1.Finding the maximum and minimum

numbers = [3, 7, 2, 8, 1, 5]

max\_num = max(numbers)

min\_num = min(numbers)

print(f"Maximum number: {max\_num}")

print(f"Minimum number: {min\_num}")

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

return arr

# Example usage

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

print("Given array is", arr)

merge\_sort(arr)

print("Sorted array is", arr)

3. Quick sort

def quick\_sort(arr):

if len(arr) <= 1:

return arr

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)

# Example

arr = [3, 6, 8, 10, 1, 2, 1]

print(quick\_sort(arr))

4. Binary search

def binary\_search(arr, target):

low = 0

high = len(arr) - 1

while low <= high:

mid = (low + high) // 2

if arr[mid] < target:

low = mid + 1

elif arr[mid] > target:

high = mid - 1

else:

return mid

return -1

5. Strassens matrix multiplication

def strassen\_matrix\_multiply(X, Y):

if len(X) == 1:

return [[X[0][0] \* Y[0][0]]]

n = len(X) // 2

A = [row[:n] for row in X[:n]]

B = [row[n:] for row in X[:n]]

C = [row[:n] for row in X[n:]]

D = [row[n:] for row in X[n:]]

E = [row[:n] for row in Y[:n]]

F = [row[n:] for row in Y[:n]]

G = [row[:n] for row in Y[n:]]

H = [row[n:] for row in Y[n:]]

P1 = strassen\_matrix\_multiply(A, [[F[i][j] - H[i][j] for j in range(n)] for i in range(n)])

P2 = strassen\_matrix\_multiply([[A[i][j] + B[i][j] for j in range(n)] for i in range(n)], H)

P3 = strassen\_matrix\_multiply([[C[i][j] + D[i][j] for j in range(n)] for i in range(n)], E)

P4 = strassen\_matrix\_multiply(D, [[G[i][j] - E[i][j] for j in range(n)] for i in range(n)])

P5 = strassen\_matrix\_multiply([[A[i][j] + D[i][j] for j in range(n)] for i in range(n)], [[E[i][j] + H[i][j] for j in range(n)] for i in range(n)])

P6 = strassen\_matrix\_multiply([[B[i][j] - D[i][j] for j in range(n)] for i in range(n)], [[G[i][j] + H[i][j] for j in range(n)] for i in range(n)])

P7 = strassen\_matrix\_multiply([[A[i][j] - C[i][j] for j in range(n)] for i in range(n)], [[E[i][j] + F[i][j] for j in range(n)] for i in range(n)])

Q = [[P5[i][j] + P4[i][j] - P2[i][j] + P6[i][j] for j in range(n)] for i in range(n)]

R = [[P1[i][j] + P2[i][j] for j in range(n)] for i

6. Karatsuba algorithm for multiplication

m = max(len(str(x)), len(str(y)))

m2 = m // 2

high1, low1 = divmod(x, 10\*\*m2)

high2, low2 = divmod(y, 10\*\*m2)

z0 = karatsuba(low1, low2)

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

z2 = karatsuba(high1, high2)

return (z2 \* 10\*\*(2\*m2)) + ((z1 - z2 - z0) \* 10\*\*m2) + z0

7. Closest pair of points using divide and conquer

import math

def dist(p1, p2):

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

def brute\_force(points, n):

min\_dist = float('inf')

for i in range(n):

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

if dist(points[i], points[j]) < min\_dist:

min\_dist = dist(points[i], points[j])

return min\_dist

def closest\_pair(points):

points.sort(key=lambda x: x[0])

return closest\_pair\_util(points, len(points))

def closest\_pair\_util(points, n):

if n <= 3:

return brute\_force(points, n)

mid = n // 2

mid\_point = points[mid]

dl = closest\_pair\_util(points[:mid], mid)

dr = closest\_pair\_util(points[mid:], n - mid)

d = min(dl, dr)

strip = []

for point in points:

if abs(point[0] - mid\_point[0]) < d:

strip.append(point)

strip.sort(key=lambda x: x[1])

min\_strip = float('inf')

for i in range(len(strip)):

j = i + 1

while j < len(strip) and (strip[j][1] - strip[i][1]) < d:

min\_strip = min(min\_strip, dist(strip[i], strip[j]))

j += 1

return min(d, min\_strip)

# Example Usage

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

print("The smallest distance is", closest\_pair(points))

8. Median of medians

def median\_of\_medians(arr):

sublists = [arr[j:j+5] for j in range(0, len(arr), 5)]

medians = [sorted(sublist)[len(sublist)//2] for sublist in sublists]

if len(medians) <= 5:

pivot = sorted(medians)[len(medians)//2]

else:

pivot = median\_of\_medians(medians)

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

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

if len(lower) == len(arr)//2:

return pivot

elif len(lower) > len(arr)//2:

return median\_of\_medians(lower)

else:

return median\_of\_medians(upper)

# Example Usage

arr = [3, 6, 8, 10, 1, 2, 5, 4, 7, 9]

result = median\_of\_medians(arr)

print("Median of the list is:", result)

9. Meet in middle technique

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

n = len(arr)

result = []

for i in range(1 << n):

subset = [arr[j] for j in range(n) if (i & (1 << j))]

if sum(subset) == target:

result.append(subset)

return result

arr = [3, 1, 7, 5, 9, 2]

target = 10

print(meet\_in\_middle(arr, target))