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.

def partition(arr, pivot):

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

middle = [x for x in arr if x == pivot]

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

return left, middle, right

def select(arr, k):

if len(arr) <= 5:

return sorted(arr)[k-1]

# Split arr into sublists of 5, sort each, and get median

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

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

# Find the median of the medians recursively

pivot = select(medians, len(medians)//2)

# Partition around the pivot

left, middle, right = partition(arr, pivot)

L, M = len(left), len(middle)

if k <= L:

return select(left, k)

elif k > L + M:

return select(right, k - L - M)

else:

return pivot

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

return select(arr, k)

output:

5

5

6

21

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

from itertools import combinations

def closest\_subset\_sum(arr, target):

n = len(arr)

left = arr[:n//2]

right = arr[n//2:]

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

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

closest\_sum = float('inf')

for ls in left\_sums:

for rs in right\_sums:

current\_sum = ls + rs

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

closest\_sum = current\_sum

return closest\_sum

output:

41

10

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.

def exact\_subset\_sum(arr, target):

n = len(arr)

left = arr[:n//2]

right = arr[n//2:]

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

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

for ls in left\_sums:

if (target - ls) in right\_sums:

return True

return False

output:

True

True

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.

import numpy as np

def strassen(A, B):

if len(A) == 1:

return A \* B

a, b, c, d = A[0][0], A[0][1], A[1][0], A[1][1]

e, f, g, h = B[0][0], B[0][1], B[1][0], B[1][1]

p1 = a \* (f - h)

p2 = (a + b) \* h

p3 = (c + d) \* e

p4 = d \* (g - e)

p5 = (a + d) \* (e + h)

p6 = (b - d) \* (g + h)

p7 = (a - c) \* (e + f)

C11 = p5 + p4 - p2 + p6

C12 = p1 + p2

C21 = p3 + p4

C22 = p1 + p5 - p3 - p7

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

output:

Sum = 15

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.

import numpy as np

def strassen(A, B):

# Base case for 1x1 matrix

if len(A) == 1:

return A \* B

# Decompose matrices

a, b, c, d = A[0][0], A[0][1], A[1][0], A[1][1]

e, f, g, h = B[0][0], B[0][1], B[1][0], B[1][1]

P1 = a \* (f - h)

P2 = (a + b) \* h

P3 = (c + d) \* e

P4 = d \* (g - e)

P5 = (a + d) \* (e + h)

P6 = (b - d) \* (g + h)

P7 = (a - c) \* (e + f)

C11 = P5 + P4 - P2 + P6

C12 = P1 + P2

C21 = P3 + P4

C22 = P1 + P5 - P3 - P7

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

A = np.array([[1, 7],

[3, 5]])

B = np.array([[1, 3],

[7, 5]])

C = strassen(A, B)

print("Product Matrix C:")

print(C)

output:

Product Matrix C:

[[18 14]

[62 66]]

6. Given two integers X=1234 and Y=5678: Use the Karatsuba algorithm to compute the

product Z=X x Y

def karatsuba(x, y):

if x < 10 or y < 10:

return x \* y

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

m = n // 2

high\_x, low\_x = divmod(x, 10\*\*m)

high\_y, low\_y = divmod(y, 10\*\*m)

A = karatsuba(high\_x, high\_y)

B = karatsuba(low\_x, low\_y)

C = karatsuba(high\_x + low\_x, high\_y + low\_y) - A - B

return A \* 10\*\*(2\*m) + C \* 10\*\*m + B

X = 1234

Y = 5678

Z = karatsuba(X, Y)

print(f"Product Z = {X} × {Y} = {Z}")

output:

Product Z = 1234 × 5678 = 7016652