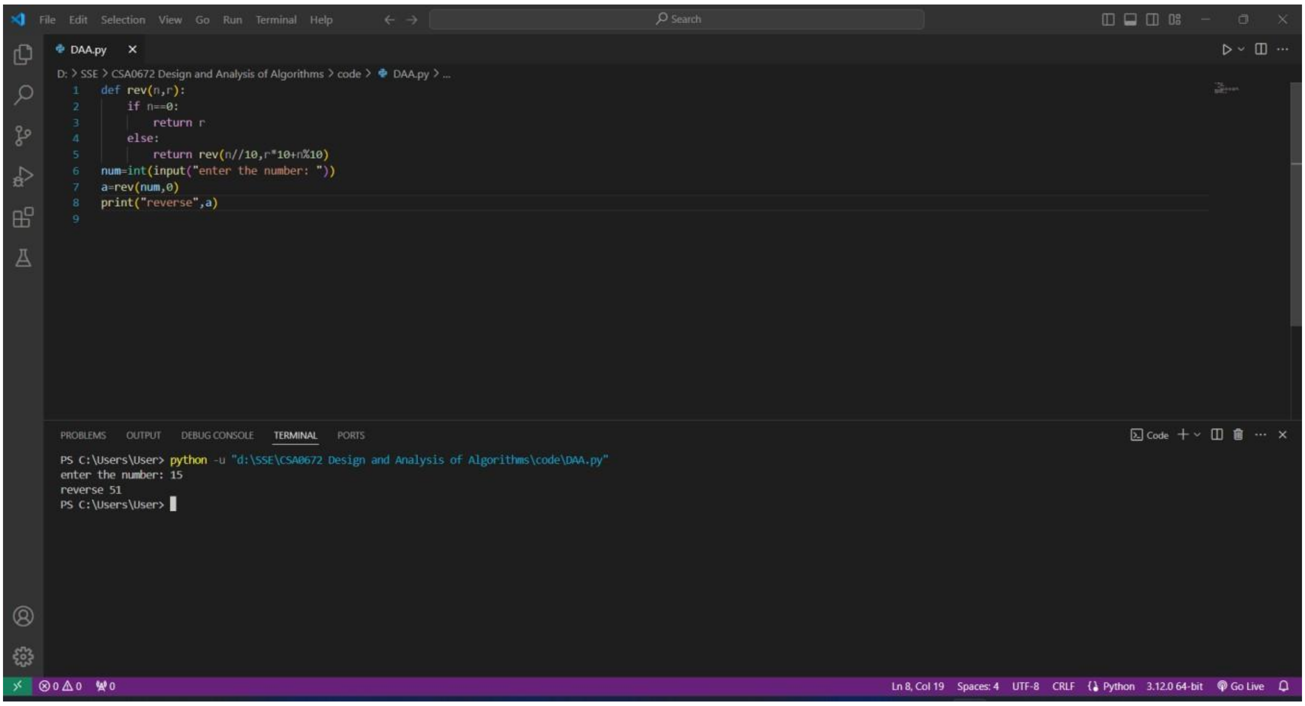
1. Write a program to find the reverse of a given number using recursive.

Code:

```
def rev(n,r):
    if n==0:
        return r
    else:
        return rev(n//10,r*10+n%10)
num=int(input("enter the number: "))
a=rev(num,0)
print("reverse",a)
```

Screenshot for I/O:



