main.py

Share Run

```python
def median_of_medians(arr):
    if len(arr) < 10:
        return sorted(arr)[len(arr) // 2]
    sublists = [arr[i:i + 5] for i in range(0, len(arr), 5)]
    medians = [sorted(sublist)[len(sublist) // 2] for sublist in sublists]
    return median_of_medians(medians)
def find_median(arr):
    n = len(arr)
    if n % 2 == 1:
        return median_of_medians(arr)
    else:
        return (median_of_medians(arr[:n//2]) + median_of_medians(arr[n//2:]))
            / 2
array = [24, 36, 42, 18, 21, 30, 45, 50, 48]
median = find_median(array)
print(f"The median is: {median}")

```

Output

```
The median is: 36

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

main.py

Share Run

```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

data = [20,18,12,8,5,-2]
sorted_data = insertion_sort(data)
print(sorted_data)

```

Output

```
[-2, 5, 8, 12, 18, 20]

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

main.py

Share Run Output

```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
arr = [20,18,12,8,5,-2]
sorted_arr = selection_sort(arr)
print(sorted_arr)

```

```
[-2, 5, 8, 12, 18, 20]

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

main.py

Share Run

```python
def longest_palindromic_subsequence(s):
    n = len(s)
    dp = [[0] * n for _ in range(n)]
    for i in range(n):
        dp[i][i] = 1
    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if s[i] == s[j]:
                dp[i][j] = dp[i + 1][j - 1] + 2
            else:
                dp[i][j] = max(dp[i + 1][j], dp[i][j - 1])

    return dp[0][n - 1]
result = longest_palindromic_subsequence("BBABCBCAB")
print("Length of Longest Palindromic Subsequence:", result)
```

Output

```
Length of Longest Palindromic Subsequence: 7

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

main.py

Share Run

Output

```python
def knapsack(weights, values, capacity):
    n = len(values)
    dp = [[0 for _ in range(capacity + 1)] for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(1, capacity + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] +
                    values[i - 1])
            else:
                dp[i][w] = dp[i - 1][w]

    return dp[n][capacity]
weights = [2,3,4,5]
values = [3,4,5,6]
capacity = 5
print(knapsack(weights, values, capacity))

```

```
7

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

main.py

Share Run

```python
import heapq
from collections import defaultdict

class Node:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq

def huffman_coding(char_freq):
    heap = [Node(char, freq) for char, freq in char_freq.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)
```

Output

```
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111

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

```
    return heap[0]

def print_codes(node, code=""):
    if node:
        if node.char is not None:
            print(f"{node.char}: {code}")
        print_codes(node.left, code + "0")
        print_codes(node.right, code + "1")

# Example usage
char_freq = {'a': 5, 'b': 9, 'c': 12, 'd': 13, 'e': 16, 'f': 45}
huffman_tree = huffman_coding(char_freq)
print_codes(huffman_tree)
```

```python
def coin_change_greedy(coins, amount):
    coins.sort(reverse=True)
    count = 0
    for coin in coins:
        while amount >= coin:
            amount -= coin
            count += 1
    return count if amount == 0 else -1
denominations = [1,3,4]
target_amount = 6
result = coin_change_greedy(denominations, target_amount)
print(f'Minimum coins needed: {result}')

```

```
Minimum coins needed: 3

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

main.py

Share Run Output

```python
def is_safe(board, row, col):
    for i in range(col):
        if board[row][i] == 1:
            return False
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    for i, j in zip(range(row, len(board)), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    return True

def solve_n_queens_util(board, col):
    if col >= len(board):
        return True
    for i in range(len(board)):
        if is_safe(board, i, col):
            board[i][col] = 1
            if solve_n_queens_util(board, col + 1):
                return True
            board[i][col] = 0
    return False
```

```
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]

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

```
def solve_n_queens(n):
    board = [[0] * n for _ in range(n)]
    if not solve_n_queens_util(board, 0):
        return "No solution exists"
    return board

# Example usage
n = 8
solution = solve_n_queens(n)
for row in solution:
    print(row)
```

main.py

Share Run

```python
numbers = [1,3,5,7,9,11,13,15,17]
minimum_value = min(numbers)
maximum_value = max(numbers)

print(f"The minimum value is: {minimum_value}")
print(f"The maximum value is: {maximum_value}")

```

Output

```
The minimum value is: 1
The maximum value is: 17

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

main.py

Share Run

```python
numbers = [22,34,35,36,43,67,12,13,15,17]
min_value = min(numbers)
max_value = max(numbers)
print(f"The minimum value is: {min_value}")
print(f"The maximum value is: {max_value}")

```

Output

```
The minimum value is: 12
The maximum value is: 67

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

```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        merge_sort(left_half)
        merge_sort(right_half)

        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1
data = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]
merge_sort(data)
print("Sorted array is:", data)
```

```
Sorted array is: [13, 17, 22, 25, 34, 36, 43, 52, 65, 67]

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