DAA\_CSA0676\_EXERCISE-2

11) Write a program to find the reverse of a given number using recursive

CODE:

def rev(num):

n=0

while num>0:

r=num%10

n=(n\*10)+r

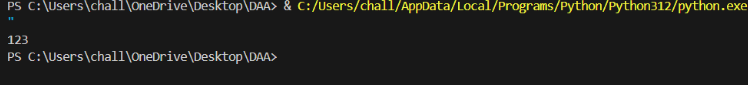
num=num//10

return n

a=321

print(rev(a))

OUTPUT:



TIME COMPLEXITY : O(d)

12.Write a program to find the perfect number.

PROGRAM:

def per(num):

sum=0

for i in range(1,num):

if num%i==0:

sum+=i

if sum==num:

return True

else:

return False

a=28

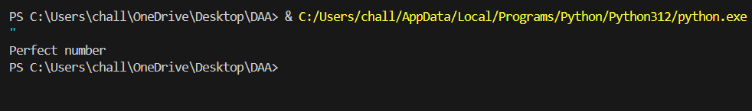
if per(a):

print("Perfect number")

else:

print("Not a perfect number")

OUTPUT:



TIME COMPLEXITY: Time complexity for the above code is

F(n)=O(n)

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

PROGRAM:

import time

def linears(a,target):

start\_time=time.time()

for num in a:

if num==target:

break

end\_time=time.time()

return end\_time,start\_time

def binarys(a,target):

start\_time=time.time()

low=0

high=len(a)-1

while low<=high:

mid=(low+high)//2

if a[mid]==target:

break

elif a[mid]<target:

low=mid+1

else:

high=mid-1

end\_time=time.time()

return end\_time,start\_time

a=list(range(100000))

target=999999

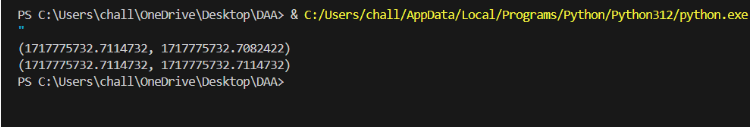
l=linears(a,target)

b=binarys(a,target)

print(l)

print(b)

OUTPUT:



TIME COMPLEXITY:

Time complexity for the above code is

O(n)+O(logn)

14. Write C programs that demonstrate the mathematical analysis of nonrecursive and recursive algorithms

PROGRAM:

def factorial\_iterative(n):

result = 1

for i in range(1, n + 1):

result \*= i

return result

def fibonacci\_recursive(n):

if n <= 1:

return n

else:

return fibonacci\_recursive(n-1) + fibonacci\_recursive(n-2)

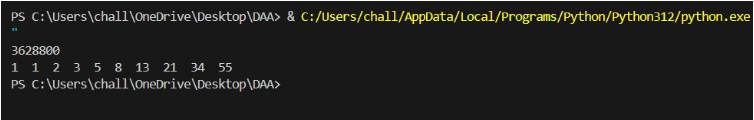
a=10

print(factorial\_iterative(a))

for i in range(1,a+1):

print(fibonacci\_recursive(i)," ",end="")

OUTPUT:



TIME COMPLEXITY:

Time complexity for the above code is

O(n)+O(2n)

15. Given an array of integers half of int in nums are even and half are odd

PROGRAM:

a=[1,2,3,4,5,6]

even=[]

odd=[]

for i in range(len(a)):

if a[i]%2==0:

even.append(a[i])

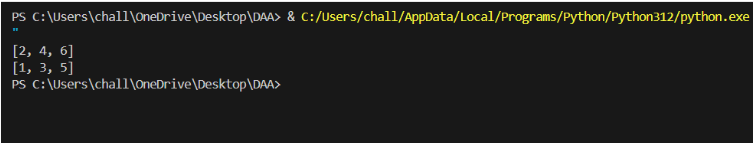
else:

odd.append(a[i])

print(even)

print(odd)

OUTPUT:



TIME COMPLEXITY:

Time complexity for the above code is

F(n)=O(n)

16. 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.