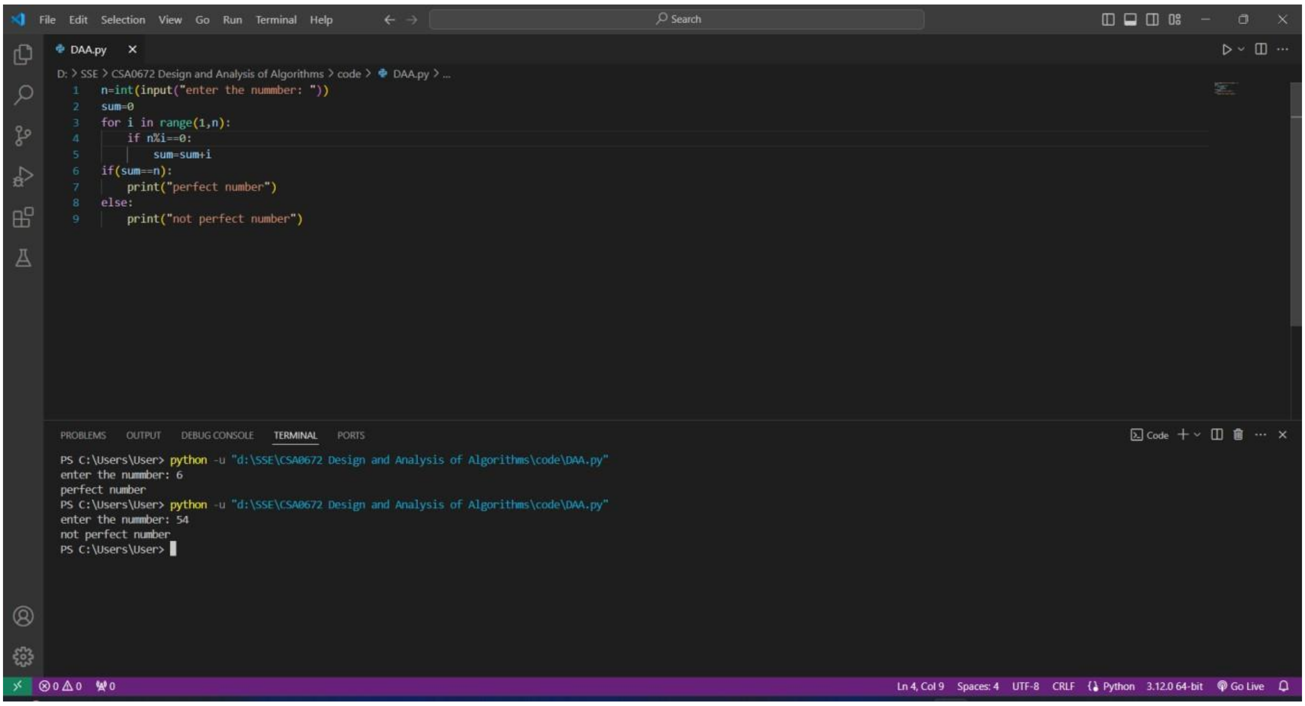
Time Complexity:O(n)

2. Write a program to find the perfect number.

Code:

```
n=int(input("enter the nummber: "))
sum=0
for i in range(1,n):
    if n%i==0:
        sum=sum+i
if(sum==n):
    print("perfect number")
else:
    print("not perfect number")
```

Screenshot for I/O:



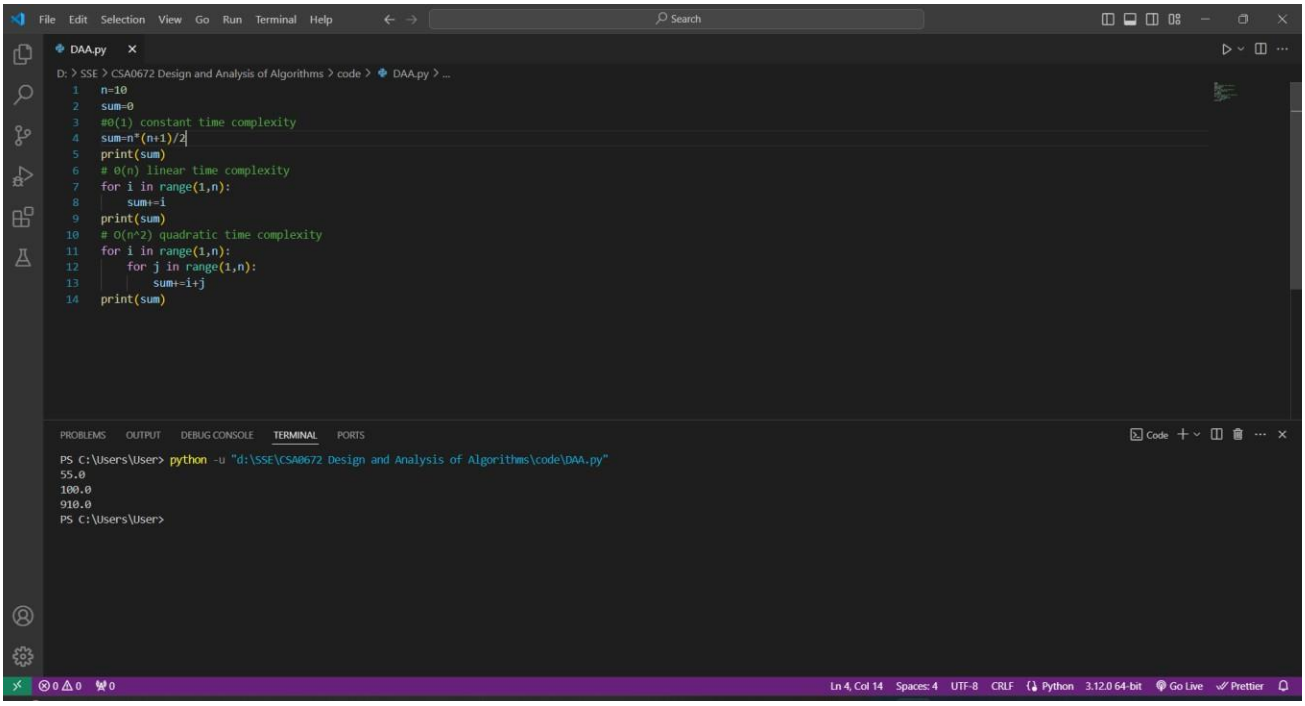
Time Complexity:O(n)

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

Code:

```
n=10
sum=0
#0(1) constant time complexity
sum=n*(n+1)/2
print(sum)
# 0(n) linear time complexity
for i in range(1,n):
    sum+=i
print(sum)
# O(n^2) quadratic time complexity
for i in range(1,n):
    for j in range(1,n):
        sum+=i+j
print(sum)
```

Screenshot for I/O:



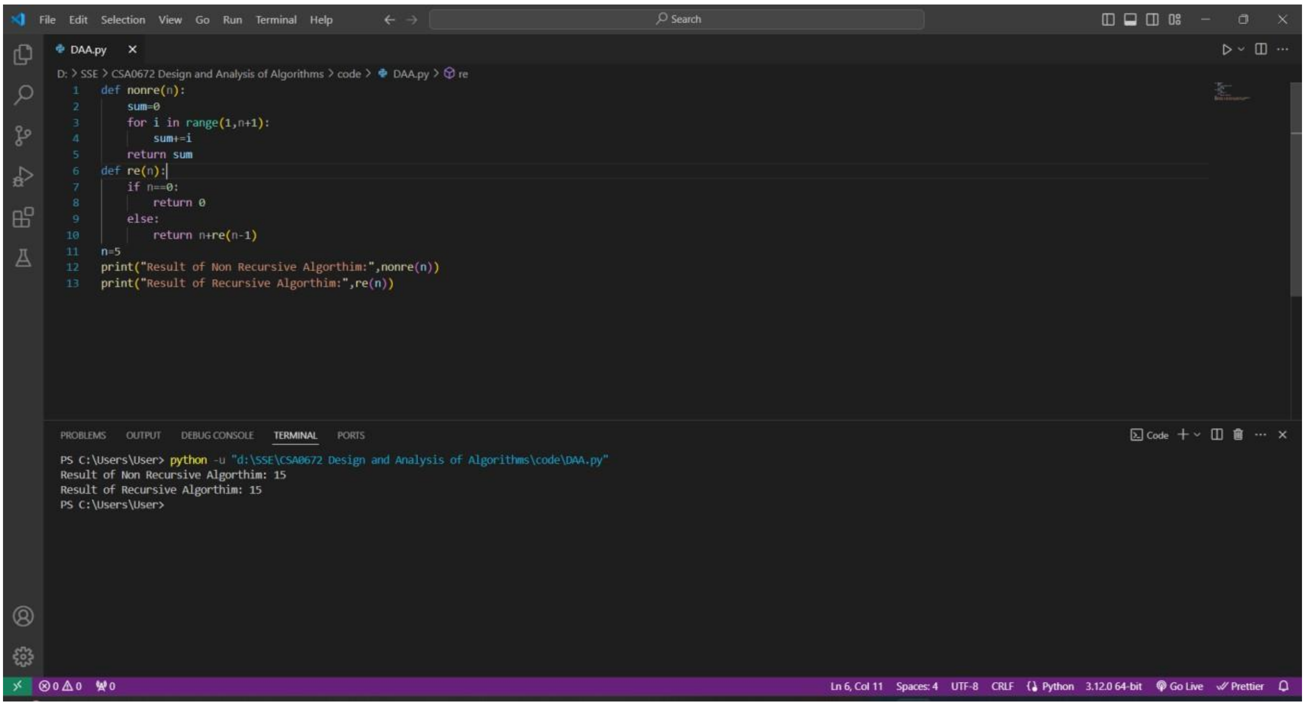
Time Complexity: O(n^2)

