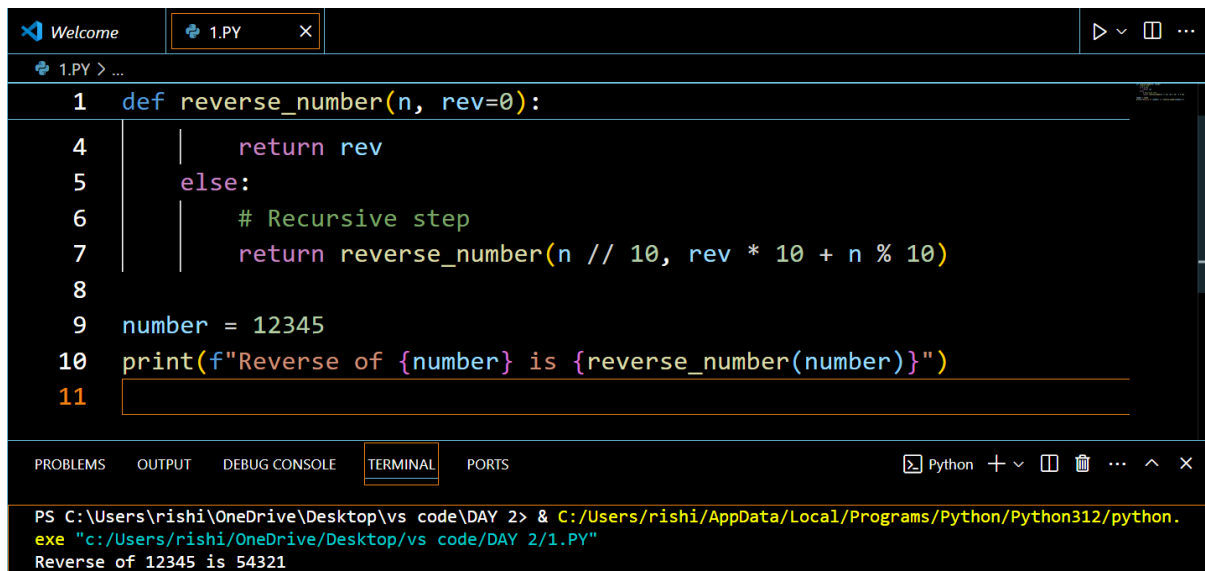


## DAY 2 –LAB PROGRAMS

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)}")
```



```
1 def reverse_number(n, rev=0):  
4     |     return rev  
5     | else:  
6     |     # Recursive step  
7     |     return reverse_number(n // 10, rev * 10 + n % 10)  
8  
9 number = 12345  
10 print(f"Reverse of {number} is {reverse_number(number)}")  
11
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

Python + v [icon] [icon] [icon] [icon]

PS C:\Users\rishi\OneDrive\Desktop\vs code\DAY 2> & C:/Users/rishi/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/rishi/OneDrive/Desktop/vs code/DAY 2/1.PY"  
Reverse of 12345 is 54321

# Time complexity:  $O(\log_{10}(n))$

2. Write a program to find the perfect number.

```
def is_perfect_number(n):
```

```
    if n < 1:
```

```
        return False
```

