

1. Selection Sort

```
Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
import time

def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        # Find the minimum element in the remaining unsorted array
        min_index = i
        for j in range(i+1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        # Swap the found minimum element with the first element of the unsorted array
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr

def measure_sort_time(arr):
    start_time = time.time()
    selection_sort(arr)
    end_time = time.time()
    return end_time - start_time

# Example usage:
unsorted_array = [64, 25, 12, 22, 11]
sorted_array = selection_sort(unsorted_array.copy())
print("Sorted array:", sorted_array)

# Measuring time complexity for different input sizes
import random

sizes = [10, 100, 1000, 5000]
for size in sizes:
    test_array = [random.randint(0, 10000) for _ in range(size)]
    time_taken = measure_sort_time(test_array)
    print(f"Time taken to sort an array of size {size}: {time_taken:.6f} seconds")

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\sleval\Desktop\Duplicate.py
Sorted array: [11, 12, 22, 25, 64]
Time taken to sort an array of size 10: 0.000000 seconds
Time taken to sort an array of size 100: 0.000000 seconds
Time taken to sort an array of size 1000: 0.028769 seconds
Time taken to sort an array of size 5000: 0.519951 seconds
>>>
```

2. Perfect Number

```
Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
def is_perfect_number(n):
    if n <= 1:
        return False

    divisors_sum = 1 # Start with 1 because 1 is a divisor of any number
    sqrt_n = int(n ** 0.5)

    for i in range(2, sqrt_n + 1):
        if n % i == 0:
            divisors_sum += i
            if i != n // i:
                divisors_sum += n // i

    return divisors_sum == n

# Example usage
n = 28
print(f"Is {n} a perfect number? {is_perfect_number(n)}")

n = 12
print(f"Is {n} a perfect number? {is_perfect_number(n)}")

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\sleval\Desktop\Duplicate.py
>>>
===== RESTART: C:\Users\sleval\Desktop\Duplicate.py =====
>>>
Is 28 a perfect number? True
Is 12 a perfect number? False
>>>
```

3. Quick Sort

```
Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
import time

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[len(arr) // 2]
        left = [x for x in arr if x < pivot]
        middle = [x for x in arr if x == pivot]
        right = [x for x in arr if x > pivot]
        return quick_sort(left) + middle + quick_sort(right)

def measure_sort_time(arr):
    start_time = time.time()
    quick_sort(arr)
    end_time = time.time()
    return end_time - start_time

# Example usage:
unsorted_array = [64, 25, 12, 22, 11]
sorted_array = quick_sort(unsorted_array.copy())
print("Sorted array:", sorted_array)

# Measuring time complexity for different input sizes
import random

sizes = [10, 100, 1000, 5000]
for size in sizes:
    test_array = [random.randint(0, 10000) for _ in range(size)]
    time_taken = measure_sort_time(test_array)

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\sleval\Desktop\Duplicate.py
Sorted array: [11, 12, 22, 25, 64]
Time taken to sort an array of size 10: 0.000000 seconds
Time taken to sort an array of size 100: 0.000000 seconds
Time taken to sort an array of size 1000: 0.001004 seconds
Time taken to sort an array of size 5000: 0.014505 seconds
>>>
```

4. Write a program to print first 2 minimum values from the numbers in below list

```
Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
def find_two_minimals(numbers):
    if len(numbers) < 2:
        return "List must contain at least two elements."

    first_min = float('inf')
    second_min = float('inf')

    for num in numbers:
        if num < first_min:
            second_min = first_min
            first_min = num
        elif num < second_min and num != first_min:
            second_min = num

    return first_min, second_min

# Example usage:
numbers = [64, 25, 12, 22, 11]
min1, min2 = find_two_minimals(numbers)
print(f"The two smallest values are: {min1} and {min2}")

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\sleval\Desktop\Duplicate.py
The two smallest values are: 11 and 12
>>>
```

