LAB 2

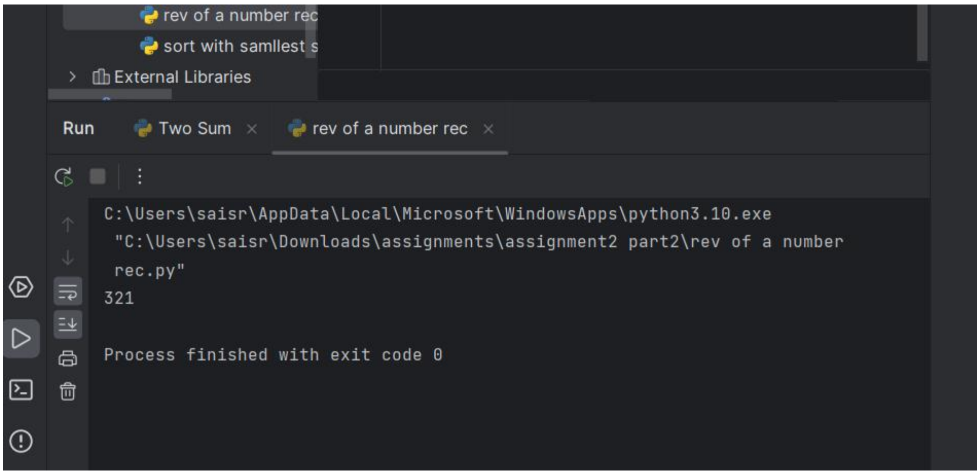
Date:5/6/24

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

   Coding:

```
def rev(n):
    r=0
    while n>0:
        digit = n%10
        r = r*10 +digit
        n = n//10
    return r
num = 123
print(rev(num))
```

   Output:



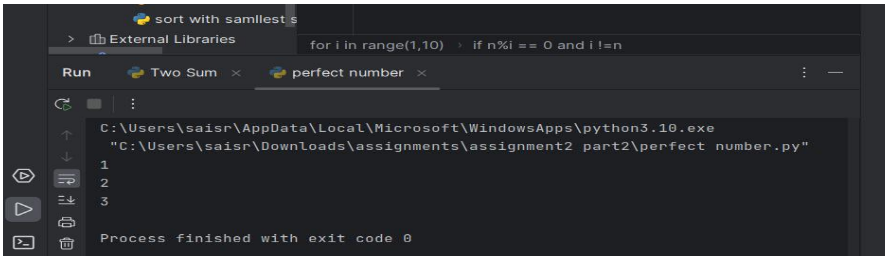
```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
 "C:\Users\saisr\Downloads\assignments\assignment2 part2\rev of a number
 rec.py"
321

Process finished with exit code 0
```

2. **Write a program to find the perfect number.**

   Coding:

```
n=6
for i in range(1,10):
    if n%i == 0 and i !=n:
        print(i)
```

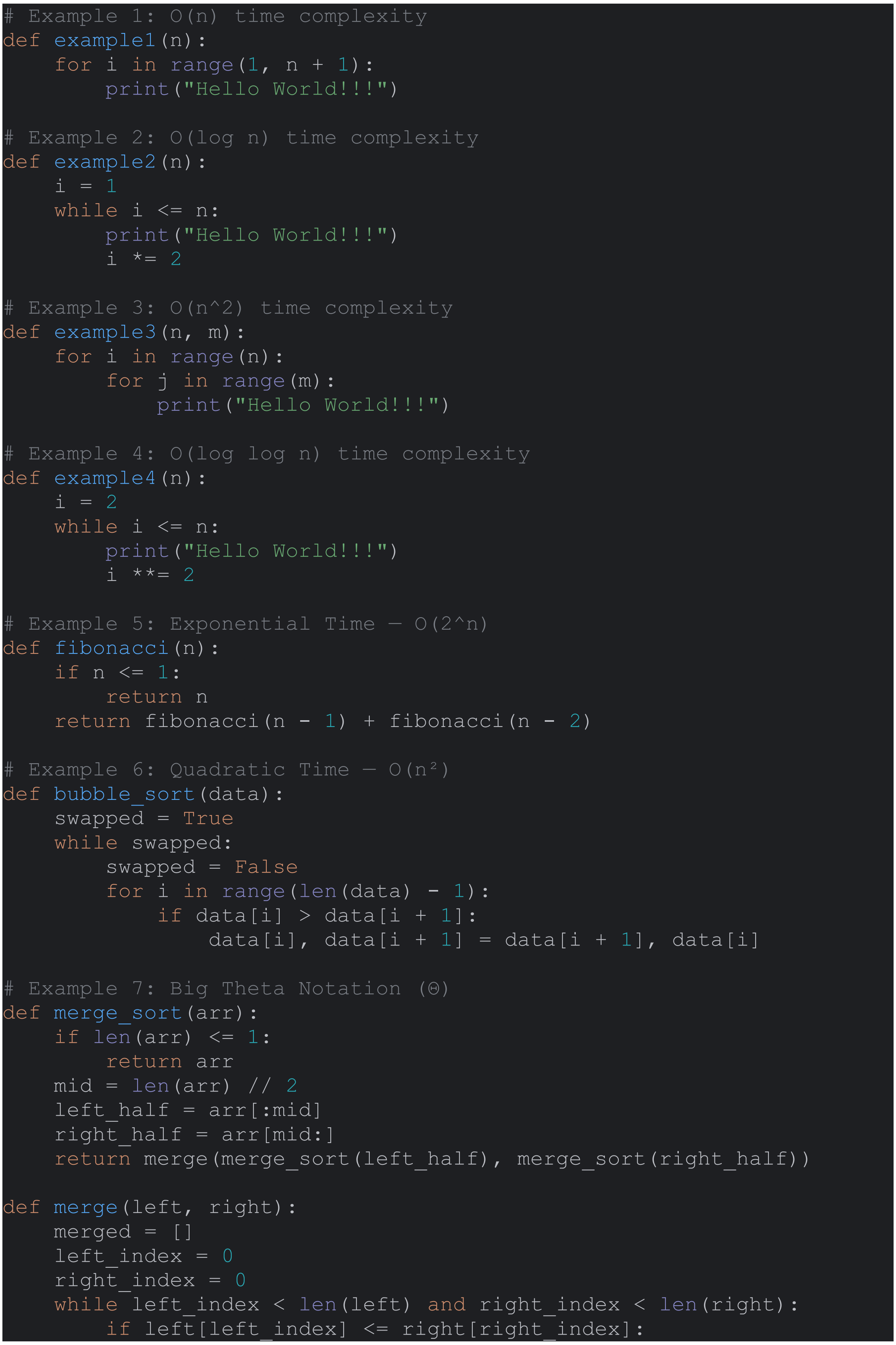
   Output:



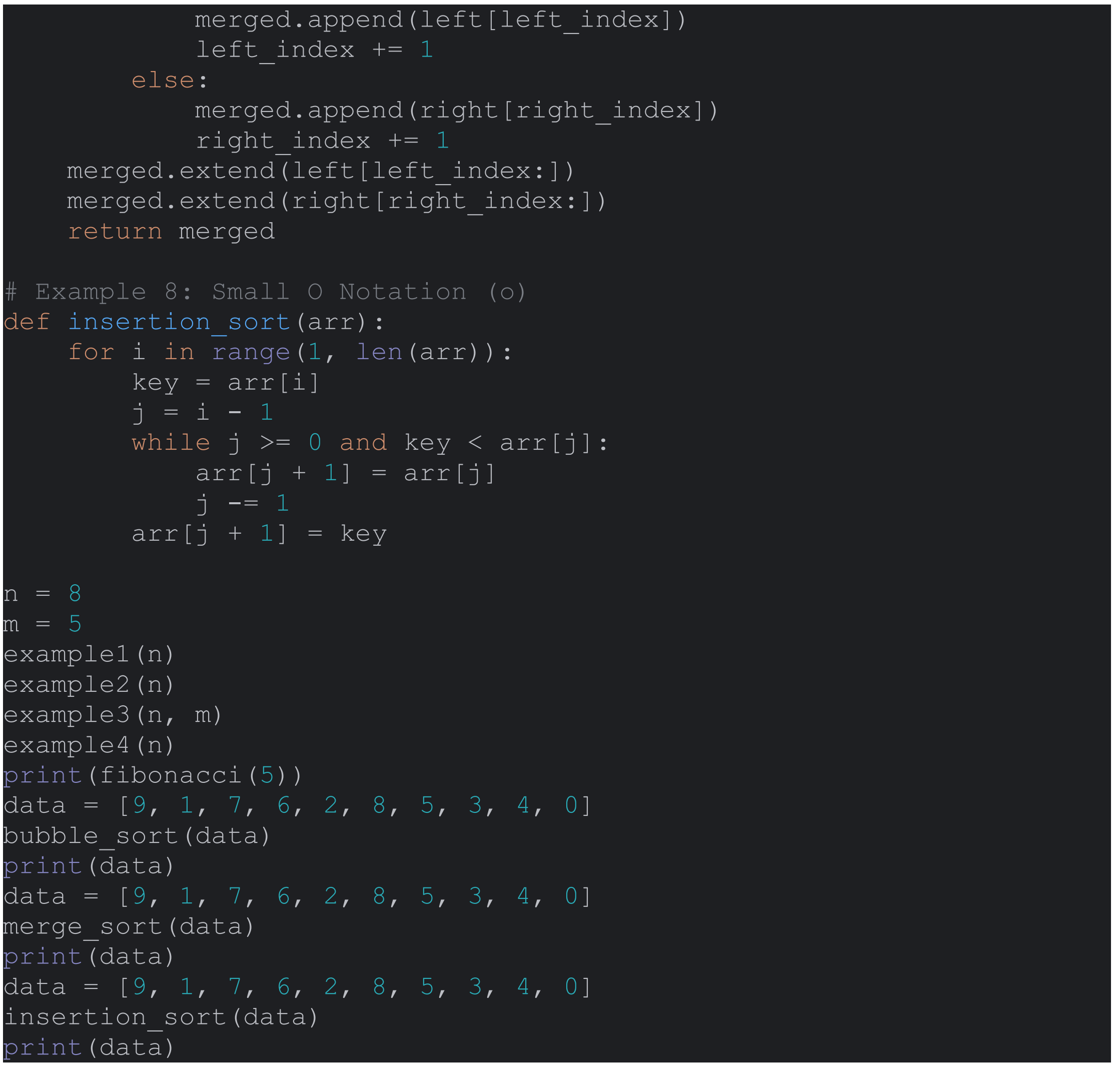
```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
 "C:\Users\saisr\Downloads\assignments\assignment2 part2\perfect number.py"
1
2
3

Process finished with exit code 0
```

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

Coding:

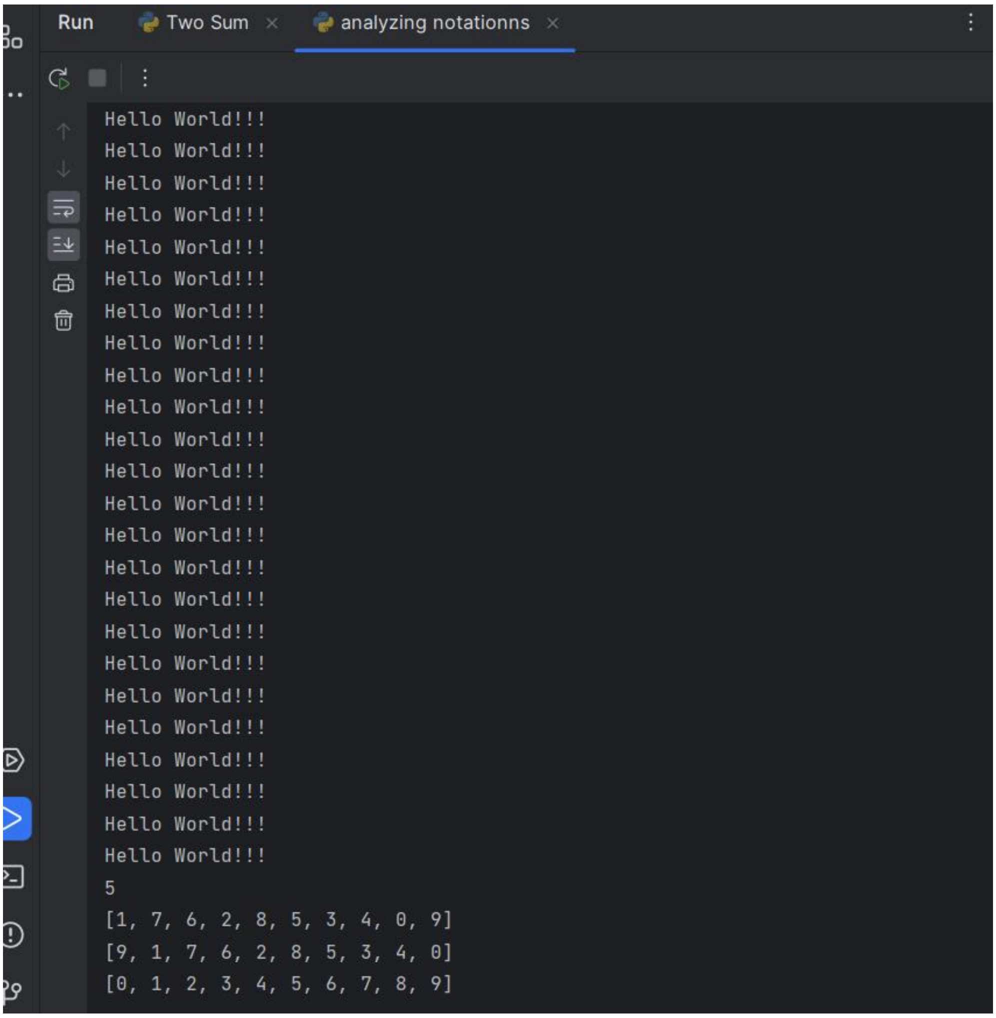


```python
# Example 1: O(n) time complexity
def example1(n):
    for i in range(1, n + 1):
        print("Hello World!!!")

# Example 2: O(log n) time complexity
def example2(n):
    i = 1
    while i <= n:
        print("Hello World!!!")
        i *= 2

# Example 3: O(n^2) time complexity
def example3(n, m):
    for i in range(n):
        for j in range(m):
            print("Hello World!!!")

# Example 4: O(log log n) time complexity
def example4(n):
    i = 2
    while i <= n:
        print("Hello World!!!")
        i **= 2

# Example 5: Exponential Time — O(2^n)
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)

# Example 6: Quadratic Time — O(n²)
def bubble_sort(data):
    swapped = True
    while swapped:
        swapped = False
        for i in range(len(data) - 1):
            if data[i] > data[i + 1]:
                data[i], data[i + 1] = data[i + 1], data[i]

# Example 7: Big Theta Notation (Θ)
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]
    return merge(merge_sort(left_half), merge_sort(right_half))

def merge(left, right):
    merged = []
    left_index = 0
    right_index = 0
    while left_index < len(left) and right_index < len(right):
        if left[left_index] <= right[right_index]:
```



```
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1
    merged.extend(left[left_index:])
    merged.extend(right[right_index:])
    return merged

# Example 8: Small O Notation (o)
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

n = 8
m = 5
example1(n)
example2(n)
example3(n, m)
example4(n)
print(fibonacci(5))
data = [9, 1, 7, 6, 2, 8, 5, 3, 4, 0]
bubble_sort(data)
print(data)
data = [9, 1, 7, 6, 2, 8, 5, 3, 4, 0]
merge_sort(data)
print(data)
data = [9, 1, 7, 6, 2, 8, 5, 3, 4, 0]
insertion_sort(data)
print(data)
```

