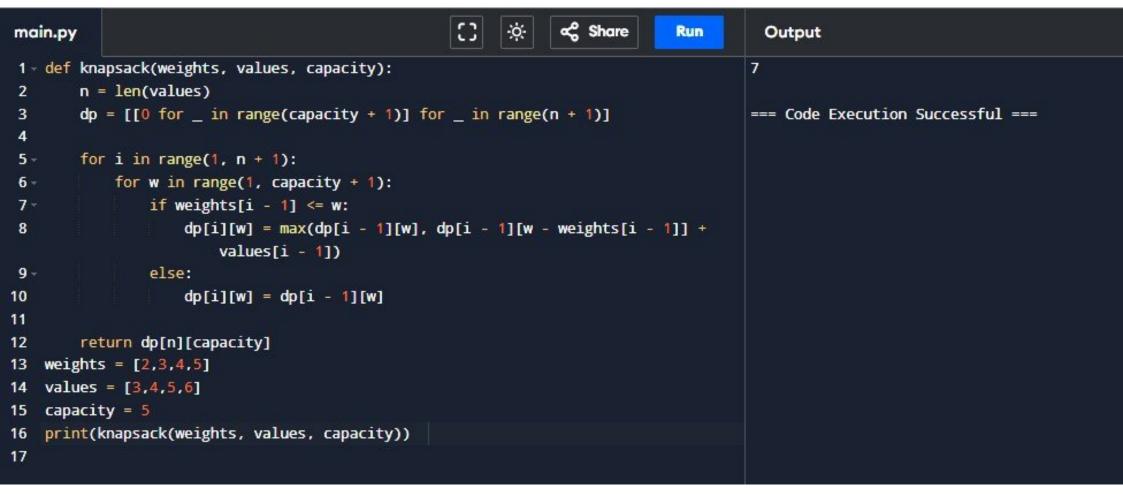
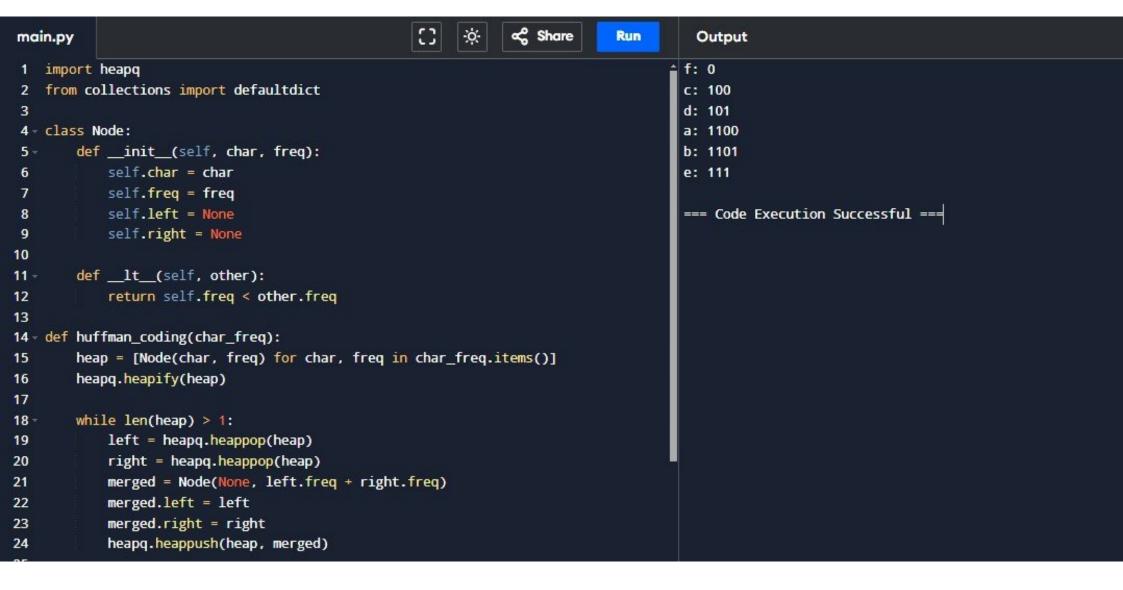


```
⋖ Share
                                                                                        Output
main.py
                                                                              Run
1 - def selection_sort(arr):
                                                                                       [-2, 5, 8, 12, 18, 20]
        n = len(arr)
                                                                                       === Code Execution Successful ===
        for i in range(n):
3 *
            min_idx = i
            for j in range(i+1, n):
5 +
                if arr[j] < arr[min_idx]:</pre>
 7
                    min_idx = j
            arr[i], arr[min_idx] = arr[min_idx], arr[i]
        return arr
10 arr = [20,18,12,8,5,-2]
    sorted_arr = selection_sort(arr)
12 print(sorted_arr)
13
```

```
⋖ Share
                                                                              Run
                                                                                        Output
main.py
1 - def longest_palindromic_subsequence(s):
                                                                                      Length of Longest Palindromic Subsequence: 7
       n = len(s)
2
       dp = [[0] * n for _ in range(n)]
                                                                                      === Code Execution Successful ===
 3
        for i in range(n):
4 -
            dp[i][i] = 1
5
        for length in range(2, n + 1):
6 -
            for i in range(n - length + 1):
7 -
                j = i + length - 1
8
                if s[i] == s[j]:
9 -
                    dp[i][j] = dp[i + 1][j - 1] + 2
10
                else:
11 -
                    dp[i][j] = max(dp[i + 1][j], dp[i][j - 1])
12
13
        return dp[0][n - 1]
14
   result = longest_palindromic_subsequence("BBABCBCAB")
16 print("Length of Longest Palindromic Subsequence:", result)
```





```
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)
```

```
1 - def coin_change_greedy(coins, amount):
                                                                                     Minimum coins needed: 3
        coins.sort(reverse=True)
2
                                                                                     === Code Execution Successful ===
       count = 0
 3
       for coin in coins:
           while amount >= coin:
                amount -= coin
 6
               count += 1
        return count if amount == 0 else -1
 8
   denominations = [1,3,4]
10 target_amount = 6
11 result = coin_change_greedy(denominations, target_amount)
12 print(f'Minimum coins needed: {result}')
13
```

```
1 - def is_safe(board, row, col):
                                                                                      [1, 0, 0, 0, 0, 0, 0, 0, 0]
        for i in range(col):
                                                                                       [0, 0, 0, 0, 0, 0, 1, 0]
2 -
            if board[row][i] == 1:
                                                                                       [0, 0, 0, 0, 1, 0, 0, 0]
 3 -
                return False
                                                                                       [0, 0, 0, 0, 0, 0, 0, 1]
 4
        for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
                                                                                       [0, 1, 0, 0, 0, 0, 0, 0]
 5 -
            if board[i][j] == 1:
                                                                                       [0, 0, 0, 1, 0, 0, 0, 0]
 6 -
                return False
                                                                                       [0, 0, 0, 0, 0, 1, 0, 0]
 7
        for i, j in zip(range(row, len(board)), range(col, -1, -1)):
                                                                                       [0, 0, 1, 0, 0, 0, 0, 0]
 8 -
            if board[i][j] == 1:
9 -
                return False
                                                                                       === Code Execution Successful ===
10
11
        return True
12
13 def solve n queens util(board, col):
        if col >= len(board):
14 -
            return True
15
        for i in range(len(board)):
16 -
            if is_safe(board, i, col):
17 -
                board[i][col] = 1
18
                if solve_n_queens_util(board, col + 1):
19 -
                    return True
20
                board[i][col] = 0
21
        return False
22
```

