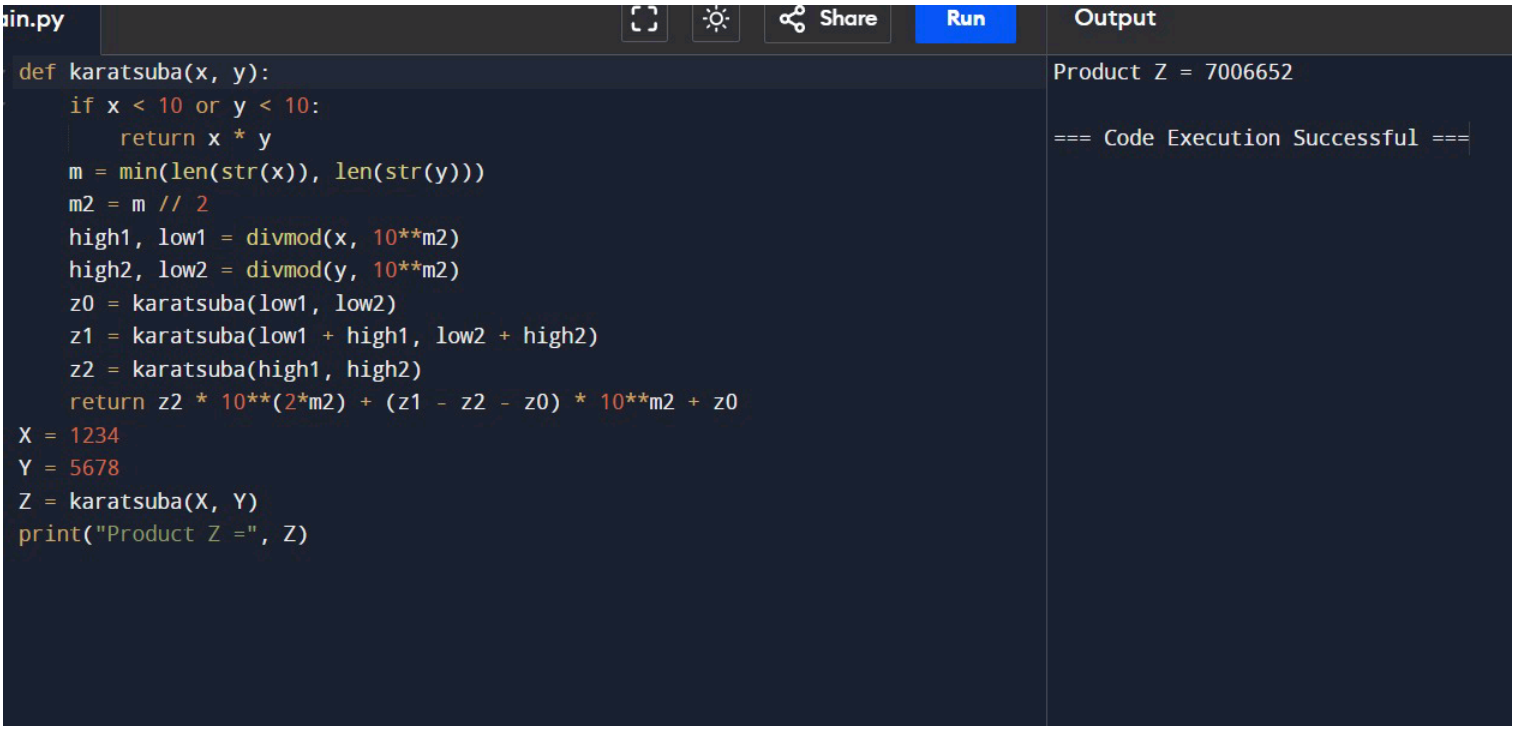