PROGRAM:

def inter(a,b):

a1=set(a)

b1=set(b)

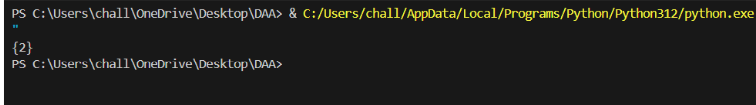
return a1.intersection(b1)

a=[1,2,2,3]

b=[2,2]

print(inter(a,b))

OUTPUT:



TIME COMPLEXITY:

Time complexity for the above code is

F(n)=O(n)

17. 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.

PROGRAM:

def inter(a,b):

c=[]

for i in a:

if i in b:

c.append(i)

return c

a=[1,2,2,4]

b=[2,2]

print(inter(a,b))

OUTPUT:

A screen shot of a computer

Description automatically generated

TIME COMPLEXITY:

Time complexity for the above code is

F(n)=O(m+n)

Where M=length of list a

N=length of list b

18. 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.

PROGRAM:

def sorta(a):

odd=sorted([x for x in a if x%2!=0])

even=sorted([x for x in a if x%2==0])

sorted\_nums=[0]\*len(a)

sorted\_nums[::2]=even

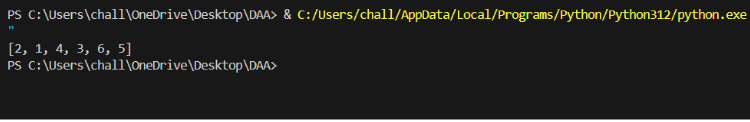
sorted\_nums[1::2]=odd

return sorted\_nums

a=[1,2,3,4,5,6]

print(sorta(a))

OUTPUT:



TIME COMPLEXITY:

Time complexity for the above code is

F(n)=O(nlogn)

19. 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.

PROGRAM:

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left = merge\_sort(arr[:mid])

right = merge\_sort(arr[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:])

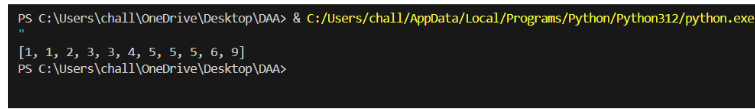
return result

nums = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

sorted\_nums = merge\_sort(nums)

print(sorted\_nums)

OUTPUT:



TIME COMPLEXITY:

Time complexity for the above code is

F(n)=O(nlogn)

20.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.

PROGRAM:

def master\_theorem(a, b, k):

if a > b\*\*k:

return "O(n^log\_b(a))"

elif a == b\*\*k:

return "O(n^k \* log(n))"

else:

return "O(n^k)"

def substitution\_method(t,n):

if n == 0:

return 1

else:

return 2 \* substitution\_method(t,n-1) + 1

def iteration\_method(t,n):

result = 0

for i in range(n):

result += 2\*\*i

return result

a = 2

b = 2

k = 1

t=2

n=5

master\_theorem\_result = master\_theorem(a, b, k)

substitution\_method\_result = substitution\_method(t,n)

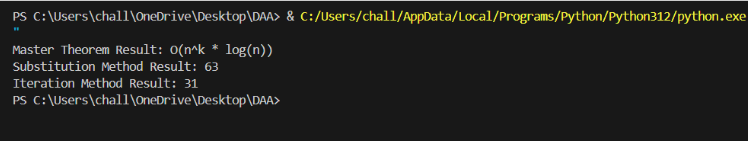
iteration\_method\_result = iteration\_method(t,n)

print("Master Theorem Result:", master\_theorem\_result)

print("Substitution Method Result:", substitution\_method\_result)

print("Iteration Method Result:", iteration\_method\_result)

OUTPUT:



TIME COMPLEXITY:

Time complexity for the above code is

F(n)=O(nlogn)+O(2n)+O(n)