

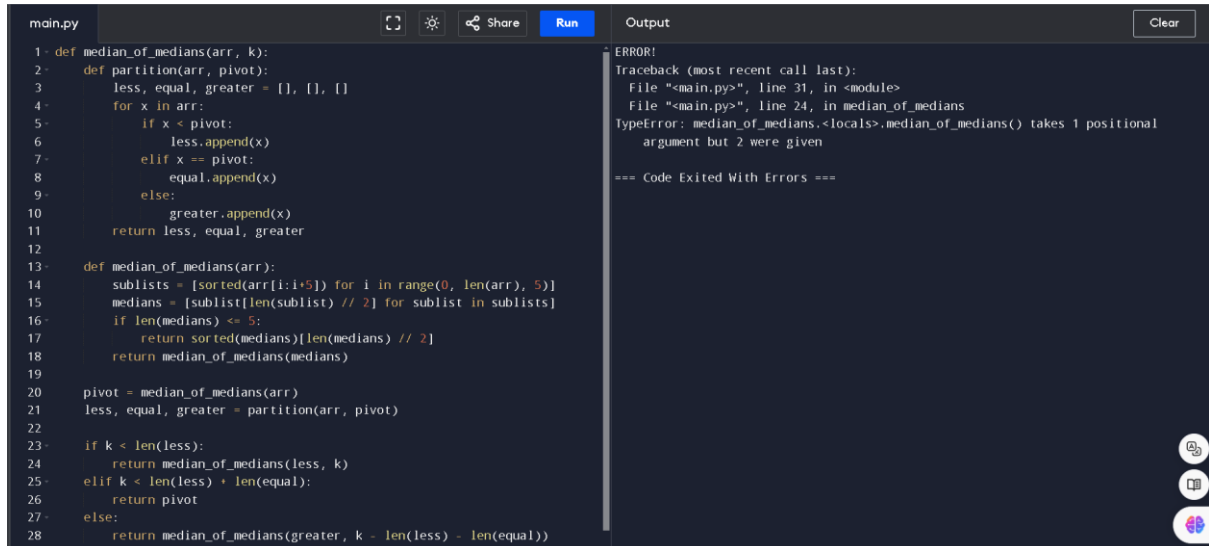
## DAY 6 LAB EXPERIMENTS

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  Run  Output  Clear
1 def median_of_medians(arr, k):
2     def partition(arr, pivot):
3         less, equal, greater = [], [], []
4         for x in arr:
5             if x < pivot:
6                 less.append(x)
7             elif x == pivot:
8                 equal.append(x)
9             else:
10                greater.append(x)
11            return less, equal, greater
12
13    def median_of_medians(arr):
14        sublists = [sorted(arr[i:i+5]) for i in range(0, len(arr), 5)]
15        medians = [sublist[len(sublist) // 2] for sublist in sublists]
16        if len(medians) <= 5:
17            return sorted(medians)[len(medians) // 2]
18        return median_of_medians(medians)
19
20    pivot = median_of_medians(arr)
21    less, equal, greater = partition(arr, pivot)
22
23    if k < len(less):
24        return median_of_medians(less, k)
25    elif k < len(less) + len(equal):
26        return pivot
27    else:
28        return median_of_medians(greater, k - len(less) - len(equal))
29
```

ERROR!  
Traceback (most recent call last):  
File "<main.py>", line 31, in <module>  
File "<main.py>", line 24, in median\_of\_medians  
TypeError: median\_of\_medians.<locals>.median\_of\_medians() takes 1 positional argument but 2 were given  
=== Code Exited With Errors ===

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.



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

```
main.py  Run  Output  Clear
1 def meet_in_the_middle_exact(arr, exact_sum):
2     def all_subset_sums(subset):
3         sums = set()
4         for r in range(len(subset) + 1):
5             for combo in combinations(subset, r):
6                 sums.add(sum(combo))
7         return sums
8
9     mid = len(arr) // 2
10    left_part = arr[:mid]
11    right_part = arr[mid:]
12
13    left_sums = all_subset_sums(left_part)
14    right_sums = all_subset_sums(right_part)
15
16    for l_sum in left_sums:
17        if (exact_sum - l_sum) in right_sums:
18            return True
19
20    return False
21
22 # Test cases
23 print(meet_in_the_middle_exact([1, 3, 9, 2, 7, 12], 15)) # Output: True or
24 print(meet_in_the_middle_exact([3, 34, 4, 12, 5, 2], 15)) # Output: True or
25
```

ERROR!  
Traceback (most recent call last):  
File "<main.py>", line 23, in <module>  
File "<main.py>", line 13, in meet\_in\_the\_middle\_exact  
File "<main.py>", line 5, in all\_subset\_sums  
NameError: name 'combinations' is not defined

=== Code Exited With Errors ===

5. Given two  $2 \times 2$  Matrices A and B

$A = \begin{pmatrix} 1 & 7 \\ 3 & 5 \end{pmatrix}$   $B = \begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix}$

Use Strassen's matrix multiplication algorithm to compute the product matrix C such that

$C = A \times B$ .

Test Cases:

Consider the following matrices for testing your implementation:

Test Case 1:

$A = \begin{pmatrix} 1 & 7 \\ 3 & 5 \end{pmatrix}$   $B = \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$

Expected Output:

$C = \begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}$

```
main.py  Run  Output  Clear

1 import numpy as np
2
3 def strassen(A, B):
4     def add(X, Y):
5         return np.add(X, Y)
6
7     def subtract(X, Y):
8         return np.subtract(X, Y)
9
10    def strassen_mult(A, B):
11        if len(A) == 2:
12            a, b, c, d = A[0][0], A[0][1], A[1][0], A[1][1]
13            e, f, g, h = B[0][0], B[0][1], B[1][0], B[1][1]
14
15            p1 = a * (f - h)
16            p2 = (a + b) * h
17            p3 = (c + d) * e
18            p4 = d * (g - e)
19            p5 = (a + d) * (e + h)
20            p6 = (b - d) * (g + h)
21            p7 = (a - c) * (e + f)
22
23            C11 = p5 + p4 - p2 + p6
24            C12 = p1 + p2
25            C21 = p3 + p4
26            C22 = p5 + p1 - p3 - p7
27
28            return [[C11, C12], [C21, C22]]
29
30    return strassen_mult(A, B)
```

```
[[34, 22], [38, 34]]

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

6. Given two integers  $X=1234$  and  $Y=5678$ : Use the Karatsuba algorithm to compute the product  $Z=X \times Y$

Test Case 1:

Input:  $x=1234, y=5678$

Expected Output:  $z=1234 \times 5678 = 701665$

```
main.py  Run  Output  Clear

1 def karatsuba(x, y):
2     if x < 10 or y < 10:
3         return x * y
4
5     m = min(len(str(x)), len(str(y)))
6     m2 = m // 2
7
8     high1, low1 = divmod(x, 10**m2)
9     high2, low2 = divmod(y, 10**m2)
10
11    z0 = karatsuba(low1, low2)
12    z1 = karatsuba((low1 + high1), (low2 + high2))
13    z2 = karatsuba(high1, high2)
14
15    return z2 * 10**(2*m2) + (z1 - z2 - z0) * 10**m2 + z0
16
17 # Test case
18 x = 1234
19 y = 5678
20 print(karatsuba(x, y)) # Output: 701665
21
```

```
7006652

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