5. Bubble Sort

```
Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
import time
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n-i-1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

def measure_sort_time(arr):
    start_time = time.time()
    bubble_sort(arr)
    end_time = time.time()
    return end_time - start_time

# Example usage:
numbers = [10, 5, 80, -2, 15, 23, 45]
sorted_numbers = bubble_sort(numbers.copy())
print("Sorted array:", sorted_numbers)

# Measuring time complexity for different input sizes
import random

sizes = [10, 100, 1000, 5000]
for size in sizes:
    test_array = [random.randint(-10000, 10000) for _ in range(size)]
    time_taken = measure_sort_time(test_array)
    print(f"Time taken to sort an array of size {size}: {time_taken:.6f} sec")

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
Sorted array: [-2, 5, 10, 15, 23, 45, 80]
Time taken to sort an array of size 10: 0.000000 seconds
Time taken to sort an array of size 100: 0.000000 seconds
Time taken to sort an array of size 1000: 0.120351 seconds
Time taken to sort an array of size 5000: 2.800817 seconds
>>>
```

6. Optimal Binary Search Tree

```
Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
def optimal_bst(keys, freq, n):
    # Initialize cost matrix
    cost = [[0 for x in range(n)] for y in range(n)]

    # cost[i][i] is equal to freq[i]
    for i in range(n):
        cost[i][i] = freq[i]

    # Now we need to consider chains of length 2, 3, ... n
    for L in range(2, n + 1): # L is chain length
        for i in range(n - L + 1):
            j = i + L - 1
            cost[i][j] = float('inf')

            # Try making all keys in interval keys[i..j] as root
            for r in range(i, j + 1):
                c = ((cost[i][r-1] if r > i else 0) +
                    (cost[r+1][j] if r < j else 0) +
                    sum(freq[i:j+1]))
                if c < cost[i][j]:
                    cost[i][j] = c

    return cost[0][n-1]

# Example usage:
keys = [10, 12, 20]
freq = [34, 8, 50]
n = len(keys)
print("Cost of Optimal BST is", optimal_bst(keys, freq, n))

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
Cost of Optimal BST is 142
>>>
```

7. Permutation of array

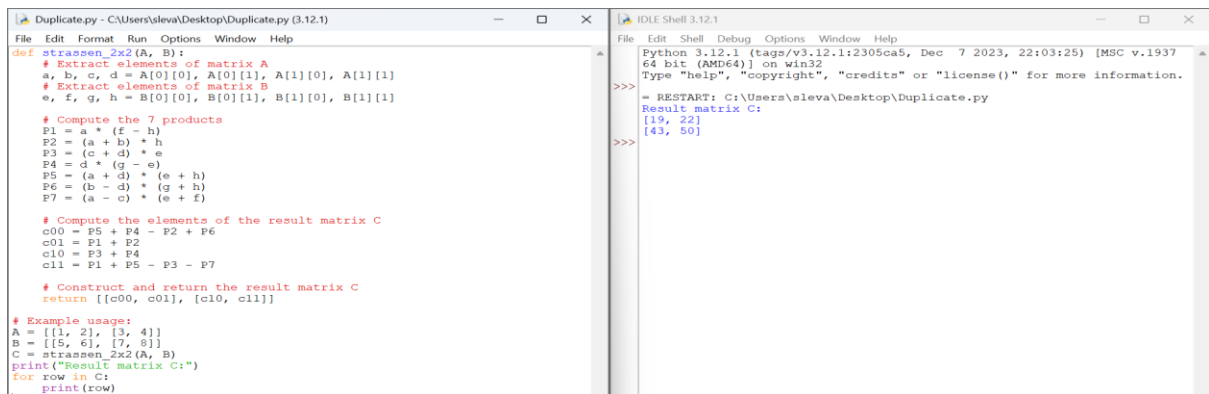
```
Duplicate.py - C:\Users\sleva\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
def permute(nums):
    def backtrack(start):
        if start == len(nums):
            result.append(nums[:])
        for i in range(start, len(nums)):
            nums[start], nums[i] = nums[i], nums[start]
            backtrack(start + 1)
            nums[start], nums[i] = nums[i], nums[start]

    result = []
    backtrack(0)
    return result

# Example usage:
nums = [1, 2, 3]
permutations = permute(nums)
for permutation in permutations:
    print(permutation)

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 =====
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 2, 1]
[3, 1, 2]
>>>
```

