1.**Finding maximum or minimum**

**Code:-**

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

**# Find the maximum number**

**max\_num = max(numbers)**

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

**# Find the minimum number**

**min\_num = min(numbers)**

**print("Minimum number:", min\_num)**

**2. Quick sort**

**Code:-**

**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)**

**3.Merge sort**

**Code:-**

**def merge\_sort(arr):**

**if len(arr) <= 1:**

**return arr**

**mid = len(arr) // 2**

**left = arr[:mid]**

**right = arr[mid:]**

**left = merge\_sort(left)**

**right = merge\_sort(right)**

**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**

**while i < len(left):**

**result.append(left[i])**

**i += 1**

**while j < len(right):**

**result.append(right[j])**

**j += 1**

**return result**

**4.Binary search**

**Code:-**

**def binary\_search(arr, x):**

**low = 0**

**high = len(arr) - 1**

**mid = 0**

**while low <= high:**

**mid = (high + low) // 2**

**# If x is greater, ignore left half**

**if arr[mid] < x:**

**low = mid + 1**

**# If x is smaller, ignore right half**

**elif arr[mid] > x:**

**high = mid - 1**

**# means x is present at mid**

**else:**

**return mid**

**# If we reach here, then the element was not present**

**return -1**

**# Test array**

**arr = [2, 3, 4, 10, 40]**

**x = 10**

**# Function call**

**result = binary\_search(arr, x)**

**if result != -1:**

**print("Element is present at index", str(result))**

**else:**

**print("Element is not present in array")**

**5.Strassens matrix multiplication**

**Code:-**

**def strassen\_matrix\_multiply(A, B):**

**n = len(A)**

**if n == 1:**

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

**new\_size = n // 2**

**A11 = [row[:new\_size] for row in A[:new\_size]]**

**A12 = [row[new\_size:] for row in A[:new\_size]]**

**A21 = [row[:new\_size] for row in A[new\_size:]]**

**A22 = [row[new\_size:] for row in A[new\_size:]]**

**B11 = [row[:new\_size] for row in B[:new\_size]]**

**B12 = [row[new\_size:] for row in B[:new\_size]]**

**B21 = [row[:new\_size] for row in B[new\_size:]]**

**B22 = [row[new\_size:] for row in B[new\_size:]]**

**S1 = [[B12[i][j] - B22[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**S2 = [[A11[i][j] + A12[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**S3 = [[A21[i][j] + A22[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**S4 = [[B21[i][j] - B11[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**S5 = [[A11[i][j] + A22[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**S6 = [[B11[i][j] + B22[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**S7 = [[A12[i][j] - A22[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**S8 = [[B21[i][j] + B22[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**S9 = [[A11[i][j] - A21[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**S10 = [[B11[i][j] + B12[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**P1 = strassen\_matrix\_multiply(A11, S1)**

**P2 = strassen\_matrix\_multiply(S2, B22)**

**P3 = strassen\_matrix\_multiply(S3, B11)**

**P4 = strassen\_matrix\_multiply(A22, S4)**

**P5 = strassen\_matrix\_multiply(S5, S6)**

**P6 = strassen\_matrix\_multiply(S7, S8)**

**P7 = strassen\_matrix\_multiply(S9, S10)**

**C11 = [[P5[i][j] + P4[i][j] - P2[i][j] + P6[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**C12 = [[P1[i][j] + P2[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**C21 = [[P3[i][j] + P4[i][j] for j in range(new\_size)] for i in range(new\_size)]**

**C22 = [[P5[i][j] + P1[i][j] - P3[i][j] - P7[i][j] for j in range(new\_size)] for i in range(new\_size)]**

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

**for i in range(new\_size):**

**for j in range(new\_size):**

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

**result[i][j + new\_size] = C12[i][j]**

**result[i + new\_size][j] = C21[i][j]**

**result[i + new\_size][j + new\_size] = C22[i][j]**

**return result**

**6.Karatsuba algorithm for multiplication**

**Code:-**

**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 conquer rule**

**Code:-**

**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):**

**min\_dist = float('inf')**

**for i in range(len(points)):**

**for j in range(i + 1, len(points)):**

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

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

**return min\_dist**

**def closest\_split\_pair(p\_x, p\_y, delta, best\_pair):**

**mid\_x = p\_x[len(p\_x) // 2][0]**

**s\_y = [x for x in p\_y if mid\_x - delta <= x[0] <= mid\_x + delta]**

**best = delta**

**for i in range(len(s\_y) - 1):**

**for j in range(i + 1, min(i + 7, len(s\_y))):**

**p, q = s\_y[i], s\_y[j]**

**dst = distance(p, q)**

**if dst < best:**

**best\_pair = p, q**

**best = dst**

**return best\_pair[0], best\_pair[1], best**

**def closest\_pair\_rec(p\_x, p\_y):**

**if len(p\_x) <= 3:**

**return brute\_force(p\_x)**

**mid = len(p\_x) // 2**

**Qx = p\_x[:mid]**

**Rx = p\_x[mid:]**

**midpoint = p\_x[mid][0]**

**Qy = []**

**Ry = []**

**for x in p\_y:**

**if x[0] <= midpoint:**

**Qy.append(x)**

**else:**

**Ry.append(x)**

**(p1, q1, delta1) = closest\_pair\_rec(Qx, Qy)**

**(p2, q2, delta2) = closest\_pair\_rec(Rx, Ry)**

**delta = min(delta1, delta2)**

**best\_pair = (p1, q1) if delta1 < delta2 else (p2, q2)**

**(p3, q3, delta3) = closest\_split\_pair(p\_x, p\_y, delta, best\_pair)**

**return min((p1, q1, delta1), (p2, q2, delta2), (p3, q3, delta3), key=lambda x: x[2])**

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

**p\_x = points.copy()**

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

**p\_y = points.copy()**

**return closest\_pair\_rec(p\_x, p\_y)**

**# Example Usage**

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

**print(closest\_pair(points))**

**8.Median of medians**

**Code:-**

**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]**

**if len(lower) == 5:**

**return pivot**

**elif len(lower) > 5:**

**return median\_of\_medians(lower)**

**else:**

**return median\_of\_medians(upper)**

**# Example Usage**

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

**result = median\_of\_medians(arr)**

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

**9.Meet in middle technique:-**

**Code:**

**def meet\_in\_the\_middle(target, nums):**

**def subset\_sums(nums):**

**res = []**

**for i in range(1 << len(nums)):**

**res.append(sum(nums[j] for j in range(len(nums)) if (i & (1 << j)) > 0))**

**return res**

**n = len(nums) // 2**

**left\_half = subset\_sums(nums[:n])**

**right\_half = subset\_sums(nums[n:])**

**right\_half.sort()**

**count = 0**

**for sum\_val in left\_half:**

**left = 0**

**right = len(right\_half) - 1**

**while left < len(right\_half) and right >= 0:**

**if sum\_val + right\_half[right] == target:**

**count += 1**

**left += 1**

**right -= 1**

**elif sum\_val + right\_half[right] < target:**

**left += 1**

**else:**

**right -= 1**

**return count**

**# Example Usage**

**target\_sum = 10**

**numbers = [1, 2, 3, 4, 5]**

**result = meet\_in\_the\_middle(target\_sum, numbers)**

**print(result)**