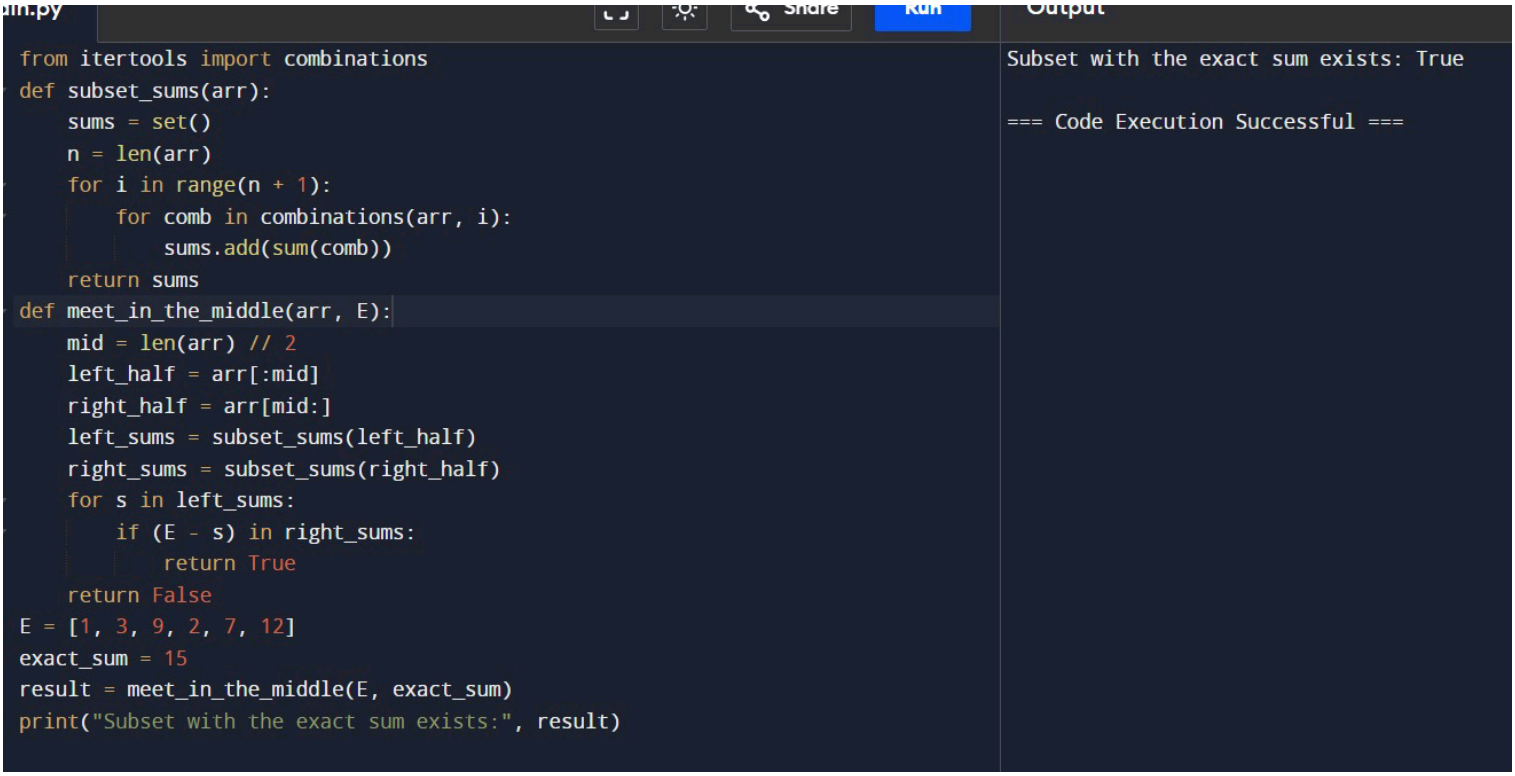

