//2]

right\_part = arr[n//2:]

left\_sums = {sum(combo) for r in range(len(left\_part) + 1) for combo in combinations(left\_part, r)}

right\_sums = {sum(combo) for r in range(len(right\_part) + 1) for combo in combinations(right\_part, r)}

for l\_sum in left\_sums:

if exact\_sum - l\_sum in right\_sums:

return True

return False

arr1 = [1, 3, 9, 2, 7, 12]

exact\_sum1 = 15

print(f"Exact subset sum exists for {exact\_sum1}: {meet\_in\_the\_middle\_exact\_sum(arr1, exact\_sum1)}")

arr2 = [3, 34, 4, 12, 5, 2]

exact\_sum2 = 15

print(f"Exact subset sum exists for {exact\_sum2}: {meet\_in\_the\_middle\_exact\_sum(arr2, exact\_sum2)}")

5. Given two 2×2 Matrices A and B A=(1 7 B=( 1 3 3 5) 7 5) Use Strassen's matrix multiplication algorithm to compute the product matrix C such that C=A×B. Test Cases: Consider the following matrices for testing your implementation: Test Case 1: A=(1 7 B=( 6 8 3 5), 4 2) Expected Output: C=(18 14 62 66)

import numpy as np

def strassen\_multiply(A, B):

if len(A) == 2:

P1 = (A[0][0] + A[1][1]) \* (B[0][0] + B[1][1])

P2 = (A[1][0] + A[1][1]) \* B[0][0]

P3 = A[0][0] \* (B[0][1] - B[1][1])

P4 = A[1][1] \* (B[1][0] - B[0][0])

P5 = (A[0][0] + A[0][1]) \* B[1][1]

P6 = (A[1][0] - A[0][0]) \* (B[0][0] + B[0][1])

P7 = (A[0][1] - A[1][1]) \* (B[1][0] + B[1][1])

C11 = P1 + P4 - P5 + P7

C12 = P3 + P5

C21 = P2 + P4

C22 = P1 + P3 - P2 + P6

return np.array([[C11, C12], [C21, C22]])

A = [[1, 7], [3, 5]]

B = [[6, 8], [4, 2]]

C = strassen\_multiply(A, B)

print("Product matrix:\n", C)

6. Given two integers X=1234 and Y=5678: Use the Karatsuba algorithm to compute the product Z=X x Y Test Case 1: Input: x=1234,y=5678 Expected Output: z=1234×5678=7016652

def karatsuba(x, y):

if x < 10 or y < 10:

return x \* y

n = max(len(str(x)), len(str(y)))

m = n // 2

high1, low1 = divmod(x, 10\*\*m)

high2, low2 = divmod(y, 10\*\*m)

z0 = karatsuba(low1, low2)

z1 = karatsuba((low1 + high1), (low2 + high2))

z2 = karatsuba(high1, high2)

return (z2 \* 10\*\*(2\*m)) + ((z1 - z2 - z0) \* 10\*\*m) + z0

x = 1234

y = 5678

result = karatsuba(x, y)

print(f"Product of {x} and {y} using Karatsuba: {result}")