8. Strassen's Matrix Multiplication



```
def strassen_2x2(A, B):
    # Extract elements of matrix A
    a, b, c, d = A[0][0], A[0][1], A[1][0], A[1][1]
    # Extract elements of matrix B
    e, f, g, h = B[0][0], B[0][1], B[1][0], B[1][1]

    # Compute the 7 products
    P1 = a * (f - h)
    P2 = (a + b) * h
    P3 = (c + d) * e
    P4 = d * (g - e)
    P5 = (a + d) * (e + h)
    P6 = (b - d) * (g + h)
    P7 = (a - c) * (e + f)

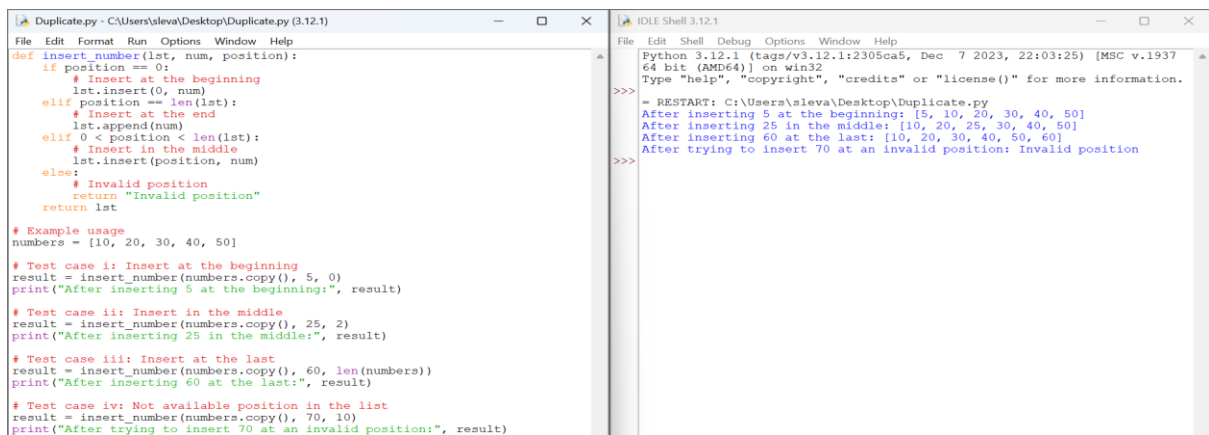
    # Compute the elements of the result matrix C
    c00 = P5 + P4 - P2 + P6
    c01 = P1 + P2
    c10 = P3 + P4
    c11 = P1 + P5 - P3 - P7

    # Construct and return the result matrix C
    return [[c00, c01], [c10, c11]]

# Example usage:
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
C = strassen_2x2(A, B)
print("Result matrix C:")
for row in C:
    print(row)
```

```
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\sleval\Desktop\Duplicate.py
Result matrix C:
[19, 22]
[43, 50]
```

9. Insert a number in list



```
def insert_number(lst, num, position):
    if position == 0:
        # Insert at the beginning
        lst.insert(0, num)
    elif position == len(lst):
        # Insert at the end
        lst.append(num)
    elif 0 < position < len(lst):
        # Insert in the middle
        lst.insert(position, num)
    else:
        # Invalid position
        return "Invalid position"
    return lst

# Example usage
numbers = [10, 20, 30, 40, 50]

# Test case i: Insert at the beginning
result = insert_number(numbers.copy(), 5, 0)
print("After inserting 5 at the beginning:", result)

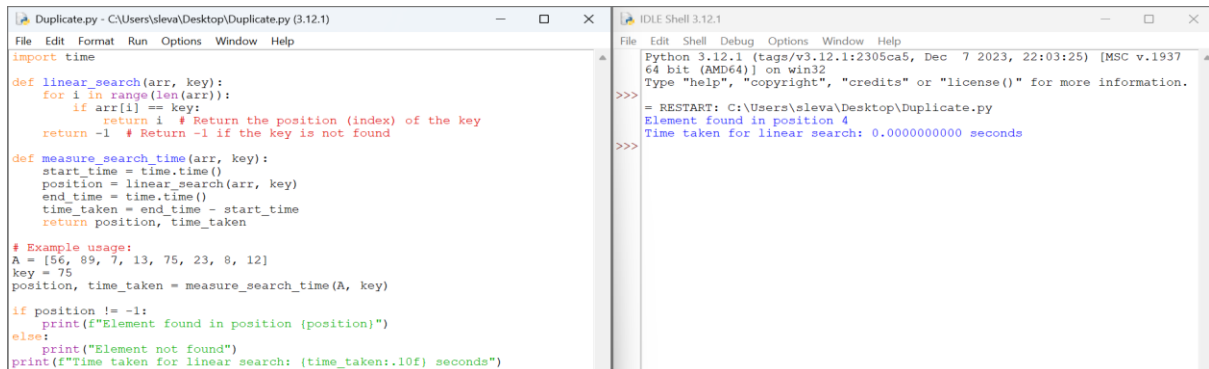
# Test case ii: Insert in the middle
result = insert_number(numbers.copy(), 25, 2)
print("After inserting 25 in the middle:", result)

# Test case iii: Insert at the last
result = insert_number(numbers.copy(), 60, len(numbers))
print("After inserting 60 at the last:", result)

# Test case iv: Not available position in the list
result = insert_number(numbers.copy(), 70, 10)
print("After trying to insert 70 at an invalid position:", result)
```

```
= RESTART: C:\Users\sleval\Desktop\Duplicate.py
After inserting 5 at the beginning: [5, 10, 20, 30, 40, 50]
After inserting 25 in the middle: [10, 20, 25, 30, 40, 50]
After inserting 60 at the last: [10, 20, 30, 40, 50, 60]
After trying to insert 70 at an invalid position: Invalid position
```

