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.

main.py

Share Run

```python
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 = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i+1], arr[high] = arr[high], arr[i+1]
    return i + 1

def find_median(arr):
    arr.sort()
    return arr[len(arr) // 2]

def median_of_medians(arr, k):
    if len(arr) <= 5:
        arr.sort()
        return arr[k-1]

    medians = []
    for i in range(0, len(arr), 5):
        group = arr[i:i+5]
        medians.append(find_median(group))

    median_of_medians_pivot = median_of_medians(medians, len(medians)//2 + 1)
    pivot_index = partition(arr, 0, len(arr) - 1, median_of_medians_pivot)

    if pivot_index == k - 1:
        return arr[pivot_index]
    elif pivot_index > k - 1:
        return median_of_medians(arr[:pivot_index], k)
    else:
        return median_of_medians(arr[pivot_index + 1:], k - pivot_index - 1)

arr1 = [12, 3, 5, 7, 19]
k1 = 2
print("Expected Output:", median_of_medians(arr1, k1))

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

Output

```
Expected Output: 5
Expected Output: 5
Expected Output: 6

=== Code Execution Successful ===
```

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.

main.py

```python
def partition(arr, low, high, pivot_index):
    pivot_value = arr[pivot_index]
    arr[pivot_index], arr[high] = arr[high], arr[pivot_index]
    store_index = low
    for i in range(low, high):
        if arr[i] < pivot_value:
            arr[store_index], arr[i] = arr[i], arr[store_index]
            store_index += 1
    arr[store_index], arr[high] = arr[high], arr[store_index]
    return store_index
def median_of_medians(arr, k):
    if len(arr) == 1:
        return arr[0]
    n = len(arr)
    sublists = [arr[i:i+5] for i in range(0, n, 5)]
    medians = [sorted(sublist)[len(sublist)//2] for sublist in sublists]
    if len(medians) <= 5:
        pivot = sorted(medians)[len(medians)//2]
    else:
        pivot = median_of_medians(medians, len(medians)//2)
    pivot_index = arr.index(pivot)
    pivot_index = partition(arr, 0, len(arr) - 1, pivot_index)
    if k == pivot_index + 1:
        return arr[pivot_index]
    elif k < pivot_index + 1:
        return median_of_medians(arr[:pivot_index], k)
    else:
        return median_of_medians(arr[pivot_index+1:], k - pivot_index - 1)
arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k1 = 6
print(median_of_medians(arr1, k1))

arr2 = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27]
k2 = 5
print(median_of_medians(arr2, k2))
```

Output

```
6
21

=== Code Execution Successful =
```

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:

```python
from itertools import combinations

def get_subsets(arr):
    """Generate all possible subsets of the given array."""
    subsets = []
    n = len(arr)
    for r in range(n+1):
        for combo in combinations(arr, r):
            subsets.append(combo)
    return subsets

def meet_in_the_middle(arr, target_sum):
    """Find the subset whose sum is closest to the target sum using the Meet in the Middle technique."""
    n = len(arr)
    left_half = arr[:n//2]
    right_half = arr[n//2:]
    left_subsets = get_subsets(left_half)
    right_subsets = get_subsets(right_half)
    left_sums = {sum(subset): subset for subset in left_subsets}
    right_sums = {sum(subset): subset for subset in right_subsets}
    sorted_left_sums = sorted(left_sums.keys())
    sorted_right_sums = sorted(right_sums.keys())
    closest_sum = float('inf')
    best_subset = ()
    i, j = 0, len(sorted_right_sums) - 1
    while i < len(sorted_left_sums) and j >= 0:
        current_sum = sorted_left_sums[i] + sorted_right_sums[j]

        if abs(target_sum - current_sum) < abs(target_sum - closest_sum):
            closest_sum = current_sum
            best_subset = left_sums[sorted_left_sums[i]] + right_sums[sorted_right_sums[j]]
        if current_sum < target_sum:
            i += 1
        else:
            j -= 1
    return best_subset, closest_sum
arr1 = [45, 34, 4, 12, 5, 2]
target1 = 42
subset1, closest_sum1 = meet_in_the_middle(arr1, target1)
print(f"Subset: {subset1}, Closest Sum: {closest_sum1}")

arr2 = [1, 3, 2, 7, 4, 6]
target2 = 10
```

Output

```
Subset: (34, 5, 2), Closest Sum: 41
Subset: (4, 6), Closest Sum: 10

=== Code Execution Successful ===
```

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.

```python
from itertools import combinations
def get_all_subset_sums(arr):
    """Generate all possible subset sums of the given array."""
    subset_sums = set()
    n = len(arr)
    for r in range(n+1):
        for combo in combinations(arr, r):
            subset_sums.add(sum(combo))
    return subset_sums

def meet_in_the_middle(arr, target_sum):
    """Determine if there exists any subset that sums exactly to the target sum using Meet in the Middle technique."""
    n = len(arr)
    left_half = arr[:n//2]
    right_half = arr[n//2:]
    left_sums = get_all_subset_sums(left_half)
    right_sums = get_all_subset_sums(right_half)
    if target_sum in left_sums:
        return True
    if target_sum in right_sums:
        return True
    for left_sum in left_sums:
        if (target_sum - left_sum) in right_sums:
            return True
    return False

arr1 = [1, 3, 9, 2, 7, 12]
target_sum1 = 15
print(meet_in_the_middle(arr1, target_sum1))

arr2 = [3, 34, 4, 12, 5, 2]
target_sum2 = 15
print(meet_in_the_middle(arr2, target_sum2))

arr3 = [3, 34, 4, 12, 5, 2]
target_sum3 = 100
print(meet_in_the_middle(arr3, target_sum3))
```

Output

```
True
True
False

=== Code Execution Successful ===
```

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.

```python
def strassen_multiplication(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]
    M1 = (a + d) * (e + h)
    M2 = (c + d) * e
    M3 = a * (f - h)
    M4 = d * (g - e)
    M5 = (a + b) * h
    M6 = (c - a) * (e + f)
    M7 = (b - d) * (g + h)
    p = M1 + M4 - M5 + M7
    q = M3 + M5
    r = M2 + M4
    s = M1 - M2 + M3 + M6
    C = [[p, q], [r, s]]
    return C

A = [[1, 7], [3, 5]]
B = [[6, 8], [4, 2]]
C = strassen_multiplication(A, B)
print("Resultant Matrix C:")
for row in C:
    print(row)
```

Output

```
Resultant Matrix C:
[34, 22]
[38, 34]

=== Code Execution Successful ===
```

6. 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.

main.py

```python
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
z = karatsuba(x, y)
print(f"{x} × {y} = {z}")
```

Output

```
1234 × 5678 = 7006652

=== Code Execution Successful ===
```