ain.py
Share
Run
Output
def karatsuba(x, y):
if x < 10 or y < 10:
return x * y
m = min(len(str(x)), len(str(y)))
m2 = m // 2
high1, low1 = divmod(x, 10**m2)
high2, low2 = divmod(y, 10**m2)
z0 = karatsuba(low1, low2)
z1 = karatsuba(low1 + high1, low2 + high2)
z2 = karatsuba(high1, high2)
return z2 * 10**(2*m2) + (z1 - z2 - z0) * 10**m2 + z0
X = 1234
Y = 5678
Z = karatsuba(X, Y)
print("Product Z =", Z)
Product Z = 7006652
=== Code Execution Successful ===


Share
Run
Output
from itertools import combinations
def subset_sums(arr):
sums = set()
n = len(arr)
for i in range(n + 1):
for comb in combinations(arr, i):
sums.add(sum(comb))
return sums
def meet_in_the_middle(arr, E):
mid = len(arr) // 2
left_half = arr[:mid]
right_half = arr[mid:]
left_sums = subset_sums(left_half)
right_sums = subset_sums(right_half)
for s in left_sums:
if (E - s) in right_sums:
return True
return False
E = [1, 3, 9, 2, 7, 12]
exact_sum = 15
result = meet_in_the_middle(E, exact_sum)
print("Subset with the exact sum exists:", result)
Subset with the exact sum exists: True
=== Code Execution Successful ===

```
        C11 = M1 + M4 - M5 + M7
        C12 = M3 + M5
        C21 = M2 + M4
        C22 = M1 - M2 + M3 + M6
        return [[C11, C12], [C21, C22]]
    return strassen_2x2(A, B)
A = [[1, 7], [1,3]]
B = [[6, 8], [7,5]]
C = strassen_multiply(A, B)
print("Matrix C:")
for row in C:
    print(row)
```

```python
def strassen_multiply(A, B):
    def strassen_2x2(A, B):
        a, b = A[0][0], A[0][1]
        c, d = A[1][0], A[1][1]
        e, f = B[0][0], B[0][1]
        g, h = 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)
        C11 = M1 + M4 - M5 + M7
        C12 = M3 + M5
        C21 = M2 + M4
        C22 = M1 - M2 + M3 + M6
        return [[C11, C12], [C21, C22]]
    return strassen_2x2(A, B)
A = [[1, 7], [1,3]]
B = [[6, 8], [7,5]]
C = strassen_multiply(A, B)
print("Matrix C:")
for row in C:
    print(row)
```

```
Matrix C:
[55, 43]
[27, 23]

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

main.py

Share Run

```python
def generate_subset_sums(arr):
    subset_sums = set()
    n = len(arr)
    for i in range(1 << n):
        subset_sum = 0
        for j in range(n):
            if i & (1 << j):
                subset_sum += arr[j]
        subset_sums.add(subset_sum)
    return sorted(subset_sums)
def closest_sum(arr, target):
    mid = len(arr) // 2
    left_part = arr[:mid]
    right_part = arr[mid:]
    left_sums = generate_subset_sums(left_part)
    right_sums = generate_subset_sums(right_part)
    closest = float('inf')
    best_sum = 0
    for l_sum in left_sums:
        for r_sum in right_sums:
            current_sum = l_sum + r_sum
            if abs(target - current_sum) < abs(target - closest):
                closest = current_sum
                best_sum = current_sum
    return best_sum
```

Output