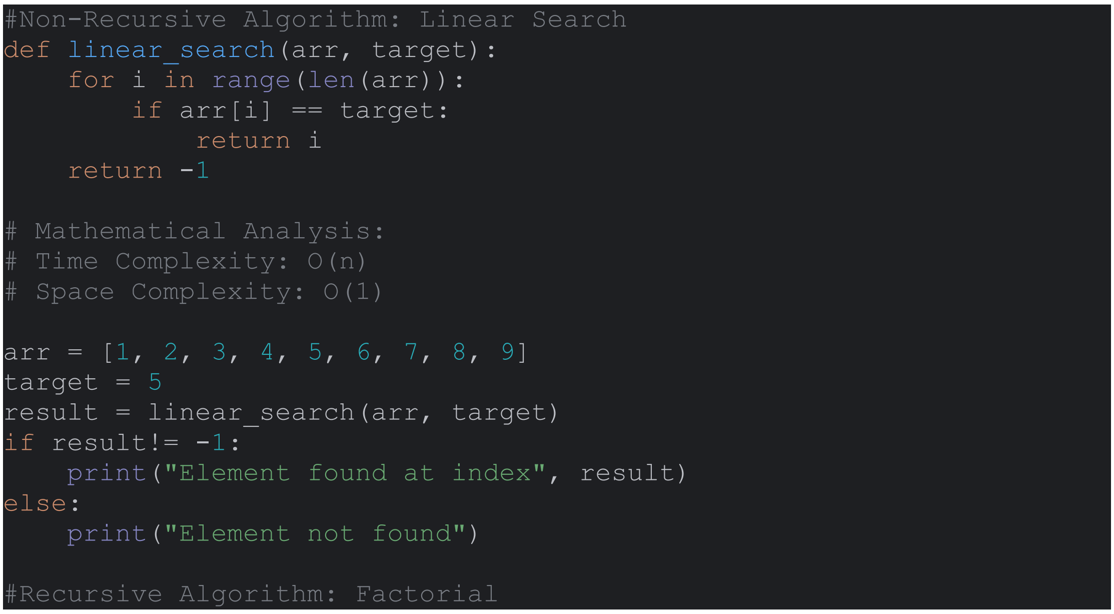
Output:



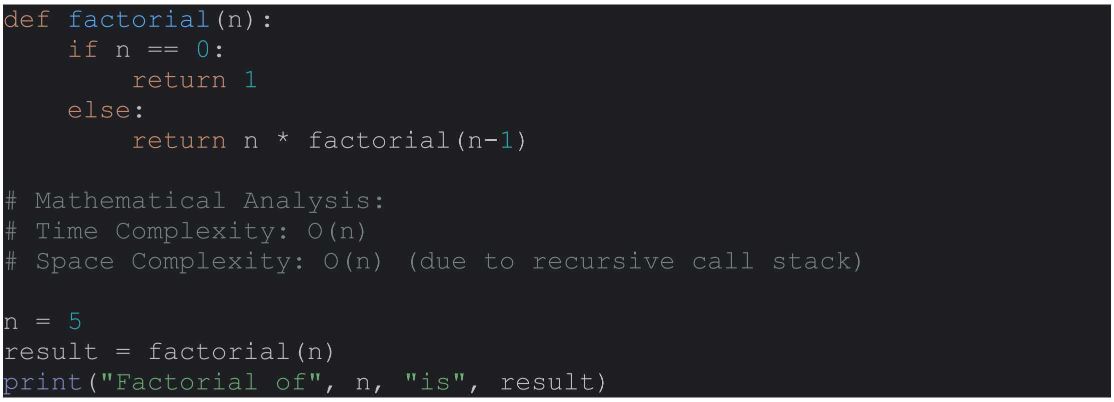
```
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
Hello World!!!
5
[1, 7, 6, 2, 8, 5, 3, 4, 0, 9]
[9, 1, 7, 6, 2, 8, 5, 3, 4, 0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

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

Coding:

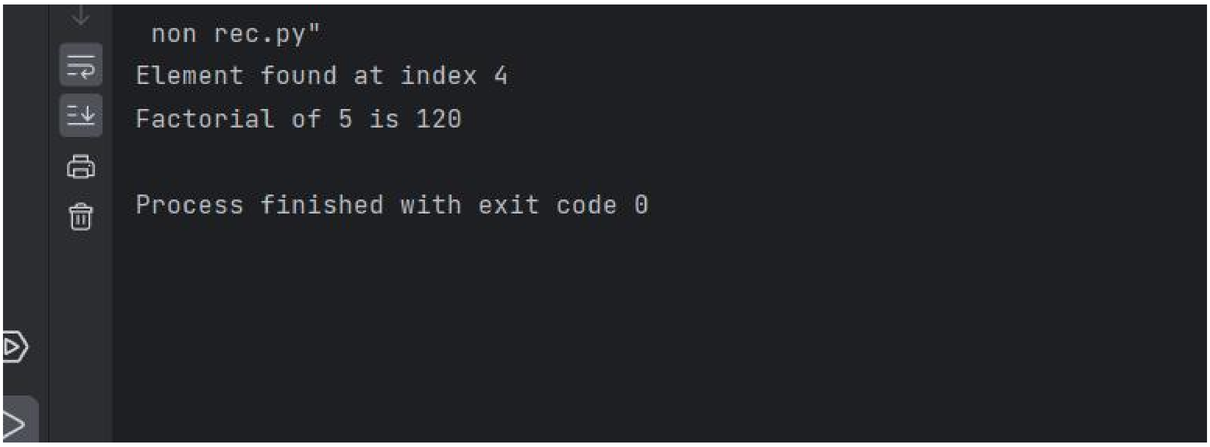


```python
#Non-Recursive Algorithm: Linear Search
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

# Mathematical Analysis:
# Time Complexity: O(n)
# Space Complexity: O(1)

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
target = 5
result = linear_search(arr, target)
if result!= -1:
    print("Element found at index", result)
else:
    print("Element not found")