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

Code:

```
def nonre(n):
    sum=0
    for i in range(1,n+1):
        sum+=i
    return sum
def re(n):
    if n==0:
        return 0
    else:
        return n+re(n-1)
n=5
print("Result of Non Recursive Algorthim:",nonre(n))
print("Result of Recursive Algorthim:",re(n))
```

Screenshot for I/O:



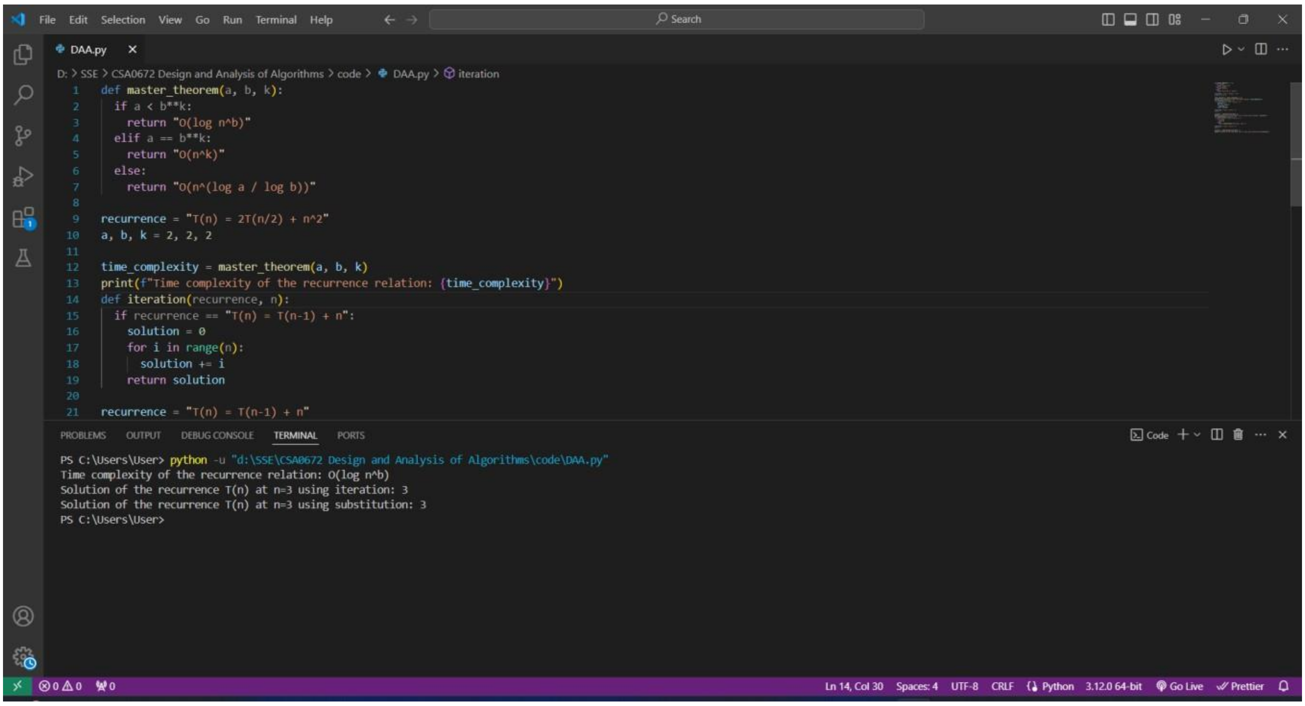
Time Complexity: O(n)