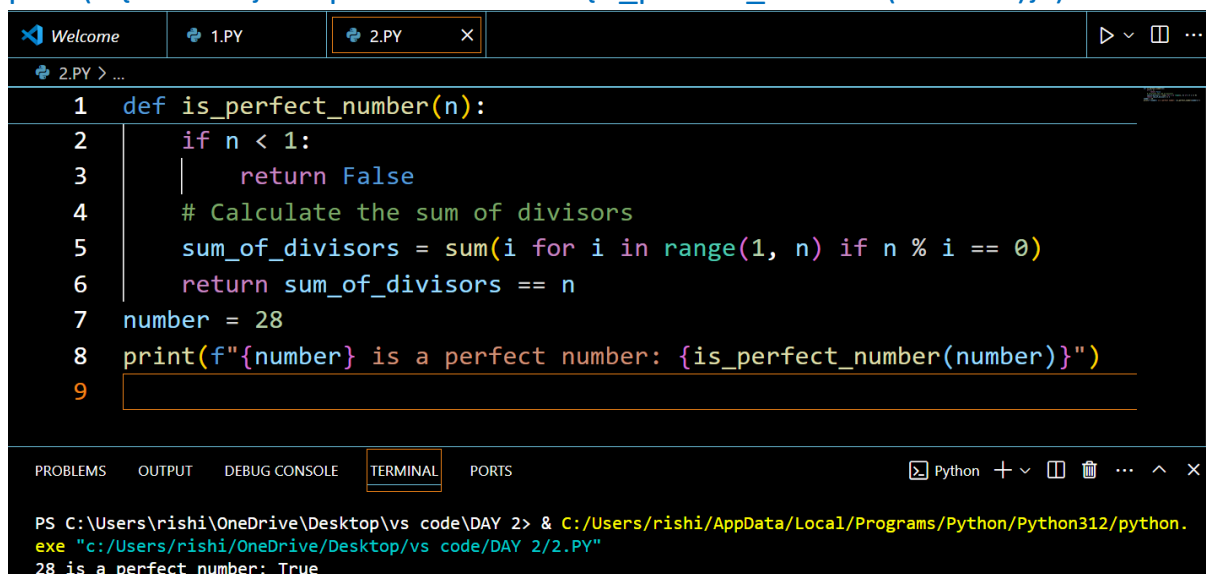
```
    # Calculate the sum of divisors
```

```
    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)}")
```



The screenshot shows a Python IDE with a dark theme. The editor window displays the following code:

```
1 def is_perfect_number(n):
2     if n < 1:
3         return False
4     # Calculate the sum of divisors
5     sum_of_divisors = sum(i for i in range(1, n) if n % i == 0)
6     return sum_of_divisors == n
7 number = 28
8 print(f"{number} is a perfect number: {is_perfect_number(number)}")
9
```

The bottom panel shows the TERMINAL output:

```
PS C:\Users\rishi\OneDrive\Desktop\vs code\DAY 2> & C:/Users/rishi/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/rishi/OneDrive/Desktop/vs code/DAY 2/2.PY"
28 is a perfect number: True
```

# 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");
}

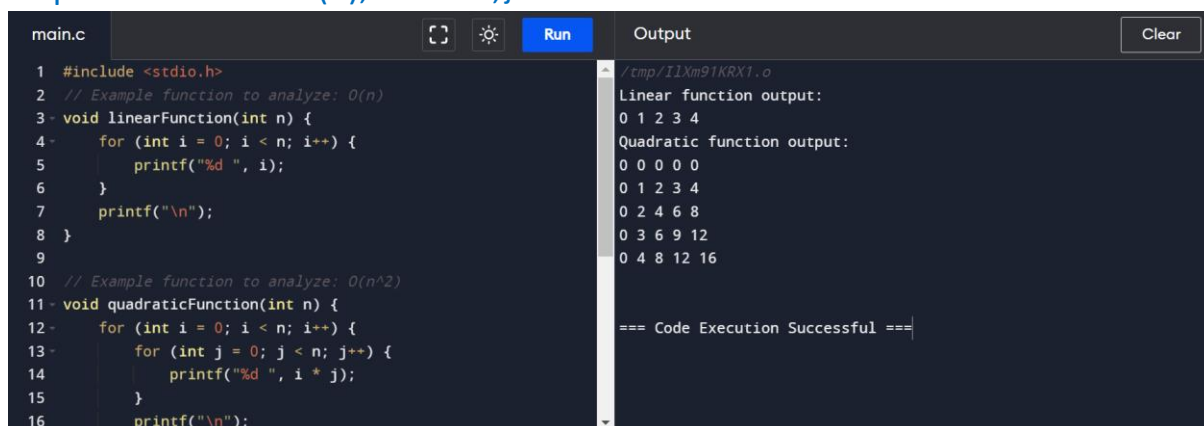
// Example function to analyze: O(n^2)
void quadraticFunction(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", i * j);
        }
        printf("\n");
    }
}

int main() {
    int n = 5;

    printf("Linear function output:\n");
    linearFunction(n);

    printf("Quadratic function output:\n");
    quadraticFunction(n); return 0;}

```