#Recursive Algorithm: Factorial
```



```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

# Mathematical Analysis:
# Time Complexity: O(n)
# Space Complexity: O(n) (due to recursive call stack)

n = 5
result = factorial(n)
print("Factorial of", n, "is", result)
```

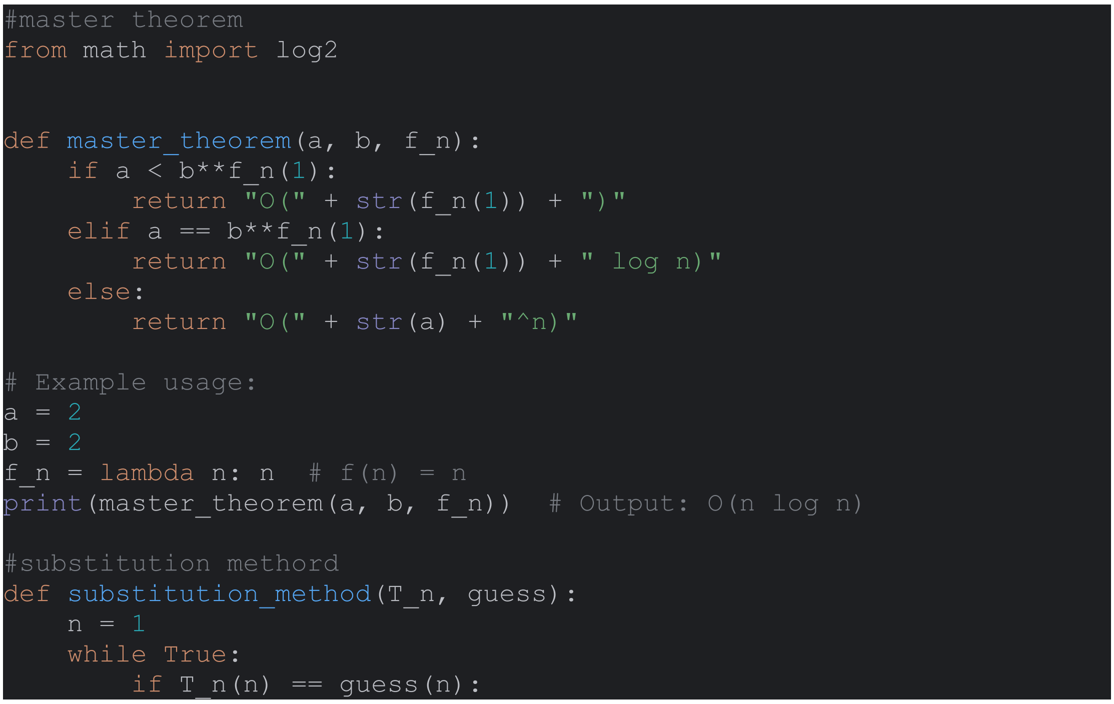
Output:



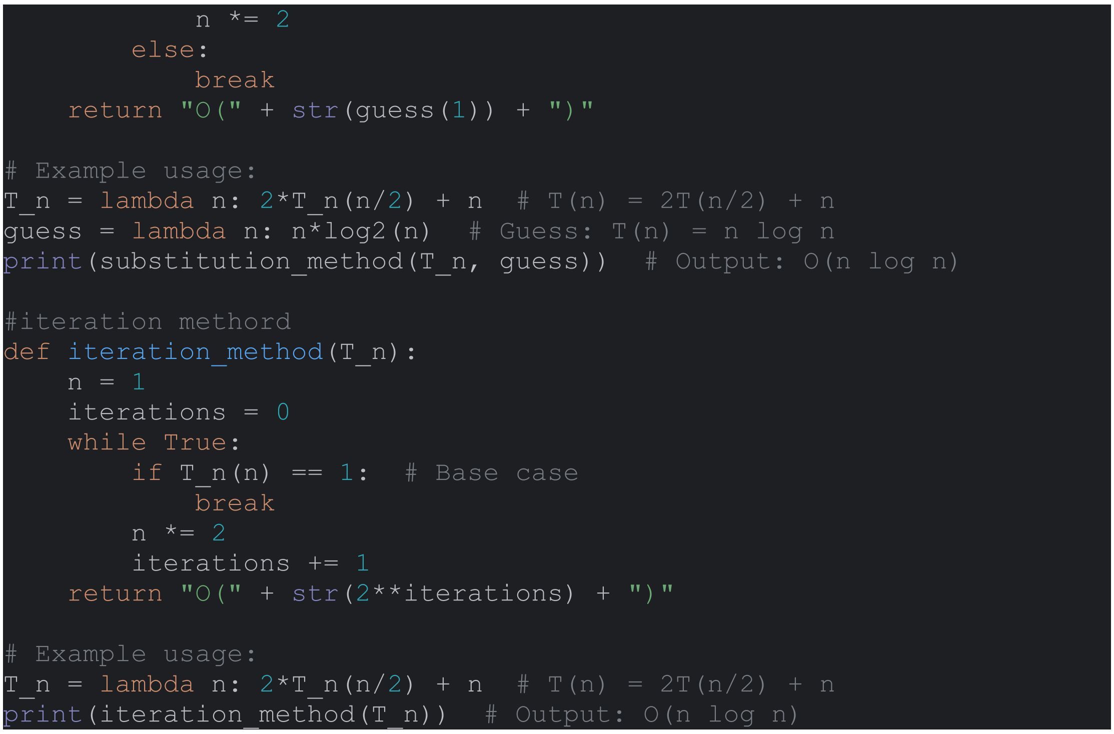
```
 non rec.py"
Element found at index 4
Factorial of 5 is 120

