**1)Finding the maximum and minimum.**

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

max\_num = max(numbers)

min\_num = min(numbers)

print("Maximum number:", max\_num)

print("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)

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(A, B):

n = len(A)

if n == 1:

return [[A[0][0] \* B[0][0]]]

mid = n // 2

A11 = [row[:mid] for row in A[:mid]]

A12 = [row[mid:] for row in A[:mid]]

A21 = [row[:mid] for row in A[mid:]]

A22 = [row[mid:] for row in A[mid:]]

B11 = [row[:mid] for row in B[:mid]]

B12 = [row[mid:] for row in B[:mid]]

B21 = [row[:mid] for row in B[mid:]]

B22 = [row[mid:] for row in B[mid:]]

P1 = strassen\_matrix\_multiply(A11, subtract\_matrices(B12, B22))

P2 = strassen\_matrix\_multiply(add\_matrices(A11, A12), B22)

P3 = strassen\_matrix\_multiply(add\_matrices(A21, A22), B11)

P4 = strassen\_matrix\_multiply(A22, subtract\_matrices(B21, B11))

P5 = strassen\_matrix\_multiply(add\_matrices(A11, A22), add\_matrices(B11, B22))

P6 = strassen\_matrix\_multiply(subtract\_matrices(A12, A22), add\_matrices(B21, B22))

P7 = strassen\_matrix\_multiply(subtract\_matrices(A11, A21), add\_matrices(B11, B12))

C11 = add\_matrices(subtract\_matrices(add\_matrices(P5, P4), P2), P6)

C12 = add\_matrices(P1, P2)

C21 = add\_matrices(P3, P4)

C22 = subtract\_matrices(subtract\_matrices(add\_matrices(P5, P1), P3), P7)

result = [[0 for \_ in range(n)] for \_ in range(n)]

for i in range(mid):

for j in range(mid):

result[i][j] = C11[i][j]

result[i][j + mid] = C12[i][j]

result[i + mid][j] = C21[i][j]

result[i + mid][j + mid] = C22[i][j]

return result

**6) Karatsuba algorithm for multiplication**

def karatsuba(x, y):

if x < 10 or y < 10:

return x \* y

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 conque**

import math

def closest\_pair(points):

def distance(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 distance(points[i], points[j]) < min\_dist:

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

return min\_dist

def closest\_util(px, py, n):

if n <= 3:

return brute\_force(px, n)

mid = n // 2

mid\_point = px[mid]

pyl = [point for point in py if point[0] < mid\_point[0]]

pyr = [point for point in py if point[0] >= mid\_point[0]]

dl = closest\_util(px, pyl, mid)

dr = closest\_util(px[mid:], pyr, n - mid)

d = min(dl, dr)

strip = [point for point in py if abs(point[0] - mid\_point[0]) < d]

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]) < min\_strip:

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

j += 1

return min(d, min\_strip)

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

px = points.copy()

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

py = points.copy()

return closest\_util(px, py, len(points))

# Example Usage

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

print(closest\_pair(points))

**8) Median of medians**

import statistics

def median\_of\_medians(arr):

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

medians = [statistics.median(sublist) for sublist in sublists]

if len(medians) <= 5:

pivot = statistics.median(medians)

else:

pivot = median\_of\_medians(medians)

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

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

return pivot, lower, upper

# Example

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

pivot, lower, upper = median\_of\_medians(arr)

print("Pivot:", pivot)

print("Lower values:", lower)

print("Upper values:", upper)

**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))