1.FIND MAXIMUM AND MINIMUM

def find\_max\_min(lst):

max\_value = min\_value = lst[0]

for num in lst:

if num > max\_value:

max\_value = num

elif num < min\_value:

min\_value = num

return max\_value, min\_value

my\_list = [3, 1, 4, 1, 5, 9, 2, 6, 5]

max\_val, min\_val = find\_max\_min(my\_list)

print("Maximum value:", max\_val)

print("Minimum value:", min\_val)

OUTPUT: Maximum value: 9

Minimum value: 1

2.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

if \_\_name\_\_ == "\_\_main\_\_":

arr = [12, 11, 13, 5, 6, 7]

print("Given array is:", arr)

merge\_sort(arr)

print("Sorted array is:", arr)

OUTPUT:

Given array is: [12, 11, 13, 5, 6, 7]

Sorted array is: [5, 6, 7, 11, 12, 13]

3.QUICK SORT

def quick\_sort(arr):

if len(arr) <= 1:

return arr

else:

pivot = arr[len(arr) // 2]

left = [x for x in arr if x < pivot]

middle = [x for x in arr if x == pivot]

right = [x for x in arr if x > pivot]

return quick\_sort(left) + middle + quick\_sort(right)

if \_\_name\_\_ == "\_\_main\_\_":

arr = [12, 4, 5, 6, 7, 3, 1, 15]

print("Given array is:", arr)

sorted\_arr = quick\_sort(arr)

print("Sorted array is:", sorted\_arr)

OUTPUT:

Given array is: [12, 4, 5, 6, 7, 3, 1, 15]

Sorted array is: [1, 3, 4, 5, 6, 7, 12, 15]

4.BINARY SEARCH

def binary\_search(arr, target):

left, right = 0, len(arr) - 1

while left <= right:

mid = (left + right) // 2

if arr[mid] == target:

return mid

elif arr[mid] < target:

left = mid + 1

else:

right = mid - 1

return -1

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

target = 5

index = binary\_search(arr, target)

print("Index of target:", index)

OUTPUT:

Index of target: 4

5.STRASSENS MATRIX MULTIPLICATION

import numpy as np

def strassen(A, B):

n = len(A)

if n == 1:

return A \* B

else:

mid = n // 2

A11, A12, A21, A22 = A[:mid, :mid], A[:mid, mid:], A[mid:, :mid], A[mid:, mid:]

B11, B12, B21, B22 = B[:mid, :mid], B[:mid, mid:], B[mid:, :mid], B[mid:, mid:]

M1 = strassen(A11 + A22, B11 + B22)

M2 = strassen(A21 + A22, B11)

M3 = strassen(A11, B12 - B22)

M4 = strassen(A22, B21 - B11)

M5 = strassen(A11 + A12, B22)

M6 = strassen(A21 - A11, B11 + B12)

M7 = strassen(A12 - A22, B21 + B22)

C11 = M1 + M4 - M5 + M7

C12 = M3 + M5

C21 = M2 + M4

C22 = M1 - M2 + M3 + M6

C = np.vstack((np.hstack((C11, C12)), np.hstack((C21, C22))))

return C

A = np.array([[1, 2], [3, 4]])

B = np.array([[5, 6], [7, 8]])

C = strassen(A, B)

print("Resultant Matrix:\n", C)

6. Karatsuba Algorithm for Multiplication

def karatsuba(x, y):

if x < 10 or y < 10:

return x \* y

m = min(len(str(x)), len(str(y))) // 2

high1, low1 = divmod(x, 10\*\*m)

high2, low2 = divmod(y, 10\*\*m)

z0 = karatsuba(low1, low2)

z1 = karatsuba((low1 + high1), (low2 + high2))

z2 = karatsuba(high1, high2)

return (z2 \* 10\*(2\*m)) + ((z1 - z2 - z0) \* 10\*m) + z0

x, y = 1234, 5678

result = karatsuba(x, y)

print("Product:", result)

OUTPUT:

Product: 11052

7. Closest Pair of Points using Divide and Conquer

import math

def closest\_pair(points):

def dist(p1, p2):

return math.sqrt((p1[0] - p2[0]) \*\* 2 + (p1[1] - p2[1]) \*\* 2)

def closest\_pair\_rec(px, py):

if len(px) <= 3:

return min((dist(px[i], px[j]), (px[i], px[j])) for i in range(len(px)) for j in range(i + 1, len(px)))[1]

mid = len(px) // 2

Qx, Rx = px[:mid], px[mid:]

midpoint = px[mid][0]

Qy, Ry = [], []

for point in py:

if point[0] <= midpoint:

Qy.append(point)

else:

Ry.append(point)

(p1, q1) = closest\_pair\_rec(Qx, Qy)

(p2, q2) = closest\_pair\_rec(Rx, Ry)

d = min(dist(p1, q1), dist(p2, q2))

(p3, q3) = closest\_split\_pair(px, py, d)

if p3 is not None and q3 is not None:

return min((p1, q1), (p2, q2), (p3, q3), key=lambda x: dist(\*x))

return min((p1, q1), (p2, q2), key=lambda x: dist(\*x))

def closest\_split\_pair(px, py, delta):

midx = px[len(px) // 2][0]

sy = [p for p in py if midx - delta <= p[0] <= midx + delta]

best = delta

best\_pair = None

for i in range(len(sy) - 1):

for j in range(i + 1, min(i + 7, len(sy))):

p, q = sy[i], sy[j]

d = dist(p, q)

if d < best:

best = d

best\_pair = (p, q)

return best\_pair if best\_pair else (None, None)

px = sorted(points, key=lambda x: x[0])

py = sorted(points, key=lambda x: x[1])

return closest\_pair\_rec(px, py)

points = [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]

closest\_points = closest\_pair(points)

print("Closest pair of points:", closest\_points)

OUTPUT:

Closest pair of points: ((2, 3), (3, 4))

8. Median of Medians

def partition(arr, low, high):

pivot = arr[high]

i = low

for j in range(low, high):

if arr[j] <= pivot:

arr[i], arr[j] = arr[j], arr[i]

i += 1

arr[i], arr[high] = arr[high], arr[i]

return i

def select(arr, low, high, k):

if low == high:

return arr[low]

pivot\_index = partition(arr, low, high)

if k == pivot\_index:

return arr[k]

elif k < pivot\_index:

return select(arr, low, pivot\_index - 1, k)

else:

return select(arr, pivot\_index + 1, high, k)

def median\_of\_medians(arr, k):

n = len(arr)

if n <= 5:

return sorted(arr)[k]

medians = [sorted(arr[i:i + 5])[2] for i in range(0, n, 5)]

pivot = median\_of\_medians(medians, len(medians) // 2)

pivot\_index = arr.index(pivot)

arr[pivot\_index], arr[-1] = arr[-1], arr[pivot\_index]

return select(arr, 0, n - 1, k)

arr = [12, 3, 5, 7, 4, 19, 26]

k = 3

median = median\_of\_medians(arr, k)

print(f"{k}th smallest element:", median)

9. Meet in the Middle Technique

from itertools import combinations

def meet\_in\_the\_middle(arr, target):

n = len(arr)

first\_half = arr[:n//2]

second\_half = arr[n//2:]

def get\_all\_sums(subset):

sums = []

for r in range(len(subset) + 1):

for combo in combinations(subset, r):

sums.append(sum(combo))

return sums

sums\_first\_half = get\_all\_sums(first\_half)

sums\_second\_half = get\_all\_sums(second\_half)

sums\_first\_half.sort()

sums\_second\_half.sort()

l = 0

r = len(sums\_second\_half) - 1

closest\_sum = float('inf')

while l < len(sums\_first\_half) and r >= 0:

current\_sum = sums\_first\_half[l] + sums\_second\_half[r]

if abs(current\_sum - target) < abs(closest\_sum - target):

closest\_sum = current\_sum

if current\_sum < target:

l += 1

else:

r -= 1

return closest\_sum

arr = [3, 34, 4, 12, 5, 2]

target = 9

closest\_sum = meet\_in\_the\_middle(arr, target)

print("Closest sum to target:", closest\_sum)

OUTPUT:

Closest sum to target: 9