Process finished with exit code 0
```

5. **Write 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.**

Coding:

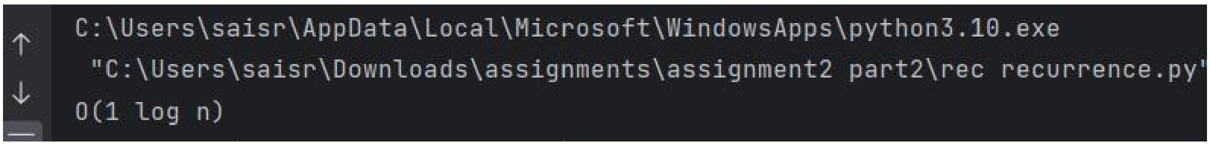


```
#master theorem
from math import log2


def master_theorem(a, b, f_n):
    if a < b**f_n(1):
        return "O(" + str(f_n(1)) + ")"
    elif a == b**f_n(1):
        return "O(" + str(f_n(1)) + " log n)"
    else:
        return "O(" + str(a) + "^n)"

# Example usage:
a = 2
b = 2
f_n = lambda n: n  # f(n) = n
print(master_theorem(a, b, f_n))  # Output: O(n log n)

#substitution methord
def substitution_method(T_n, guess):
    n = 1
    while True:
        if T_n(n) == guess(n):
```



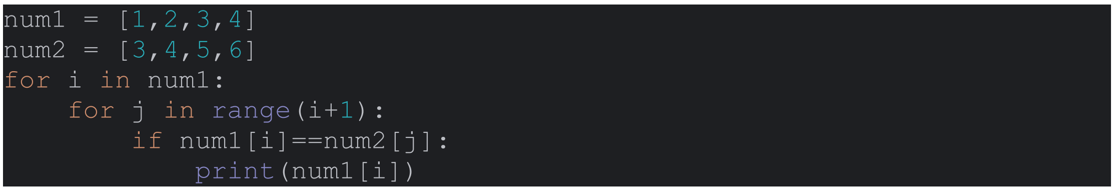
```
            n *= 2
        else:
            break
    return "O(" + str(guess(1)) + ")"

# Example usage:
T_n = lambda n: 2*T_n(n/2) + n  # T(n) = 2T(n/2) + n
guess = lambda n: n*log2(n)  # Guess: T(n) = n log n
print(substitution_method(T_n, guess))  # Output: O(n log n)

#iteration methord
def iteration_method(T_n):
    n = 1
    iterations = 0
    while True:
        if T_n(n) == 1:  # Base case
            break
        n *= 2
        iterations += 1
    return "O(" + str(2**iterations) + ")"

# Example usage:
T_n = lambda n: 2*T_n(n/2) + n  # T(n) = 2T(n/2) + n
print(iteration_method(T_n))  # Output: O(n log n)
```

Output:



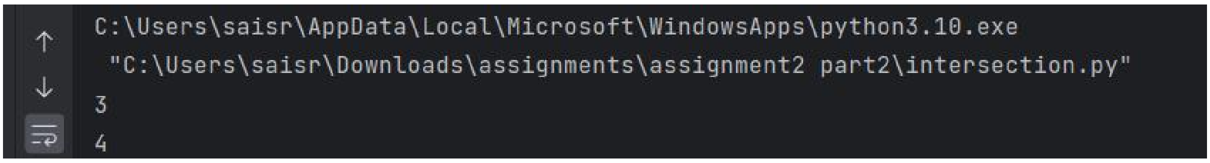
```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
 "C:\Users\saisr\Downloads\assignments\assignment2 part2\rec recurrence.py"
O(1 log n)
```

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.**
   Coding:



```
num1 = [1,2,3,4]
num2 = [3,4,5,6]
for i in num1:
    for j in range(i+1):
        if num1[i]==num2[j]:
            print(num1[i])
```

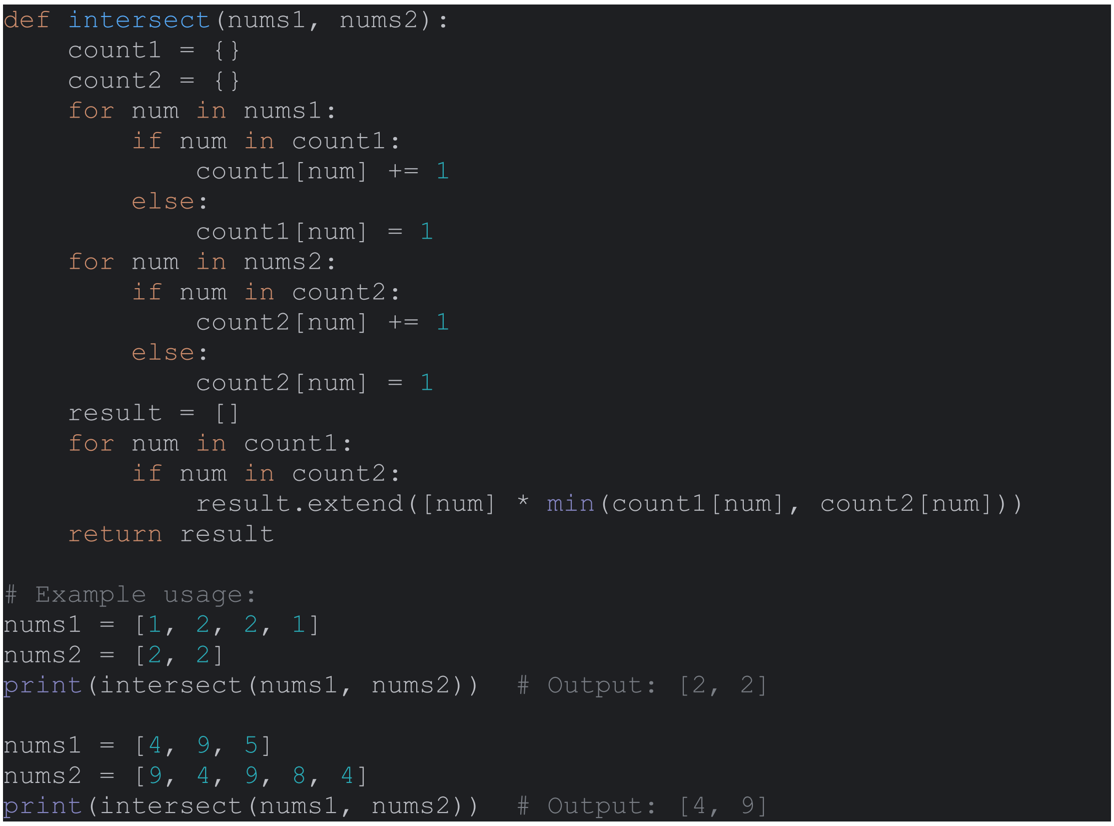
Output:



```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
 "C:\Users\saisr\Downloads\assignments\assignment2 part2\intersection.py"
3
4
```

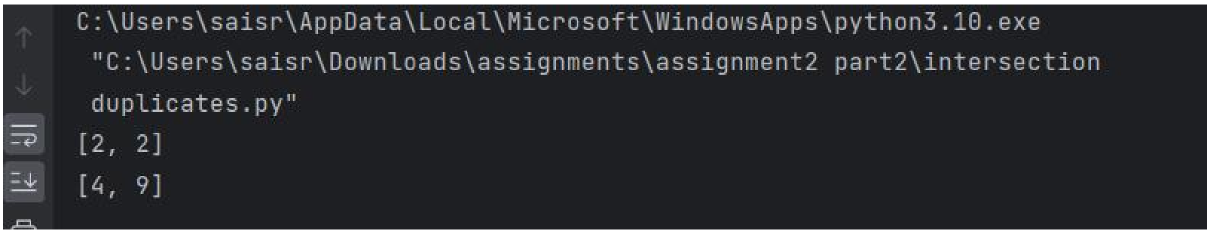
7. **Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in any order.**

Coding:



```
def intersect(nums1, nums2):
    count1 = {}
    count2 = {}
    for num in nums1:
        if num in count1:
            count1[num] += 1
        else:
            count1[num] = 1
    for num in nums2:
        if num in count2:
            count2[num] += 1
        else:
            count2[num] = 1
    result = []
    for num in count1:
        if num in count2:
            result.extend([num] * min(count1[num], count2[num]))
    return result

# Example usage:
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
print(intersect(nums1, nums2))  # Output: [2, 2]

nums1 = [4, 9, 5]
nums2 = [9, 4, 9, 8, 4]
print(intersect(nums1, nums2))  # Output: [4, 9]
```

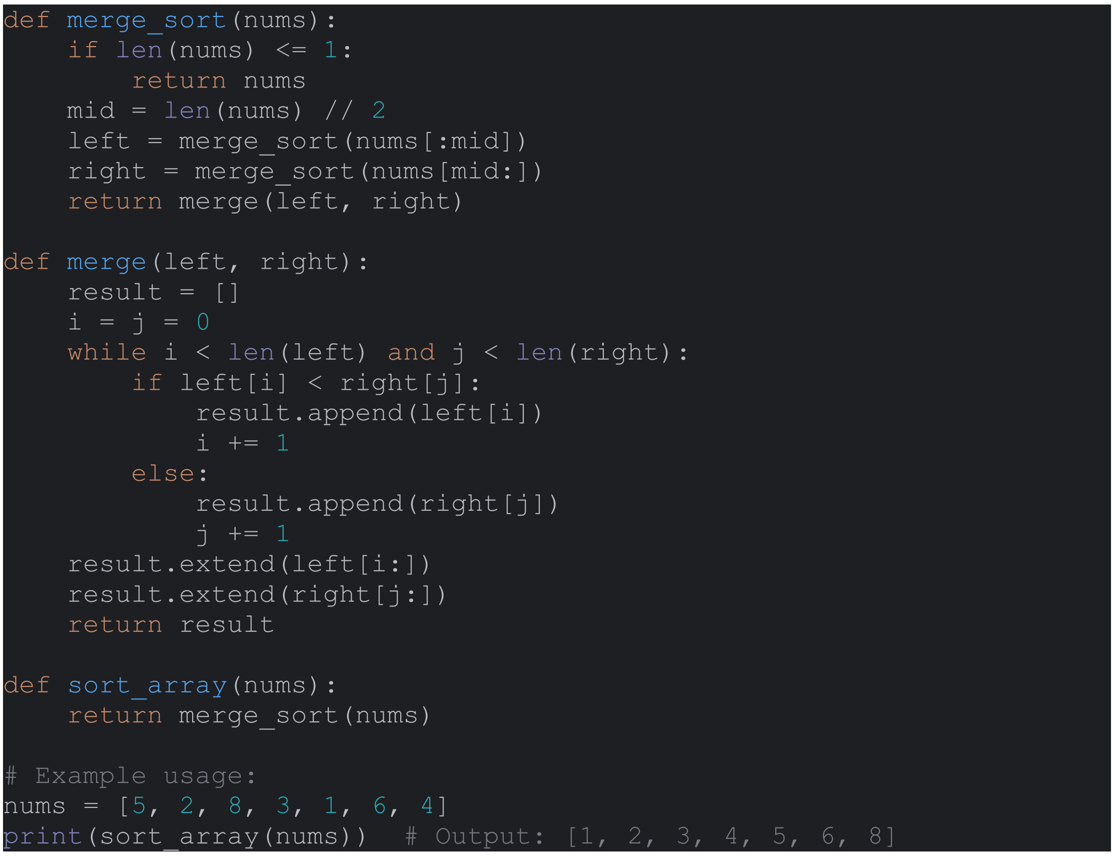
Output:



```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
 "C:\Users\saisr\Downloads\assignments\assignment2 part2\intersection
 duplicates.py"
[2, 2]
[4, 9]
```

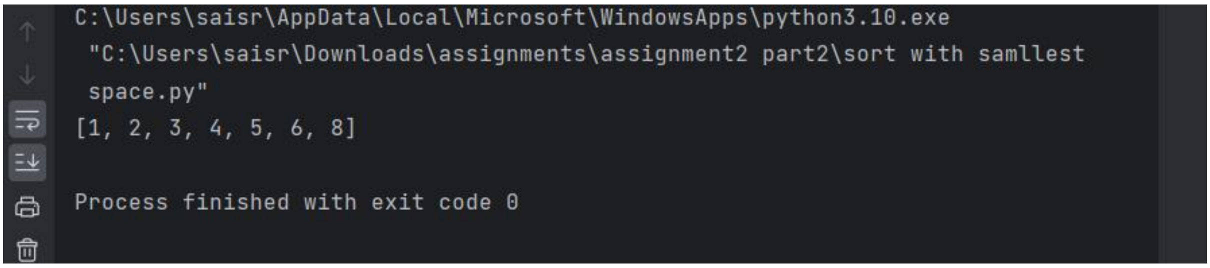
8. **Given an array of integers nums, sort the array in ascending order and return it.You must solve the problem without using any built-in functions in O(nlog(n)) time complexity and with the smallest space complexity possible.**

Coding:



```
def merge_sort(nums):
    if len(nums) <= 1:
        return nums
    mid = len(nums) // 2
    left = merge_sort(nums[:mid])
    right = merge_sort(nums[mid:])
    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

def sort_array(nums):
    return merge_sort(nums)

# Example usage:
nums = [5, 2, 8, 3, 1, 6, 4]
print(sort_array(nums))  # Output: [1, 2, 3, 4, 5, 6, 8]
```

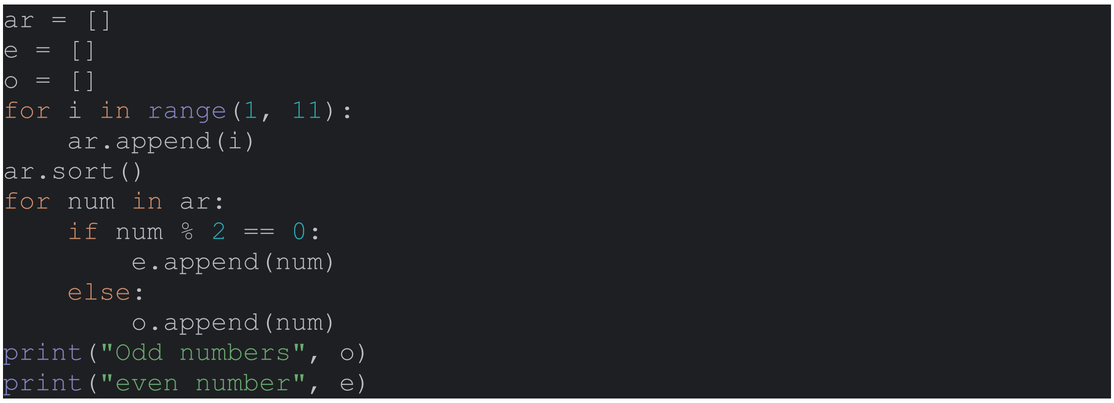
Output:



```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
 "C:\Users\saisr\Downloads\assignments\assignment2 part2\sort with samllest
 space.py"
[1, 2, 3, 4, 5, 6, 8]

Process finished with exit code 0
```

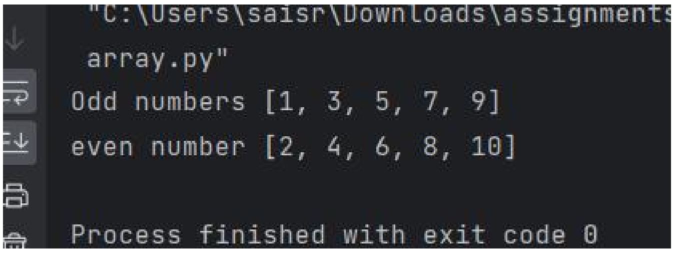
9. **Given an array of integers nums, half of the integers in nums are odd, and the other half are even. Sort the array so that whenever nums[i] is odd, i is odd, and whenever nums[i] is even, i is even. Return any answer array that satisfies this condition.**

Coding:



```
ar = []
e = []
o = []
for i in range(1, 11):
    ar.append(i)
ar.sort()
for num in ar:
    if num % 2 == 0:
        e.append(num)
    else:
        o.append(num)
print("Odd numbers", o)
print("even number", e)
```

Output:



```
 "C:\Users\saisr\Downloads\assignments
 array.py"
Odd numbers [1, 3, 5, 7, 9]
even number [2, 4, 6, 8, 10]

Process finished with exit code 0
```