**Day 2 programs**

1.Reverse number:

def reverse\_number(num, rev=0):

if num == 0:

return rev

else:

return reverse\_number(num // 10, rev \* 10 + num % 10)

num = 12345

print(reverse\_number(num))

2.Perfect number:

def is\_perfect(n):

sum\_divisors = 0

for i in range(1, n // 2 + 1):

if n % i == 0:

sum\_divisors += i

return sum\_divisors == n

n = 28

print(is\_perfect(n))

3.Demonstrate Usage of Big-O Notation:

def constant\_time(n):

return n + 1

def linear\_time(arr):

total = 0

for num in arr:

total += num

return total

def quadratic\_time(arr):

for i in range(len(arr)):

for j in range(len(arr)):

print(i, j)

print(constant\_time(5))

print(linear\_time([1, 2, 3, 4, 5]))

quadratic\_time([1, 2, 3])

4.Mathematical Analysis of Non-Recursive and Recursive Algorithms:

def linear\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

def factorial(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n - 1)

print(linear\_search([1, 2, 3, 4, 5], 4))

print(factorial(5))

5.Solving Recurrence relations:

def master\_theorem(n):

if n == 1:

return 1

return 2 \* master\_theorem(n // 2) + n

print(master\_theorem(8))

def substitution\_method(n):

if n == 1:

return 1

return substitution\_method(n - 1) + 1

print(substitution\_method(5))

def iteration\_method(n):

if n == 1:

return 1

return 2 \* iteration\_method(n // 2) + n

6.Intersection unique:

def intersection\_unique(nums1, nums2):

return list(set(nums1) & set(nums2))

nums1 = [1, 2, 2, 1]

nums2 = [2, 2]

print(intersection\_unique(nums1, nums2))

from collections import Counter

7.intersection multiset:

def intersection\_multiset(nums1, nums2):

counts1 = Counter(nums1)

counts2 = Counter(nums2)

intersection = []

for num in counts1:

if num in counts2:

intersection.extend([num] \* min(counts1[num], counts2[num]))

return intersection

nums1 = [1, 2, 2, 1]

nums2 = [2, 2]

print(intersection\_multiset(nums1, nums2))

8.merge sort:

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

L = arr[:mid]

R = arr[mid:]

merge\_sort(L)

merge\_sort(R)

i = j = k = 0

while i < len(L) and j < len(R):

if L[i] < R[j]:

arr[k] = L[i]

i += 1

else:

arr[k] = R[j]

j += 1

k += 1

while i < len(L):

arr[k] = L[i]

i += 1

k += 1

while j < len(R):

arr[k] = R[j]

j += 1

k += 1

return arr

nums = [5, 2, 3, 1]

print(merge\_sort(nums))

9.sort half\_odd and half\_even:

def sort\_half\_odd\_half\_even(nums):

odd\_index = 0

even\_index = 1

n = len(nums)

while odd\_index < n and even\_index < n:

if nums[odd\_index] % 2 == 0:

while even\_index < n and nums[even\_index] % 2 == 0:

even\_index += 2

if even\_index < n:

nums[odd\_index], nums[even\_index] = nums[even\_index], nums[odd\_index]

odd\_index += 2

return nums

nums = [4, 1, 2, 3, 6, 7, 8, 5]

print(sort\_half\_odd\_half\_even(nums))

10.sorted array:

def sort\_array\_by\_parity(nums):

odd\_index = 1

even\_index = 0

n = len(nums)

while odd\_index < n and even\_index < n:

if nums[even\_index] % 2 == 0:

even\_index += 2

elif nums[odd\_index] % 2 == 1:

odd\_index += 2

else:

nums[even\_index], nums[odd\_index] = nums[odd\_index], nums[even\_index]

even\_index += 2

odd\_index += 2

return nums

nums = [4, 1, 2, 3, 6, 7, 8, 5]

print(sort\_array\_by\_parity(nums))