10. Linear Search



```
import time

def linear_search(arr, key):
    for i in range(len(arr)):
        if arr[i] == key:
            return i # Return the position (index) of the key
    return -1 # Return -1 if the key is not found

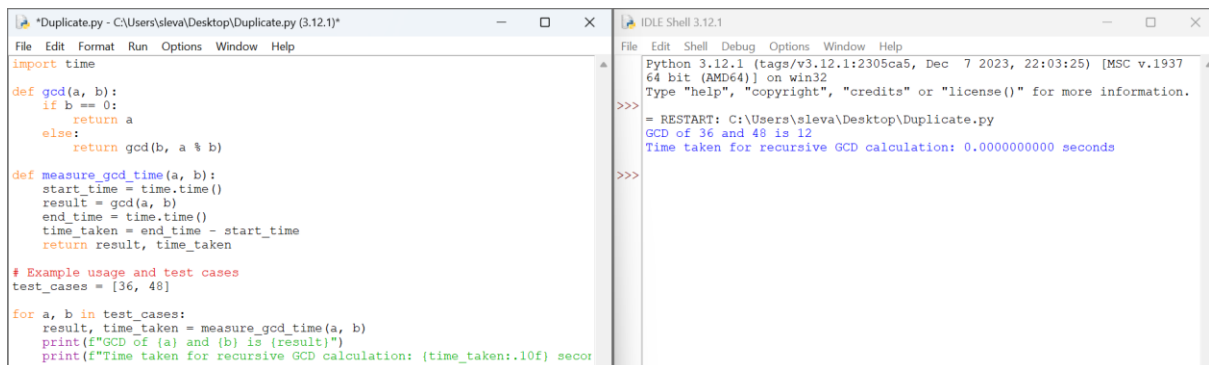
def measure_search_time(arr, key):
    start_time = time.time()
    position = linear_search(arr, key)
    end_time = time.time()
    time_taken = end_time - start_time
    return position, time_taken

# Example usage:
A = [56, 89, 7, 13, 75, 23, 8, 12]
key = 75
position, time_taken = measure_search_time(A, key)

if position != -1:
    print(f"Element found in position {position}")
else:
    print("Element not found")
print(f"Time taken for linear search: {time_taken:.10f} seconds")
```

```
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\sleval\Desktop\Duplicate.py
Element found in position 4
Time taken for linear search: 0.0000000000 seconds
```

11. GCD



```
import time

def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

def measure_gcd_time(a, b):
    start_time = time.time()
    result = gcd(a, b)
    end_time = time.time()
    time_taken = end_time - start_time
    return result, time_taken

# Example usage and test cases
test_cases = [36, 48]

for a, b in test_cases:
    result, time_taken = measure_gcd_time(a, b)
    print(f"GCD of {a} and {b} is {result}")
    print(f"Time taken for recursive GCD calculation: {time_taken:.10f} seconds")
```

```
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\sleval\Desktop\Duplicate.py
GCD of 36 and 48 is 12
Time taken for recursive GCD calculation: 0.0000000000 seconds
```