DAY-6 PROGRAMS

1. To implement the median of medians algorithm ensure that you handle the worst case time complexity efficiency while finding the k-th smallest element in an unsorted array.

Arr=[12,3,5,7,19] k=2

Expected output:5

Code:

import random

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

if len(arr) == 1:

return arr[0]

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

medians = [sorted(subarray)[len(subarray)//2] for subarray in subarrays]

if len(medians) <= 5:

pivot = sorted(medians)[len(medians)//2]

else:

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

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]

if k <= len(left):

return median\_of\_medians(left, k)

elif k <= len(left) + len(middle):

return middle[0]

else:

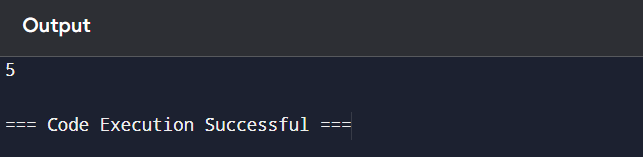
return median\_of\_medians(right, k - len(left) - len(middle))

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

k = 2

print(median\_of\_medians(arr, k))

output:



2. To implement a function median of medians (arr,k)that takes an unsorted array arr and an integer,and return the k-th smallest element in the array.

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

Output:An intrger representing the k-th smallest elements in the array.

Code:

import random

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

if len(arr) == 1:

return arr[0]

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

medians = [sorted(subarray)[len(subarray)//2] for subarray in subarrays]

if len(medians) <= 5:

pivot = sorted(medians)[len(medians)//2]

else:

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

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]

if k <= len(left):

return median\_of\_medians(left, k)

elif k <= len(left) + len(middle):

return middle[0]

else:

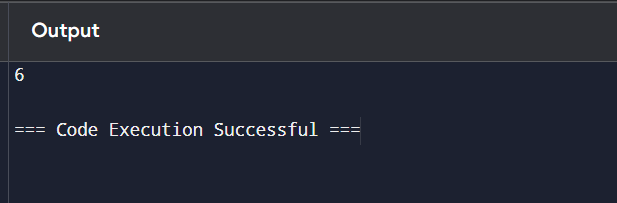
return median\_of\_medians(right, k - len(left) - len(middle))

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

k = 6

print(median\_of\_medians(arr, k))

output:



3. write a program to implement meet in the middle technique.give 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 eifficiency find this subset.

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

Target sum=42

Code:

from itertools import combinations

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

n = len(arr)

mid = n // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

def get\_subset\_sums(arr):

subset\_sums = set()

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

for combo in combinations(arr, r):

subset\_sums.add(sum(combo))

return sorted(subset\_sums)

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

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

closest\_sum = None

closest\_diff = float('inf')

for left\_sum in left\_sums:

target\_right\_sum = target - left\_sum

low, high = 0, len(right\_sums) - 1

while low <= high:

mid = (low + high) // 2

right\_sum = right\_sums[mid]

current\_sum = left\_sum + right\_sum

current\_diff = abs(current\_sum - target)

if current\_diff < closest\_diff:

closest\_diff = current\_diff

closest\_sum = current\_sum

if current\_sum < target:

low = mid + 1

else:

high = mid - 1

return closest\_sum

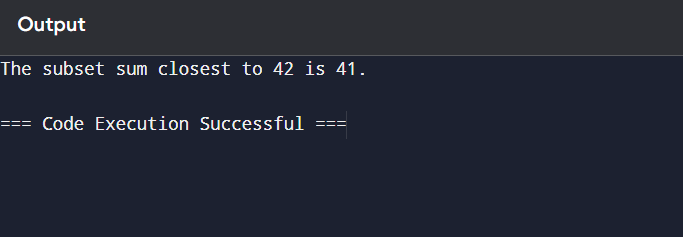
arr = [45, 34, 4, 12, 5, 2]

target = 42

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

print(f"The subset sum closest to {target} is {closest\_sum}.")

output:



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. E = {1, 3, 9, 2, 7, 12}

exact Sum = 15

code:

from itertools import combinations

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

exact\_sum = 15

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

left\_sums = set()

for i in range(len(left\_half) + 1):

for combo in combinations(left\_half, i):

left\_sums.add(sum(combo))

right\_sums = set()

for i in range(len(right\_half) + 1):

for combo in combinations(right\_half, i):

right\_sums.add(sum(combo))

found = False

for left\_sum in left\_sums:

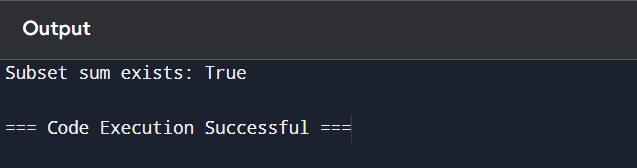
if (exact\_sum - left\_sum) in right\_sums:

found = True

break

print("Subset sum exists:", found)

output:



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=(13

3 5) 17)

Expected Output: C=(18 14

62 66)

Code:

def strassen\_matrix\_multiply(A, B):

if len(A) == 2:

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)

C = [[p5 + p4 - p2 + p6, p1 + p2], [p3 + p4, p1 + p5 - p3 - p7]]

return C

else:

return "Input matrices are not 2x2."

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

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

C = strassen\_matrix\_multiply(A, B)

print(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

Code:

def karatsuba\_multiplication(x, y):

x\_str = str(x)

y\_str = str(y)

n = max(len(x\_str), len(y\_str))

x1 = int(x\_str[:-n//2])

x0 = int(x\_str[-n//2:])

y1 = int(y\_str[:-n//2])

y0 = int(y\_str[-n//2:])

p1 = x1 \* y1

p2 = x0 \* y0

p3 = (x1 + x0) \* (y1 + y0)

z = p1 \* 10\*\*(n) + (p3 - p1 - p2) \* 10\*\*(n//2) + p2

return z

k=karatsuba\_multiplication(1234,5678)

print(k)

output:

