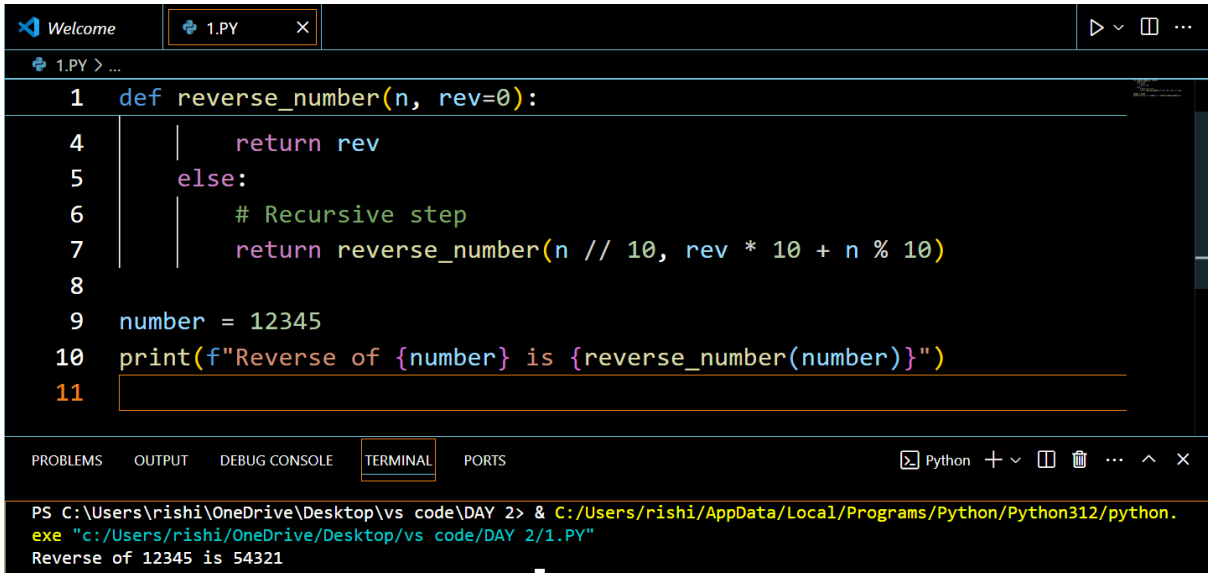
DAY 2 –LAB PROGRAMS

1. Write a program to find the reverse of a given number using recursive.

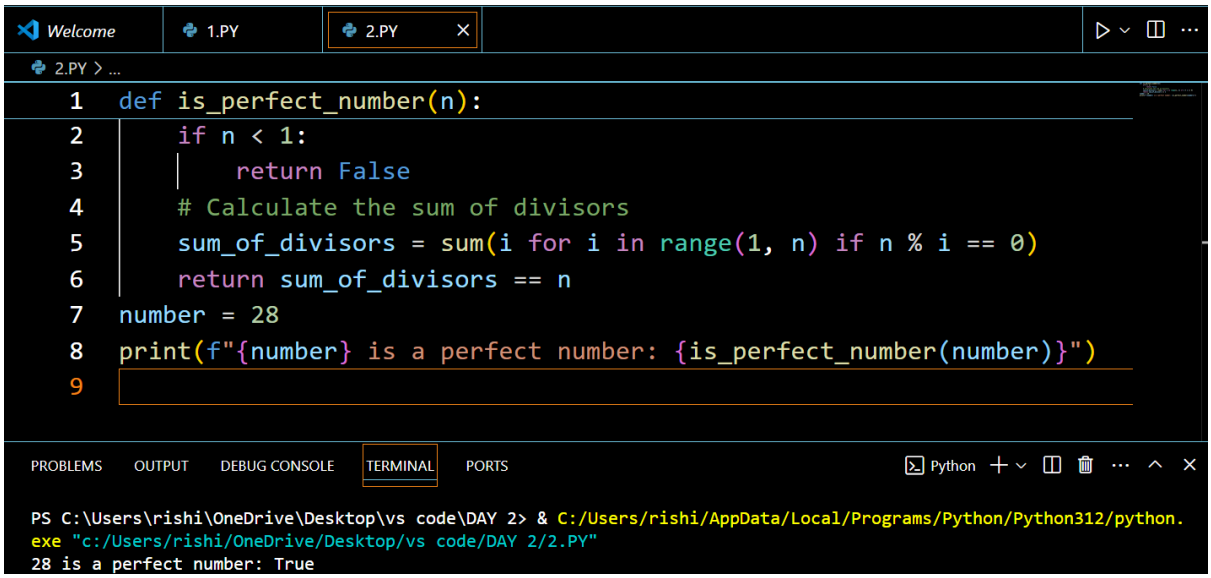
```
def reverse_number(n, rev=0):
    # Base case
    if n == 0:
        return rev
    else:
        # Recursive step
        return reverse_number(n // 10, rev * 10 + n % 10)

number = 12345
print(f"Reverse of {number} is {reverse_number(number)}")
```




```
# Time complexity: O(log10(n))
```

