



main.py



Share

Run

Output

Clear

```
1 def generate_subset_sums(arr):
2     subset_sums = []
3     n = len(arr)
4     for i in range(1 << n):
5         subset_sum = 0
6         for j in range(n):
7             if i & (1 << j):
8                 subset_sum += arr[j]
9         subset_sums.append(subset_sum)
10    return subset_sums
11 def meet_in_the_middle(arr, target):
12     mid = len(arr) // 2
13     left = arr[:mid]
14     right = arr[mid:]
15     left_sums = generate_subset_sums(left)
16     right_sums = generate_subset_sums(right)
17     right_sums.sort()
18     closest_sum = float('-inf')
19     for sum_left in left_sums:
20         for sum_right in right_sums:
```

10

=== Code Execution Successful ===



main.py



Share

Run

Output

Clear



JS

GO

```
1 def partition(arr, pivot):
2     left = [x for x in arr if x < pivot]
3     right = [x for x in arr if x > pivot]
4     return left, pivot, right
5 def median_of_medians(arr, k):
6     if len(arr) <= 5:
7         return sorted(arr)[k]
8     medians = [sorted(arr[i:i + 5])[len(arr[i:i + 5]) // 2] for
9                 i in range(0, len(arr), 5)]
10    pivot = median_of_medians(medians, len(medians) // 2)
11    left, pivot, right = partition(arr, pivot)
12    if k < len(left):
13        return median_of_medians(left, k)
14    elif k == len(left):
15        return pivot
16    else:
17        return median_of_medians(right, k - len(left) - 1)
18 arr1 = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27]
19 print(median_of_medians(arr1, 5))
20
```

23

=== Code Execution Successful ===



main.py



Share

Run

Output

Clear

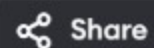
```
1 def strassen_2x2(A, B):
2     a, b, c, d = A[0][0], A[0][1], A[1][0], A[1][1]
3     e, f, g, h = B[0][0], B[0][1], B[1][0], B[1][1]
4     P1 = a * (f - h)
5     P2 = (a + b) * h
6     P3 = (c + d) * e
7     P4 = d * (g - e)
8     P5 = (a + d) * (e + h)
9     P6 = (b - d) * (g + h)
10    P7 = (a - c) * (e + f)
11    C11 = P5 + P4 - P2 + P6
12    C12 = P1 + P2
13    C21 = P3 + P4
14    C22 = P1 + P5 - P3 - P7
15    return [[C11, C12], [C21, C22]]
16 A1 = [[1, 7], [3, 5]]
17 B1 = [[6, 8], [4, 2]]
18 A2 = [[1, 7], [3, 5]]
19 B2 = [[1, 3], [7, 5]]
20 print("Test Case 1 Output:", strassen_2x2(A1, B1))
```

Test Case 1 Output: `[[34, 22], [38, 34]]`Test Case 2 Output: `[[50, 38], [38, 34]]`

=== Code Execution Successful ===



main.py



Run

Output

Clear

```
1 def partition(arr, pivot):
2     left = [x for x in arr if x < pivot]
3     right = [x for x in arr if x > pivot]
4     return left, pivot, right
5 def select(arr, k):
6     if len(arr) <= 5:
7         return sorted(arr)[k]
8     medians = [sorted(arr[i:i + 5])[len(arr[i:i + 5]) // 2] for
9                 i in range(0, len(arr), 5)]
10    pivot = select(medians, len(medians) // 2)
11    left, pivot, right = partition(arr, pivot)
12    if k < len(left):
13        return select(left, k)
14    elif k < len(left) + 1:
15        return pivot
16    else:
17        return select(right, k - len(left) - 1)
18 arr1 = [12, 3, 5, 7, 19]
19 print(select(arr1, 2))
```

7

=== Code Execution Successful ===



main.py



Share

Run

Output

Clear

```
1 def karatsuba(x, y):
2     if x < 10 or y < 10:
3         return x * y
4     n = max(len(str(x)), len(str(y)))
5     m = n // 2
6     a = x // 10**m
7     b = x % 10**m
8     c = y // 10**m
9     d = y % 10**m
10    P1 = karatsuba(a, c)
11    P2 = karatsuba(b, d)
12    P3 = karatsuba(a + b, c + d)
13    return P1 * 10**(2 * m) + (P3 - P1 - P2) * 10**m + P2
14 x = 1234
15 y = 5678
16 print("Output:", karatsuba(x, y))
```

Output:7016652

=== Code Execution Successful ===

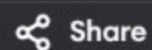


JS





main.py



Run

Output

Clear



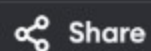
```
10     return subset_sums
11 def meet_in_the_middle(arr, target):
12     mid = len(arr) // 2
13     left = arr[:mid]
14     right = arr[mid:]
15     left_sums = generate_subset_sums(left)
16     right_sums = generate_subset_sums(right)
17     right_sums.sort()
18     closest_sum = float('-inf')
19     for sum_left in left_sums:
20         for sum_right in right_sums:
21             current_sum = sum_left + sum_right
22             if abs(target - current_sum) < abs(target -
                closest_sum):
23                 closest_sum = current_sum
24     return closest_sum
25 set1 = [1, 3, 2, 7, 4, 6]
26 target1 = 10
27 print(meet_in_the_middle(set1, target1))
```

10

=== Code Execution Successful ===



main.py



Run

Output

Clear



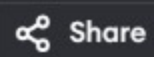
```
9     subset_sums.append(subset_sum)
10     return subset_sums
11 def meet_in_the_middle(arr, target):
12     mid = len(arr) // 2
13     left = arr[:mid]
14     right = arr[mid:]
15     left_sums = generate_subset_sums(left)
16     right_sums = generate_subset_sums(right)
17     right_sums_set = set(right_sums)
18     for sum_left in left_sums:
19         if (target - sum_left) in right_sums_set:
20             return True
21     return False
22 arr1 = [1, 3, 9, 2, 7, 12]
23 target1 = 15
24 arr2 = [3, 34, 4, 12, 5, 2]
25 target2 = 15
26 print(meet_in_the_middle(arr1, target1))
27 print(meet_in_the_middle(arr2, target2))
```

True
True

=== Code Execution Successful ===



main.py



Run

Output

Clear

```
1 def generate_subset_sums(arr):
2     subset_sums = []
3     n = len(arr)
4     for i in range(1 << n):
5         subset_sum = 0
6         for j in range(n):
7             if i & (1 << j):
8                 subset_sum += arr[j]
9             subset_sums.append(subset_sum)
10    return subset_sums
11 def meet_in_the_middle(arr, target):
12     mid = len(arr) // 2
13     left = arr[:mid]
14     right = arr[mid:]
15     left_sums = generate_subset_sums(left)
16     right_sums = generate_subset_sums(right)
17     right_sums_set = set(right_sums)
18     for sum_left in left_sums:
19         if (target - sum_left) in right_sums_set:
20             return True
```

True

True

=== Code Execution Successful ===