```

main.c  Run  Output  Clear
1 #include <stdio.h>
2 // Example function to analyze: O(n)
3 void linearFunction(int n) {
4     for (int i = 0; i < n; i++) {
5         printf("%d ", i);
6     }
7     printf("\n");
8 }
9
10 // Example function to analyze: O(n^2)
11 void quadraticFunction(int n) {
12     for (int i = 0; i < n; i++) {
13         for (int j = 0; j < n; j++) {
14             printf("%d ", i * j);
15         }
16         printf("\n");

```

```

/tmp/ILXm91KRX1.o
Linear function output:
0 1 2 3 4
Quadratic function output:
0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12
0 4 8 12 16

=== Code Execution Successful ===

```

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");
```

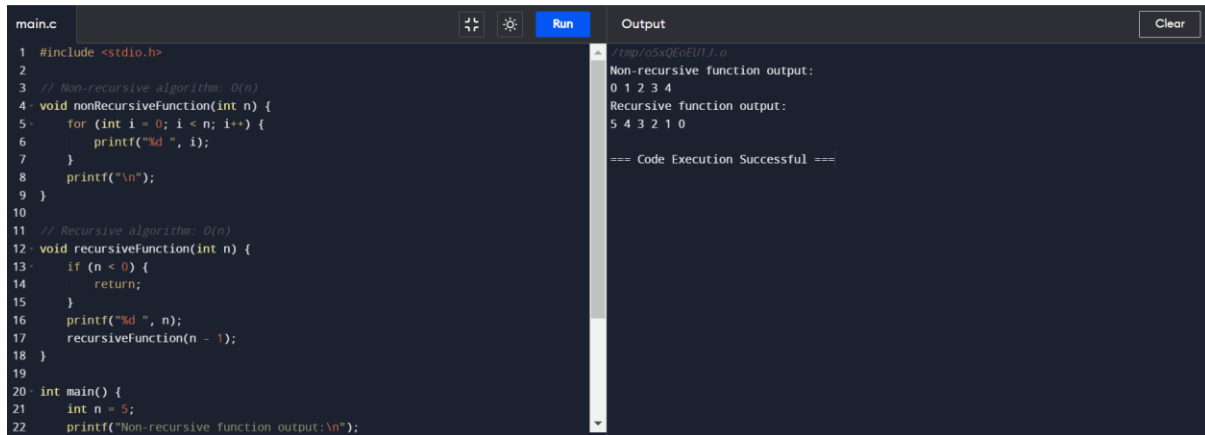
```
    nonRecursiveFunction(n);
```

```
    printf("Recursive function output:\n");
```

```
    recursiveFunction(n);
```

return 0;

}



```
main.c
1 #include <stdio.h>
2
3 // Non-recursive algorithm: O(n)
4 void nonRecursiveFunction(int n) {
5     for (int i = 0; i < n; i++) {
6         printf("%d ", i);
7     }
8     printf("\n");
9 }
10
11 // Recursive algorithm: O(n)
12 void recursiveFunction(int n) {
13     if (n < 0) {
14         return;
15     }
16     printf("%d ", n);
17     recursiveFunction(n - 1);
18 }
19
20 int main() {
21     int n = 5;
22     printf("Non-recursive function output:\n");
23
24     nonRecursiveFunction(n);
25
26     printf("Recursive function output:\n");
27     recursiveFunction(n);
28
29     return 0;
30 }
```

Output

```
/tmp/asxQEoEUTJ.o
Non-recursive function output:
0 1 2 3 4
Recursive function output:
5 4 3 2 1 0

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

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;
}

```

The screenshot shows a C code editor with the following code:

```

1 #include <stdio.h>
2 // Example recurrence relation: T(n) = 2T(n/2) + O(n)
3 void exampleMasterTheorem(int n) {
4     if (n <= 1) {
5         return;
6     }
7     printf("Current n: %d\n", n);
8     exampleMasterTheorem(n / 2);
9     exampleMasterTheorem(n / 2);
10 }
11
12 // Example of iterative method
13 void exampleIterationMethod(int n) {
14     while (n > 0) {
15         printf("Current n: %d\n", n);
16         n--;
17     }
18 }
19
20 int main() {
21     int n = 8;
22     printf("Master Theorem function output:\n");