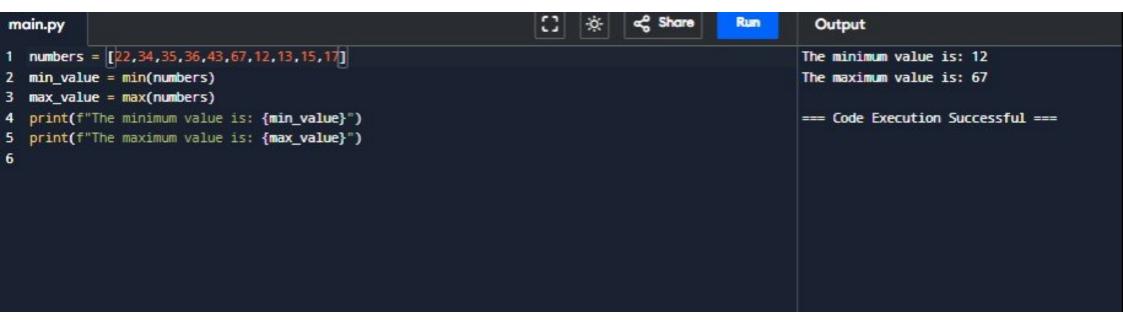
u

Output

main.py

```
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)
```





```
1 def merge_sort(arr):
                                                                                                      Sorted array is: [13, 17, 22, 25, 34, 36, 43, 52, 65, 67]
2 -
       if len(arr) > 1:
3
           mid = len(arr) // 2
                                                                                                         === Code Execution Successful ===
           left_half = arr[:mid]
4
5
           right_half = arr[mid:]
6
7
           merge_sort(left_half)
           merge_sort(right_half)
8
9
           i = j = k = 0
10
           while i < len(left_half) and j < len(right_half):
11
12 -
                if left_half[i] < right_half[j]:</pre>
                    arr[k] = left_half[i]
13
                    i += 1
14
15
                else:
                    arr[k] = right_half[j]
16
17
                   j += 1
18
                k += 1
19
           while i < len(left_half):
20
                arr[k] = left_half[i]
21
22
                i += 1
23
                k += 1
24
           while j < len(right_half):
25
                arr[k] = right_half[j]
26
               j += 1
27
28
                k += 1
29 data = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]
30 merge_sort(data)
31 print("Sorted array is:", data)
```