DAY-6

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

CODE:

```
arr = [12, 3, 5, 7, 19]
k = 2  # Looking for the 2nd smallest element
left, right = 0, len(arr) - 1
k_index = k - 1  # Adjust for zero-based indexing
while True:
    medians = []
    for i in range(left, right + 1, 5):
        # Create a subarray of at most 5 elements
        subarr = arr[i:min(i + 5, right + 1)]
        subarr.sort()          medians.append(subarr[len(subarr) // 2])
    if len(medians) == 1:
        median_of_medians = medians[0]
    else:
        medians.sort()
        median_of_medians = medians[len(medians) // 2]  # Get the median
    pivot_index = arr.index(median_of_medians)
    arr[pivot_index], arr[right] = arr[right], arr[pivot_index]  # Move pivot to end
    pivot_index = left  # Reset pivot index for partitioning
for j in range(left, right):
        if arr[j] < median_of_medians:
            arr[pivot_index], arr[j] = arr[j], arr[pivot_index]
            pivot_index += 1
arr[pivot_index], arr[right] = arr[right], arr[pivot_index]  # Move pivot to its final place
    if pivot_index == k_index:
        result = arr[pivot_index]  # Found the k-th smallest element
        break
    elif pivot_index > k_index:
        right = pivot_index - 1  # Search in the left partition
    else:
        left = pivot_index + 1  # Search in the right partition
print("The {}-th smallest element is: {}".format(k, result))
```

OUTPUT:

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

k = 2

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

CODE:

```
def partition(arr, low, high, pivot):
    # Partition the array around the pivot element
    pivot_index = arr.index(pivot)
    arr[pivot_index], arr[high] = arr[high], arr[pivot_index]
    i = low
    for j in range(low, high):
        if arr[j] < pivot:
            arr[i], arr[j] = arr[j], arr[i]
            i += 1
    arr[i], arr[high] = arr[high], arr[i]
    return i
def find_median(arr):
    arr.sort()
    return arr[len(arr) // 2]
def select(arr, left, right, k):
    if right - left + 1 <= 5:
        sublist = arr[left:right + 1]
        sublist.sort()
        return sublist[k]
    medians = []
    for i in range(left, right + 1, 5):
        sub_right = min(i + 4, right)
        medians.append(find_median(arr[i:sub_right + 1]))
    median_of_medians = select(medians, 0, len(medians) - 1, len(medians) // 2)
    pivot_index = partition(arr, left, right, median_of_medians)
    if pivot_index == k:
        return arr[pivot_index]
    elif pivot_index > k:
        return select(arr, left, pivot_index - 1, k)
    else:
        return select(arr, pivot_index + 1, right, k)
def kth_smallest(arr, k):
    return select(arr, 0, len(arr) - 1, k - 1)
arr = [12, 3, 5, 7, 19]
k = 2
result = kth_smallest(arr, k)
print("The {}-th smallest element is: {}".format(k, result))
```

OUTPUT:

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

k = 2

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

<u>CODE:</u>

```
from itertools import combinations
set_values = [45, 34, 4, 12, 5, 2]
target_sum = 42
n = len(set_values)
mid = n // 2
first_half = set_values[:mid]
second_half = set_values[mid:]
def generate_sums(arr):
    sums = set()
    for r in range(len(arr) + 1):  # +1 to include empty subset
        for combo in combinations(arr, r):
            sums.add(sum(combo))
    return sums
sums_first_half = generate_sums(first_half)
sums_second_half = generate_sums(second_half)
sums_second_half = sorted(sums_second_half)
closest_sum = None
closest_diff = float('inf')
for sum1 in sums_first_half:
    # Required sum from the second half
    required = target_sum - sum1
        low, high = 0, len(sums_second_half) - 1
    while low <= high:
        mid = (low + high) // 2
        if sums_second_half[mid] < required:
            low = mid + 1
        else:
            high = mid - 1
                for candidate in (sums_second_half[low-1] if low > 0 else None, sums_second_half[low] if
low < len(sums_second_half) else None):
        if candidate is not None:
            current_sum = sum1 + candidate
            current_diff = abs(target_sum - current_sum)
            if current_diff < closest_diff:
                closest_diff = current_diff
                closest_sum = current_sum
print("The closest sum to the target {} is: {}".format(target_sum, closest_sum))
```

OUTPUT:

The closest sum to the target 42 is: 42

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

CODE:

```
from itertools import combinations
set_values = [45, 34, 4, 12, 5, 2]
target_sum = 42
n = len(set_values)
mid = n // 2
first_half = set_values[:mid]
second_half = set_values[mid:]
def generate_sums(arr):
    sums = set()
    for r in range(len(arr) + 1):  # +1 to include empty subset
        for combo in combinations(arr, r):
            sums.add(sum(combo))
    return sums
sums_first_half = generate_sums(first_half)
sums_second_half = generate_sums(second_half)
sums_second_half = sorted(sums_second_half)
closest_sum = None
closest_diff = float('inf')
for sum1 in sums_first_half:
    required = target_sum - sum1
        low, high = 0, len(sums_second_half) - 1
    while low <= high:
        mid = (low + high) // 2
        if sums_second_half[mid] < required:
            low = mid + 1
        else:
            high = mid - 1
                for candidate in (sums_second_half[low-1] if low > 0 else None, sums_second_half[low] if
low < len(sums_second_half) else None):
        if candidate is not None:
            current_sum = sum1 + candidate
            current_diff = abs(target_sum - current_sum)
            if current_diff < closest_diff:
                closest_diff = current_diff
                closest_sum = current_sum
print("The closest sum to the target {} is: {}".format(target_sum, closest_sum))
```

OUTPUT:

True: A subset that sums exactly to 15 exists.

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
35 , 42)

<u>CODE:</u>

```
import numpy as np

def strassen_multiply(A, B):

    if len(A) == 2 and len(B) == 2

        C = np.zeros((2, 2))

        C[0][0] = A[0][0] * B[0][0] + A[0][1] * B[1][0]  # C11

        C[0][1] = A[0][0] * B[0][1] + A[0][1] * B[1][1]  # C12

        C[1][0] = A[1][0] * B[0][0] + A[1][1] * B[1][0]  # C21

        C[1][1] = A[1][0] * B[0][1] + A[1][1] * B[1][1]  # C22

         return C

 A = np.array([[1, 7], [3, 5]])

B = np.array([[6, 8], [4, 2]])

C = strassen_multiply(A, B)

print("Product Matrix C:\n", C)
```

OUTPUT:

```
Product Matrix C:

 [[34. 62.]

 [38. 46.]]
```

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

<u>CODE:</u>

```
def karatsuba(x, y):
    # Base case for recursion
    if x < 10 or y < 10:
        return x * y
    m = min(len(str(x)), len(str(y)))
    half_m = m // 2
    a = x // 10**half_m
    b = x % 10**half_m
    c = y // 10**half_m
    d = y % 10**half_m
    ac = karatsuba(a, c)
    bd = karatsuba(b, d)
    abcd = karatsuba(a + b, c + d)
    return ac * 10**(2 * half_m) + (abcd - ac - bd) * 10**half_m + bd
x = 1234
y = 5678
result = karatsuba(x, y)
print("Product Z =", result)
```

OUTPUT:

7016652