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

def partition(arr, low, high, pivot):

for i in range(low, high):

if arr[i] == pivot:

arr[i], arr[high] = arr[high], arr[i]

break

pivot\_index = low

for i in range(low, high):

if arr[i] < pivot:

arr[i], arr[pivot\_index] = arr[pivot\_index], arr[i]

pivot\_index += 1

arr[pivot\_index], arr[high] = arr[high], arr[pivot\_index]

return pivot\_index

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

if low == high:

return arr[low]

medians = []

i = low

while i <= high:

sub\_right = min(i + 4, high)

sublist = sorted(arr[i:sub\_right + 1])

medians.append(sublist[len(sublist) // 2])

i += 5

if len(medians) == 1:

pivot = medians[0]

else:

pivot = median\_of\_medians(medians, 0, len(medians) - 1, len(medians) // 2)

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

if k == pivot\_index:

return arr[k]

elif k < pivot\_index:

return median\_of\_medians(arr, low, pivot\_index - 1, k)

else:

return median\_of\_medians(arr, pivot\_index + 1, high, k)

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

k = 2

result = median\_of\_medians(arr, 0, len(arr) - 1, k)

print(f"The {k+1}-th smallest element is {result}")

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

def partition(arr, pivot):

low = []

high = []

for num in arr:

if num < pivot:

low.append(num)

elif num > pivot:

high.append(num)

return low, pivot, high

def select\_pivot(arr):

# Split arr into sublists of 5, sort each sublist, and pick the median

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

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

if len(medians) <= 5:

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

else:

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

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

if len(arr) == 1:

return arr[0]

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

else:

return pivot\_value

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

k1 = 6

print(median\_of\_medians(arr1, k1 - 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. Set[] = {45, 34, 4, 12, 5, 2} .Target Sum : 42

import itertools

import bisect

def generate\_subset\_sums(arr):

all\_sums = []

# Generate all subsets using itertools

for r in range(len(arr) + 1):

for subset in itertools.combinations(arr, r):

all\_sums.append(sum(subset))

return all\_sums

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

n = len(arr)

first\_half = arr[:n//2]

second\_half = arr[n//2:]

sums\_first\_half = generate\_subset\_sums(first\_half)

sums\_second\_half = generate\_subset\_sums(second\_half)

sums\_second\_half.sort()

closest\_sum = float('inf')

for sum\_first in sums\_first\_half:

required = target - sum\_first

idx = bisect.bisect\_left(sums\_second\_half, required)

if idx < len(sums\_second\_half):

current\_sum = sum\_first + sums\_second\_half[idx]

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

closest\_sum = current\_sum

if idx > 0:

current\_sum = sum\_first + sums\_second\_half[idx - 1]

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

closest\_sum = current\_sum

return closest\_sum

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

target\_sum = 42

result = meet\_in\_the\_middle(arr, target\_sum)

print(f"The closest sum to the target {target\_sum} is {result}")

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.

from itertools import combinations

def get\_subset\_sums(arr):

subset\_sums = set()

n = len(arr)

for r in range(n+1):

for subset in combinations(arr, r):

subset\_sums.add(sum(subset))

return subset\_sums

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

mid = len(arr) // 2

left\_part = arr[:mid]

right\_part = arr[mid:]

left\_sums = get\_subset\_sums(left\_part)

right\_sums = get\_subset\_sums(right\_part)

for left\_sum in left\_sums:

if (exact\_sum - left\_sum) in right\_sums:

return True

return False

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

exact\_sum = 15

result = meet\_in\_the\_middle(arr, exact\_sum)

print(result)

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)

def strassen\_multiply(A, B):

# Extract elements from matrix A

a11, a12 = A[0][0], A[0][1]

a21, a22 = A[1][0], A[1][1]

b11, b12 = B[0][0], B[0][1]

b21, b22 = B[1][0], B[1][1]

P1 = (a11 + a22) \* (b11 + b22)

P2 = (a21 + a22) \* b11

P3 = a11 \* (b12 - b22)

P4 = a22 \* (b21 - b11)

P5 = (a11 + a12) \* b22

P6 = (a21 - a11) \* (b11 + b12)

P7 = (a12 - a22) \* (b21 + b22)

C11 = P1 + P4 - P5 + P7

C12 = P3 + P5

C21 = P2 + P4

C22 = P1 + P3 - P2 + P6

C = [[C11, C12], [C21, C22]]

return C

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

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

C = strassen\_multiply(A, B)

print("Matrix C (A x B):")

for row in C:

print(row)

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):

# Base case for recursion: when the numbers are small enough

if x < 10 or y < 10:

return x \* y

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

half = n // 2

high\_x = x // 10\*\*half

low\_x = x % 10\*\*half

high\_y = y // 10\*\*half

low\_y = y % 10\*\*half

z0 = karatsuba(low\_x, low\_y) # low\_x \* low\_y

z1 = karatsuba((low\_x + high\_x), (low\_y + high\_y)) # (low\_x + high\_x) \* (low\_y + high\_y)

z2 = karatsuba(high\_x, high\_y) # high\_x \* high\_y

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

x = 1234

y = 5678

result = karatsuba(x, y)

print(f"The product of {x} and {y} using Karatsuba is: {result}")