2. Write a program to find the perfect number.

```
def is_perfect_number(n):
    if n < 1:
        return False
    # Calculate the sum of divisors
    sum_of_divisors = sum(i for i in range(1, n) if n % i == 0)
    return sum_of_divisors == n
number = 28
print(f"{number} is a perfect number: {is_perfect_number(number)}")
```

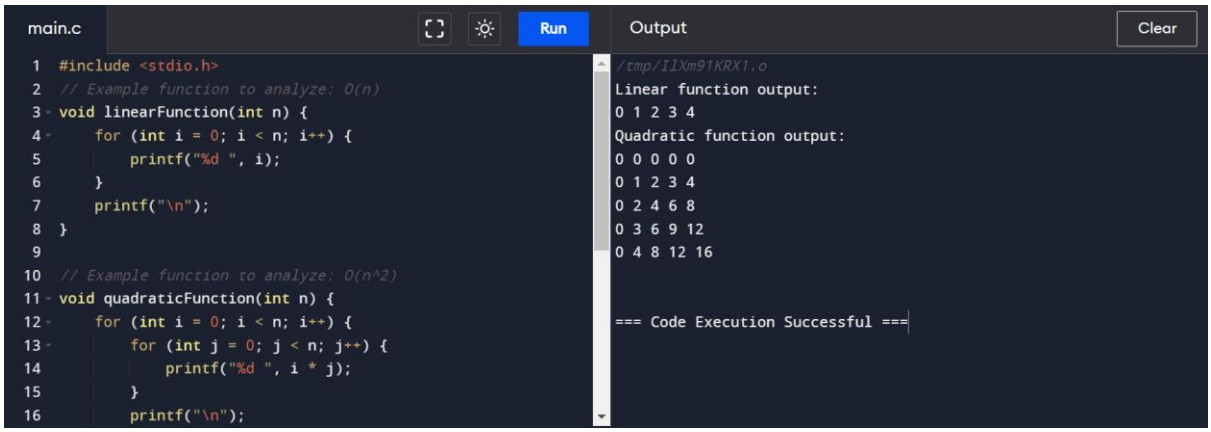



```
# Time complexity: O(n)
```

3. Write C program that demonstrates the usage of these notations by analyzing the time complexity of some example algorithms.

```
#include <stdio.h>

// Example function to analyze: O(n)

void linearFunction(int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", i);
    }
```

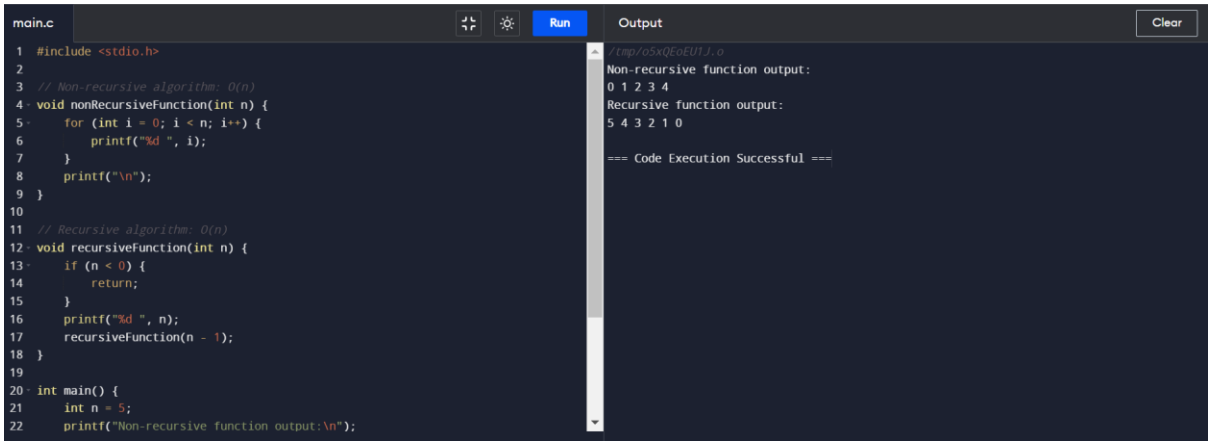
```
    printf("\n");
}

// Example function to analyze: O(n^2)
void quadraticFunction(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", i * j);
        }
        printf("\n");
    }
}
int main() {
    int n = 5;
    printf("Linear function output:\n");
    linearFunction(n);
    printf("Quadratic function output:\n");
    quadraticFunction(n);return 0;}
```