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.

Code:

```
def master_theorem(a, b, k):
  if a < b**k:
    return "O(log n^b)"
  elif a == b**k:
    return "O(n^k)"
  else:
    return "O(n^(log a / log b))"

recurrence = "T(n) = 2T(n/2) + n^2"
a, b, k = 2, 2, 2

time_complexity = master_theorem(a, b, k)
print(f"Time complexity of the recurrence relation:
{time_complexity}")
def iteration(recurrence, n):
  if recurrence == "T(n) = T(n-1) + n":
    solution = 0
    for i in range(n):
      solution += i
    return solution

recurrence = "T(n) = T(n-1) + n"
n = 3

solution = iteration(recurrence, n)
print(f"Solution of the recurrence T(n) at n={n} using iteration:
{solution}")
def substitution(recurrence, n):
  if recurrence == "T(n) = T(n-1) + 1":
    if n == 0:
      return 0
    else:
      return substitution(recurrence, n-1) + 1

recurrence = "T(n) = T(n-1) + 1"
n = 3

solution = substitution(recurrence, n)
print(f"Solution of the recurrence T(n) at n={n} using
substitution: {solution}")
```

Screenshot for I/O:



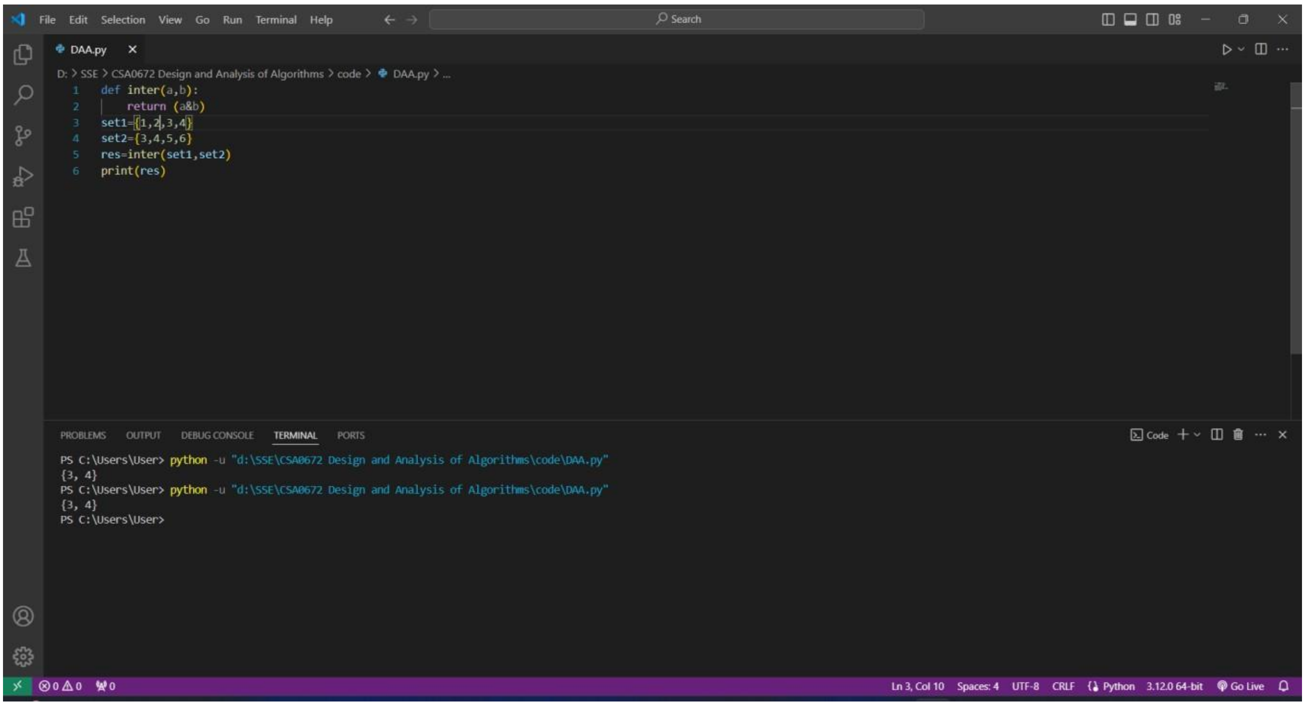
Time Complexity: O(n)

