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)
```