```
Subset sum closest to target: 41

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

main.py

Share Run

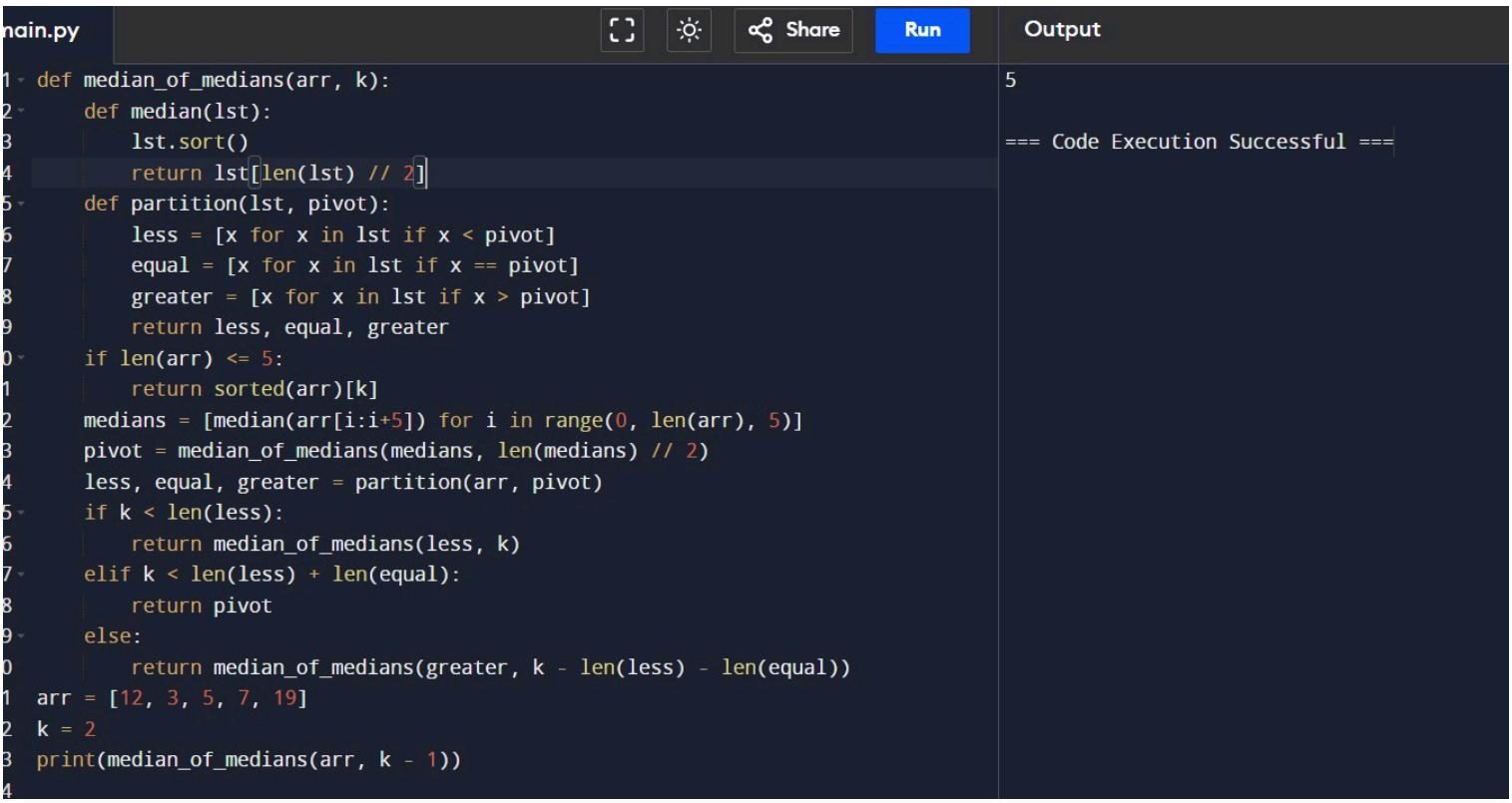
```python
def median_of_medians(arr, k):
    def median(lst):
        lst.sort()
        return lst[len(lst) // 2]
    def partition(lst, pivot):
        less = [x for x in lst if x < pivot]
        equal = [x for x in lst if x == pivot]
        greater = [x for x in lst if x > pivot]
        return less, equal, greater
    if len(arr) <= 5:
        return sorted(arr)[k]
    medians = [median(arr[i:i+5]) for i in range(0, len(arr), 5)]
    pivot = median_of_medians(medians, len(medians) // 2)
    less, equal, greater = partition(arr, pivot)
    if k < len(less):
        return median_of_medians(less, k)
    elif k < len(less) + len(equal):
        return pivot
    else:
        return median_of_medians(greater, k - len(less) - len(equal))
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k = 6
print(median_of_medians(arr, k - 1))

```

Output

```
6

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


main.py
Share
Run
Output
def median_of_medians(arr, k):
def median(lst):
lst.sort()
return lst[len(lst) // 2]
def partition(lst, pivot):
less = [x for x in lst if x < pivot]
equal = [x for x in lst if x == pivot]
greater = [x for x in lst if x > pivot]
return less, equal, greater
if len(arr) <= 5:
return sorted(arr)[k]
medians = [median(arr[i:i+5]) for i in range(0, len(arr), 5)]
pivot = median_of_medians(medians, len(medians) // 2)
less, equal, greater = partition(arr, pivot)
if k < len(less):
return median_of_medians(less, k)
elif k < len(less) + len(equal):
return pivot
else:
return median_of_medians(greater, k - len(less) - len(equal))
arr = [12, 3, 5, 7, 19]
k = 2
print(median_of_medians(arr, k - 1))
5
=== Code Execution Successful ===