```

The terminal output window shows the following results:

```

Master Theorem function output:
Current n: 8
Current n: 4
Current n: 2
Current n: 2
Current n: 4
Current n: 2
Current n: 2
Iteration Method function output:
Current n: 8
Current n: 7
Current n: 6
Current n: 5
Current n: 4
Current n: 3
Current n: 2
Current n: 1
=== Code Execution Successful ===

```

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):
```

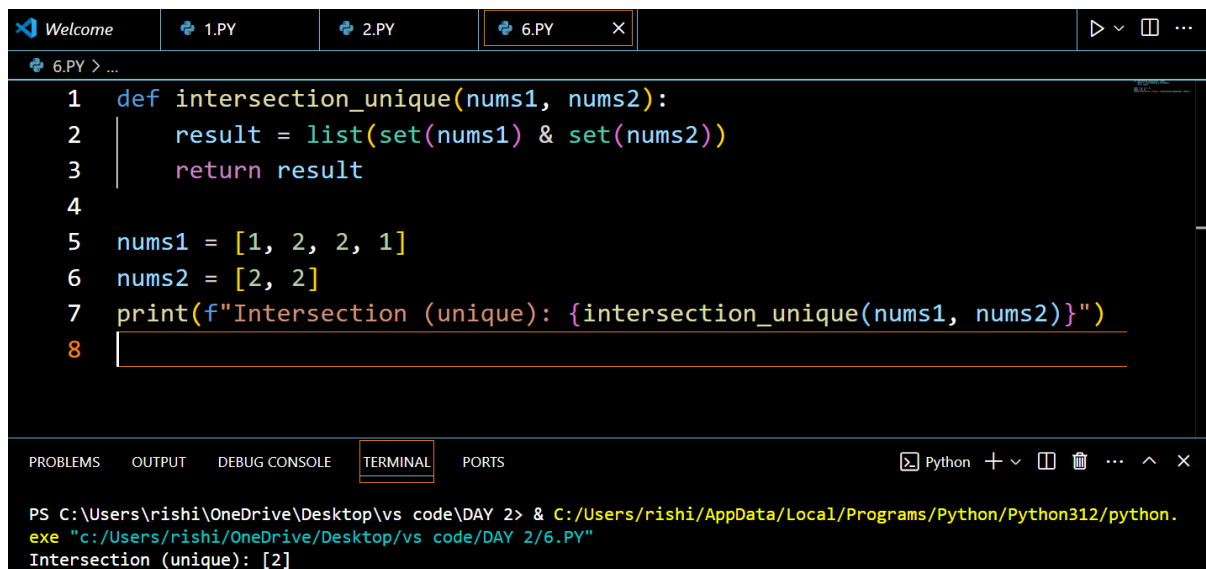
```
    result = list(set(nums1) & set(nums2))
```

```
    return result
```

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

```
nums2 = [2, 2]
```

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



The screenshot shows a VS Code editor with a file named '6.PY' open. The code in the editor is as follows:

```
1 def intersection_unique(nums1, nums2):
2     result = list(set(nums1) & set(nums2))
3     return result
4
5 nums1 = [1, 2, 2, 1]
6 nums2 = [2, 2]
7 print(f"Intersection (unique): {intersection_unique(nums1, nums2)}")
8
```

The terminal at the bottom shows the command to run the script and its output:

```
PS C:\Users\rishi\OneDrive\Desktop\vs code\DAY 2> & C:/Users/rishi/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/rishi/OneDrive/Desktop/vs code/DAY 2/6.PY"
Intersection (unique): [2]
```

# 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  $O(n \log(n))$  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]
```