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.

Code:

```
def inter(a,b):
    return (a&b)
set1={1,2,3,4}
set2={3,4,5,6}
res=inter(set1,set2)
print(res)
```

Screenshot for I/O:



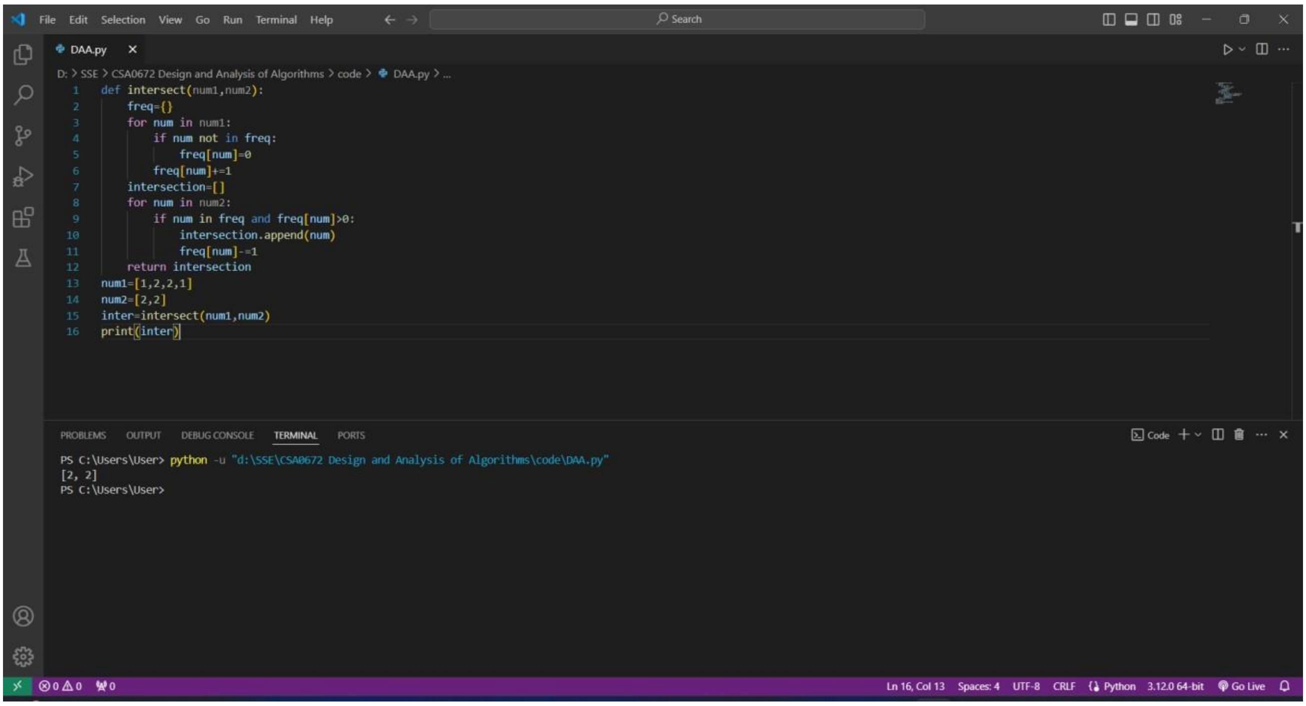
Time Complexity: O(n)

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.