4. Write C programs that demonstrate the mathematical analysis of non-recursive and recursive algorithms.

```
#include <stdio.h>

// Non-recursive algorithm: O(n)
void nonRecursiveFunction(int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", i);
    }
    printf("\n");
}

// Recursive algorithm: O(n)
void recursiveFunction(int n) {
    if (n < 0) {
        return;
    }
    printf("%d ", n);
    recursiveFunction(n - 1);
}

int main() {
    int n = 5;
    printf("Non-recursive function output:\n");
    nonRecursiveFunction(n);
    printf("Recursive function output:\n");
    recursiveFunction(n);
```

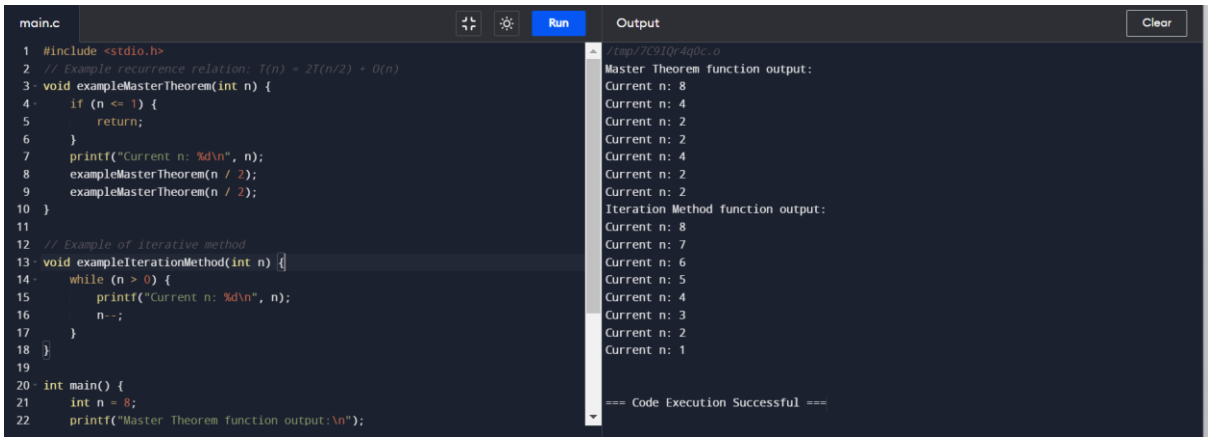
```
    return 0;
}
```



5. Write C programs for solving recurrence relations using the Master Theorem, Substitution Method, and Iteration Method will demonstrate how to calculate the time complexity of an example recurrence relation using the specified technique.

```
#include <stdio.h>

// Example recurrence relation: T(n) = 2T(n/2) + O(n)

void exampleMasterTheorem(int n) {
    if (n <= 1) {
        return;
    }
    printf("Current n: %d\n", n);
    exampleMasterTheorem(n / 2);
    exampleMasterTheorem(n / 2);
}


// Example of iterative method

void exampleIterationMethod(int n) {
    while (n > 0) {
        printf("Current n: %d\n", n);
```

```
        n--;
    }
}

int main() {
    int n = 8;
    printf("Master Theorem function output:\n");
    exampleMasterTheorem(n);  // O(n log n)
    printf("Iteration Method function output:\n");
    exampleIterationMethod(n);  // O(n)
    return 0;
}
```