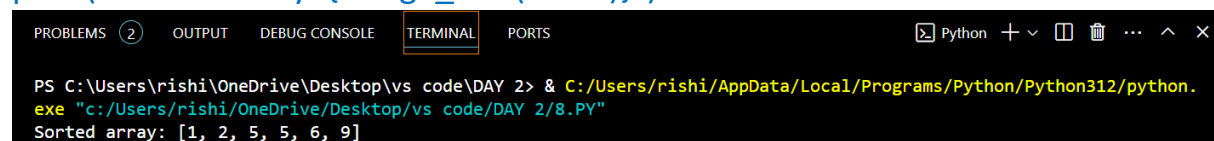
```
    j += 1
```

```
    k += 1
```

```
return nums
```

```
nums = [5, 2, 9, 1, 5, 6]
```

```
print(f"Sorted array: {merge_sort(nums)}")
```

A screenshot of a VS Code terminal window. The terminal has tabs for 'PROBLEMS' (with a count of 2), 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is active and highlighted with a red border), and 'PORTS'. The terminal output shows the command: `PS C:\Users\rishi\OneDrive\Desktop\vs code\DAY 2> & C:/Users/rishi/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/rishi/OneDrive/Desktop/vs code/DAY 2/8.PY"` followed by the output: `Sorted array: [1, 2, 5, 5, 6, 9]`.

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - [ ] [ ] ... ^ x  
PS C:\Users\rishi\OneDrive\Desktop\vs code\DAY 2> & C:/Users/rishi/AppData/Local/Programs/Python/Python312/python.  
exe "c:/Users/rishi/OneDrive/Desktop/vs code/DAY 2/8.PY"  
Sorted array: [1, 2, 5, 5, 6, 9]
```

```
# 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))
```

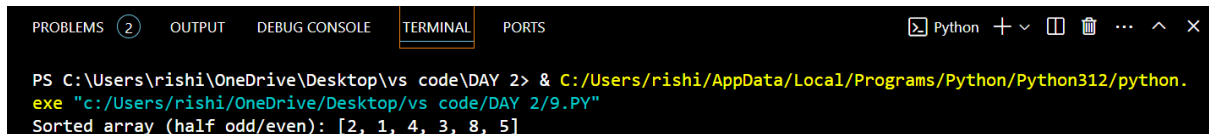
```
        else:
```

```
            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)}")
```



```
PROBLEMS (2) OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\rishi\OneDrive\Desktop\vs code\DAY 2> & C:/Users/rishi/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/rishi/OneDrive/Desktop/vs code/DAY 2/9.PY"
Sorted array (half odd/even): [2, 1, 4, 3, 8, 5]
```

Time complexity:  $O(n \log n)$

10. Sort the array so that whenever  $\text{nums}[i]$  is odd,  $i$  is odd, and whenever  $\text{nums}[i]$  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:
```

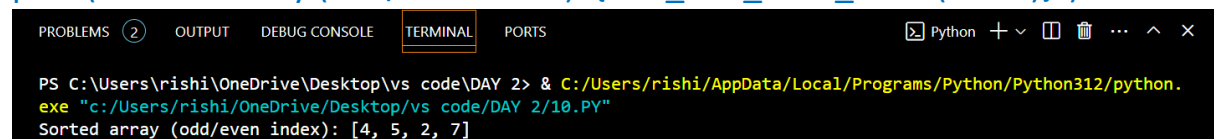
```
        result[even_idx] = num
```

```
        even_idx += 2
```

```
    return result
```

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

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



The screenshot shows a VS Code terminal window with the 'TERMINAL' tab selected. The terminal displays the command prompt 'PS C:\Users\rishi\OneDrive\Desktop\vs code\DAY 2>' followed by the command to run a Python script: '& C:/Users/rishi/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/rishi/OneDrive/Desktop/vs code/DAY 2/10.PY"'. The output of the script is 'Sorted array (odd/even index): [4, 5, 2, 7]'. The terminal window has a dark theme and includes standard VS Code interface elements like the tab bar and window controls.

```
PS C:\Users\rishi\OneDrive\Desktop\vs code\DAY 2> & C:/Users/rishi/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/rishi/OneDrive/Desktop/vs code/DAY 2/10.PY"
Sorted array (odd/even index): [4, 5, 2, 7]
```

# Time complexity:  $O(n)$