Code:

```
def intersect(num1,num2):
    freq={}
    for num in num1:
        if num not in freq:
            freq[num]=0
        freq[num]+=1
    intersection=[]
    for num in num2:
        if num in freq and freq[num]>0:
            intersection.append(num)
            freq[num]-=1
    return intersection
num1=[1,2,2,1]
num2=[2,2]
inter=intersect(num1,num2)
print(inter)
```

Screenshot for I/O:



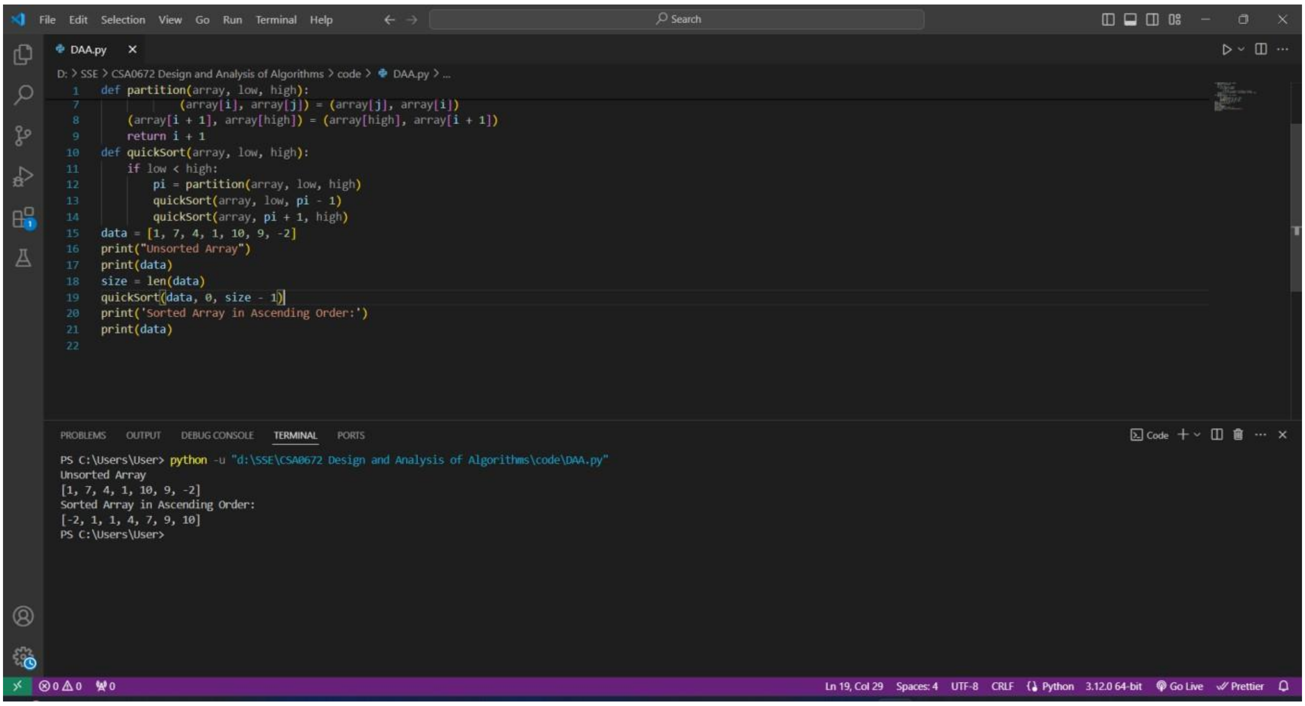
Time Complexity: O(n*m)

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.