6. Given two integer arrays nums1 and nums2, return an array of their Intersection. Each element in the result must be unique and you may return the result in any order.

```
def intersection_unique(nums1, nums2):
    result = list(set(nums1) & set(nums2))
    return result


nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
```

print(f"Intersection (unique): {intersection_unique(nums1, nums2)}")

Time complexity: O(n + m), where n and m are the lengths of nums1 and nums2

7. Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in any order.

from collections import Counter

def intersection_with_counts(nums1, nums2):

    c1 = Counter(nums1)

    c2 = Counter(nums2)

    intersection = c1 & c2

    result = list(intersection.elements())

    return result

nums1 = [1, 2, 2, 1]

nums2 = [2, 2]

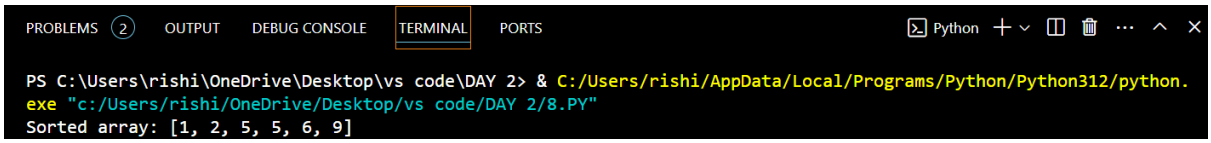
print(f"Intersection (with counts): {intersection_with_counts(nums1, nums2)}")

```
# Time complexity: O(n + m), where n and m are the lengths of nums1 and nums2
```

8. Given an array of integers nums, sort the array in ascending order and return it.You must solve the problem without using any built-in functions in O(nlog(n)) time complexity and with the smallest space complexity possible.

```
def merge_sort(nums):
    if len(nums) > 1:
        mid = len(nums) // 2
        left_half = nums[:mid]
        right_half = nums[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]:
                nums[k] = left_half[i]
                i += 1
            else:
                nums[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            nums[k] = left_half[i]
```

```
            i += 1
            k += 1

        while j < len(right_half):
            nums[k] = right_half[j]
            j += 1
            k += 1

    return nums

nums = [5, 2, 9, 1, 5, 6]
print(f"Sorted array: {merge_sort(nums)}")
```

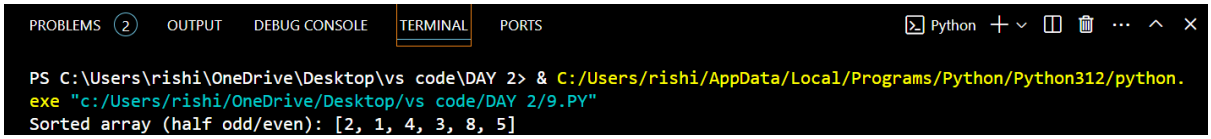



```
# Time complexity: O(n log n)
```

9. Given an array of integers nums, half of the integers in nums are odd, and the other half are even.

```
def sort_halves_odd_even(nums):
    odd_numbers = sorted([x for x in nums if x % 2 != 0])
    even_numbers = sorted([x for x in nums if x % 2 == 0])
    result = []
    for i in range(len(nums)):
        if i % 2 == 0:
            result.append(even_numbers.pop(0))
        else:
            result.append(odd_numbers.pop(0))
    return result
```

```
nums = [3, 1, 4, 2, 5, 8]
print(f"Sorted array (half odd/even): {sort_halves_odd_even(nums)}")
```



Time complexity: O(n log n)

10. Sort the array so that whenever nums[i] is odd, i is odd, and whenever nums[i] is even, i is even. Return any answer array that satisfies this condition.

```
def sort_odd_even_index(nums):
    odds = [x for x in nums if x % 2 != 0]
    evens = [x for x in nums if x % 2 == 0]
    result = [0] * len(nums)
    odd_idx = 1
    even_idx = 0

    for num in odds:
        result[odd_idx] = num
        odd_idx += 2

    for num in evens:
        result[even_idx] = num
        even_idx += 2

    return result

nums = [4, 2, 5, 7]
```

print(f"Sorted array (odd/even index): {sort_odd_even_index(nums)}")

Time complexity: O(n)