1. Number of ways to move the ball out of boundary

Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

```python
def findPaths(m, n, N, i, j):
    # Directions array for moving in 4 possible directions (up, down, left, r
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    # 3D DP array to store the number of ways to move out of the grid
    dp = [[[0 for _ in range(n)] for _ in range(m)] for _ in range(N + 1)]

    # Initialize the base cases
    for step in range(1, N + 1):
        for row in range(m):
            for col in range(n):
                for direction in directions:
                    newRow, newCol = row + direction[0], col + direction[1]
                    if newRow < 0 or newRow >= m or newCol < 0 or newCol >= r
                        dp[step][row][col] += 1
                    else:
                        dp[step][row][col] += dp[step - 1][newRow][newCol]

    # Return the result stored in dp[N][i][j]
    return dp[N][i][j]

# Test cases
print(findPaths(2, 2, 2, 0, 0))  # Output: 6
print(findPaths(1, 3, 3, 0, 1))  # Output: 12
```

IDLE Shell 3.12.1

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
6
12
>>>
```

2. Recursive function

Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

```python
def rob_linear(nums):
    if not nums:
        return 0
    if len(nums) == 1:
        return nums[0]

    dp = [0] * len(nums)
    dp[0] = nums[0]
    dp[1] = max(nums[0], nums[1])

    for i in range(2, len(nums)):
        dp[i] = max(dp[i-1], dp[i-2] + nums[i])

    return dp[-1]

def rob(nums):
    if not nums:
        return 0
    if len(nums) == 1:
        return nums[0]

    # Scenario 1: Exclude the last house
    max1 = rob_linear(nums[:-1])

    # Scenario 2: Exclude the first house
    max2 = rob_linear(nums[1:])

    # Return the maximum of both scenarios
    return max(max1, max2)

# Test cases
print("The max money you can rob without alerting police : ", rob([2, 3, 2]))
print("The max money you can rob without alerting police : ",rob([1, 2, 3, 1]
```

IDLE Shell 3.12.1

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
The max money you can rob without alerting police :  3
The max money you can rob without alerting police :  4
>>>
```

3. Non-Recursive Algorithm – I

Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

```python
def climb_stairs(n):
    if n == 1:
        return 1
    if n == 2:
        return 2

    prev2 = 1
    prev1 = 2

    for i in range(3, n + 1):
        current = prev1 + prev2
        prev2 = prev1
        prev1 = current

    return prev1

# Test cases
print("The number of steps climbed :",climb_stairs(4))  # Output: 5
```

IDLE Shell 3.12.1

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
The number of steps climbed : 5
>>>
```

4. Non-Recursive Algorithm – II

Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

```python
def unique_paths(m, n):
    # Create a 2D dp array with dimensions m x n
    dp = [[1] * n for _ in range(m)]

    # Fill the dp array
    for i in range(1, m):
        for j in range(1, n):
            dp[i][j] = dp[i-1][j] + dp[i][j-1]

    return dp[m-1][n-1]

# Test case
print(unique_paths(7, 3))  # Output: 28
```

IDLE Shell 3.12.1

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
28
>>>
```

5. Median-of Medians

Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

```python
def find_median(arr):
    arr.sort()
    return arr[len(arr) // 2]

def partition(arr, low, high, pivot):
    pivot_value = arr[pivot]
    arr[pivot], arr[high] = arr[high], arr[pivot]
    store_index = low
    for i in range(low, high):
        if arr[i] < pivot_value:
            arr[i], arr[store_index] = arr[store_index], arr[i]
            store_index += 1
    arr[store_index], arr[high] = arr[high], arr[store_index]
    return store_index

def select(arr, low, high, k):
    if low == high:
        return arr[low]

    while True:
        if low == high:
            return arr[low]

        pivot_index = median_of_medians(arr, low, high)
        pivot_index = partition(arr, low, high, pivot_index)

        if k == pivot_index:
            return arr[k]
        elif k < pivot_index:
            high = pivot_index - 1
        else:
            low = pivot_index + 1

def median_of_medians(arr, low, high):
    n = high - low + 1
    if n < 10:
        return partition5(arr, low, high)

    medians = []
    i = low
    while i <= high:
```

Ln: 62 Col: 0

IDLE Shell 3.12.1

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
5
>>>
```

Ln: 6 Col: 0

6. Kth smallest Element

Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

```python
def partition(arr, low, high, pivot_index):
    pivot_value = arr[pivot_index]
    arr[pivot_index], arr[high] = arr[high], arr[pivot_index]
    store_index = low
    for i in range(low, high):
        if arr[i] < pivot_value:
            arr[i], arr[store_index] = arr[store_index], arr[i]
            store_index += 1
    arr[store_index], arr[high] = arr[high], arr[store_index]
    return store_index

def median_of_medians(arr, low, high, k):
    if high - low + 1 <= 5:
        return sorted(arr[low:high+1])[k]

    medians = []
    for i in range(low, high + 1, 5):
        sub_high = min(i + 4, high)
        median = sorted(arr[i:sub_high + 1])[len(arr[i:sub_high + 1]) // 2]
        medians.append(median)

    median_of_median = median_of_medians(medians, 0, len(medians) - 1, len(me
    pivot_index = arr.index(median_of_median)
    pivot_index = partition(arr, low, high, pivot_index)

    if k == pivot_index:
        return arr[k]
    elif k < pivot_index:
        return median_of_medians(arr, low, pivot_index - 1, k)
    else:
        return median_of_medians(arr, pivot_index + 1, high, k)

def kth_smallest(arr, k):
    return median_of_medians(arr, 0, len(arr) - 1, k - 1)

# Test case
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k = 6
print(kth_smallest(arr, k))
```

Ln: 40 Col: 0

IDLE Shell 3.12.1

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
6
>>>
```

Ln: 6 Col: 0

7. Closest pair of points

Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

```python
import heapq

def k_closest(points, k):
    # Calculate the distance for each point and store in a heap
    heap = []
    for (x, y) in points:
        distance = x * y + y * y  # We use squared distance for comparison
        heapq.heappush(heap, (distance, (x, y)))

    # Extract the k closest points
    result = []
    for _ in range(k):
        result.append(heapq.heappop(heap)[1])

    return result

# Test case
points = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k = 2
print(k_closest(points, k))  # Expected Output: [[-2, 2], [0, 1]]
```

IDLE Shell 3.12.1

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
[(-2, 2), (0, 1)]
>>>
```

8. 4-sum

Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)

File Edit Format Run Options Window Help

```python
def four_sum_count(A, B, C, D):
    sum_map = {}

    # Calculate sums of pairs from A and B and store them in the hashmap
    for a in A:
        for b in B:
            sum_ab = a + b
            if sum_ab in sum_map:
                sum_map[sum_ab] += 1
            else:
                sum_map[sum_ab] = 1

    # Calculate sums of pairs from C and D and check if their negatives exist
    count = 0
    for c in C:
        for d in D:
            sum_cd = c + d
            target = -sum_cd
            if target in sum_map:
                count += sum_map[target]

    return count

# Test cases
A1 = [1, 2]
B1 = [-2, -1]
C1 = [-1, 2]
D1 = [0, 2]
print(four_sum_count(A1, B1, C1, D1))  # Expected Output: 2
```

IDLE Shell 3.12.1

File Edit Shell Debug Options Window Help

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleva\Desktop\Duplicate.py
2
>>>
```