Code:

```
def partition(array, low, high):
    pivot = array[high]
    i = low - 1
    for j in range(low, high):
        if array[j] <= pivot:
            i = i + 1
            (array[i], array[j]) = (array[j], array[i])
    (array[i + 1], array[high]) = (array[high], array[i + 1])
    return i + 1
def quickSort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quickSort(array, low, pi - 1)
        quickSort(array, pi + 1, high)
data = [1, 7, 4, 1, 10, 9, -2]
print("Unsorted Array")
print(data)
size = len(data)
quickSort(data, 0, size - 1)
print('Sorted Array in Ascending Order:')
print(data)
```

Screenshot for I/O:

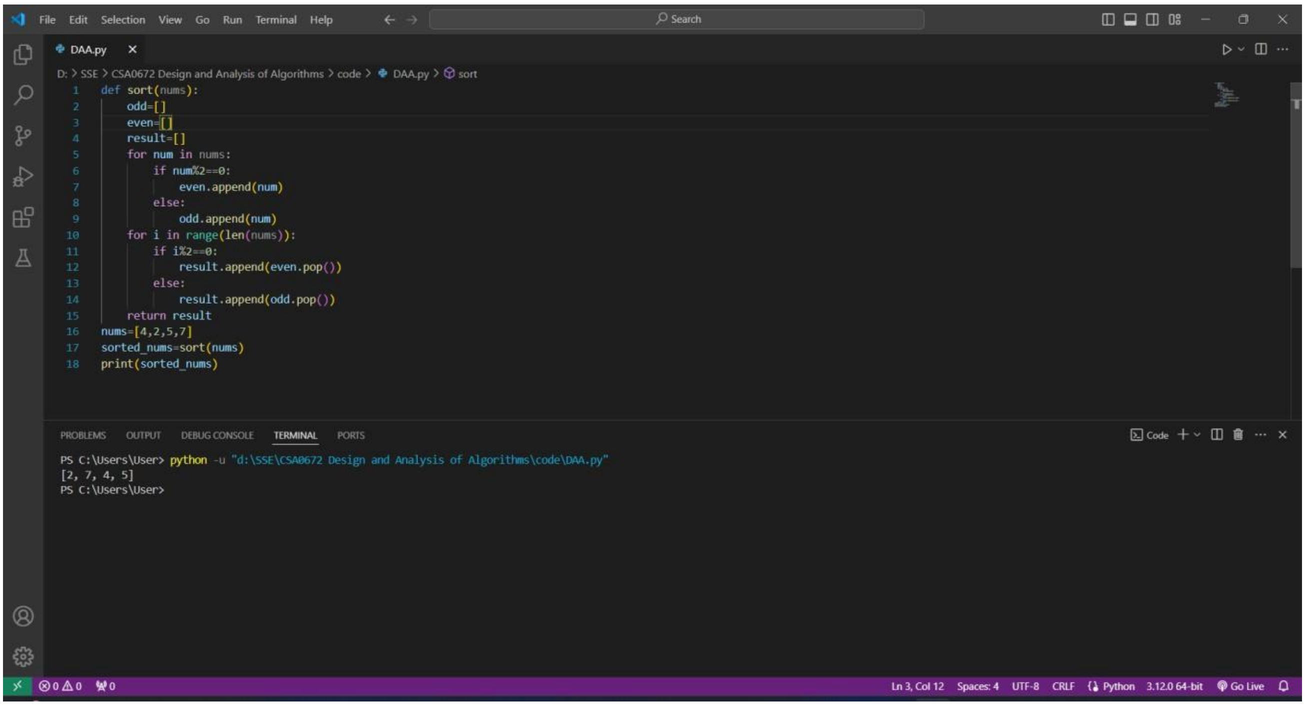


9. Given an array of integers nums, half of the integers in nums are odd, and the other half are even.

Code:

```
def sort(nums):
    odd=[]
    even=[]
    result=[]
    for num in nums:
        if num%2==0:
            even.append(num)
        else:
            odd.append(num)
    for i in range(len(nums)):
        if i%2==0:
            result.append(even.pop())
        else:
            result.append(odd.pop())
    return result
nums=[4,2,5,7]
sorted_nums=sort(nums)
print(sorted_nums)
```

Screenshot for I/O:



Time Complexity: O(n*m)