1. Finding the maximum and minimum

```
def maxi(arr, low, high):
  if low == high:
    return arr[low], arr[low]
  mid = (low + high) // 2
  lmin, lmax = maxi(arr, low, mid)
  rmin, rmax = maxi(arr, mid + 1, high)
  return min(lmin, rmin), max(lmax, rNmax)
arr = [10, 2, 6, 7, 4, 1, 9]
min_element, max_element = maxi(arr, 0, len(arr) - 1)
print("Minimum element:", min_element)
print("Maximum element:", max_element)
```

output

```
Minimum element: 1
Maximum element: 10
```

2. Merge sort

```
def merge(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]

        merge(L)
        merge(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

arr = [12, 11, 13, 5, 6, 7]
merge(arr)
```

```
print("Sorted array is:", arr)

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

3. Quick sort

```
def quick(arr):
    if len(arr) <= 1:
        return arr

    p = arr[len(arr) // 2]
    l = [x for x in arr if x < p]
    m = [x for x in arr if x == p]
    r = [x for x in arr if x > p]

    return quick(l) + m + quick(r)

arr = [3, 6, 8, 10, 1, 2, 1]
s = quick(arr)
print("Sorted array:", s)

output
Sorted array: [1, 1, 2, 3, 6, 8, 10]
```

3. Binary search

```
def binary(arr, t):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2

        if arr[mid] == t:
            return mid
        elif arr[mid] < t:
            left = mid + 1
        else:
            right = mid - 1

    return -1

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
target = 4
index = binary(arr, target)
print(f"Element {target} found at index {index}")

output
Element 4 found at index 3
```

4. Strassens matrix multiplication

```
def add_matrix(A, B):
    return [[A[i][j] + B[i][j] for j in range(len(A[0]))] for i in range(len(A))]

def sub_matrix(A, B):
    return [[A[i][j] - B[i][j] for j in range(len(A[0]))] for i in range(len(A))]

def strassen(A, B):
    if len(A) == 1:
        return [[A[0][0] * B[0][0]]]


    mid = len(A) // 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(add_matrix(A11, A22), add_matrix(B11, B22))
    P2 = strassen(add_matrix(A21, A22), B11)
    P3 = strassen(A11, sub_matrix(B12, B22))
    P4 = strassen(A22, sub_matrix(B21, B11))
    P5 = strassen(add_matrix(A11, A12), B22)
    P6 = strassen(sub_matrix(A21, A11), add_matrix(B11, B12))
    P7 = strassen(sub_matrix(A12, A22), add_matrix(B21, B22))


    C11 = add_matrix(sub_matrix(add_matrix(P1, P4), P5), P7)
    C12 = add_matrix(P3, P5)
    C21 = add_matrix(P2, P4)
    C22 = add_matrix(sub_matrix(add_matrix(P1, P3), P2), P6)


    C = []
    for i in range(mid):
        C.append(C11[i] + C12[i])
    for i in range(mid):
        C.append(C21[i] + C22[i])

    return C

A = [
```

```
    [1, 2, 3,4],
    [5, 6, 7, 8],
    [1,2,3,4],
    [5,6,7,8]
]

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

C = strassen(A, B)
for row in C:
    print(row)

output
[10, 60, 10, 70]
[26, 140, 26, 166]
[10, 60, 10, 70]
[26, 140, 26, 166]
```

5. Karatsuba algorithm for multiplication

```
def karatsuba(x, y):
    if x < 10 or y < 10:
        return x * y

    n = max(len(str(x)), len(str(y)))
    m = n // 2

    x1, x0 = divmod(x, 10**m)
    y1, y0 = divmod(y, 10**m)

    z2 = karatsuba(x1, y1)
    z0 = karatsuba(x0, y0)
    z1 = karatsuba(x1 + x0, y1 + y0) - z2 - z0

    return z2 * 10**(2*m) + z1 * 10**m + z0
x = 1234
y = 5678
result = karatsuba(x, y)
print("Product:", result)

output
Product: 7006652
```

6. Closest pair of points using divide and conquer

```
def closest_pair(points):
    def distance(p1, p2):
        return ((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2) ** 0.5

    def closest_pair_recursive(px, py):
        if len(px) <= 3:
            return brute_force(px)

        mid = len(px) // 2
        Qx = px[:mid]
        Rx = px[mid:]

        midpoint = px[mid][0]
        Qy = list(filter(lambda x: x[0] <= midpoint, py))
        Ry = list(filter(lambda x: x[0] > midpoint, py))

        (p1, q1) = closest_pair_recursive(Qx, Qy)
        (p2, q2) = closest_pair_recursive(Rx, Ry)

        if distance(p1, q1) < distance(p2, q2):
            d = distance(p1, q1)
            min_pair = (p1, q1)
        else:
            d = distance(p2, q2)
            min_pair = (p2, q2)

        strip = [p for p in py if abs(p[0] - midpoint) < d]
        for i in range(len(strip)):
            for j in range(i + 1, min(i + 7, len(strip))):
                if distance(strip[i], strip[j]) < d:
                    d = distance(strip[i], strip[j])
                    min_pair = (strip[i], strip[j])

        return min_pair

    def brute_force(points):
        min_dist = float('inf')
        min_pair = None
        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])
                    min_pair = (points[i], points[j])
        return min_pair

    px = sorted(points, key=lambda x: x[0])
    py = sorted(points, key=lambda x: x[1])
```

```
    return closest_pair_recursive(px, py)
points = [(2.1, 3.3), (3.0, 1.5), (1.4, 2.7), (4.7, 3.6), (3.4, 0.8)]
p1, p2 = closest_pair(points)
print("Closest pair:", p1, p2)

output
Closest pair: (3.0, 1.5) (3.4, 0.8)
```

7. Median of medians

```
def median_of_medians(arr, k):
    if len(arr) <= 5:
        return sorted(arr)[k]

    sublists = [arr[i:i + 5] for i in range(0, len(arr), 5)]
    medians = [sorted(sublist)[len(sublist) // 2] for sublist in sublists]

    pivot = median_of_medians(medians, len(medians) // 2)

    low = [x for x in arr if x < pivot]
    high = [x for x in arr if x > pivot]
    pivots = [x for x in arr if x == pivot]

    if k < len(low):
        return median_of_medians(low, k)
    elif k < len(low) + len(pivots):
        return pivot
    else:
        return median_of_medians(high, k - len(low) - len(pivots))

arr = [12, 3, 5, 7, 4, 19, 26]
k = 3
result = median_of_medians(arr, k)
print("k-th smallest element:", result)

output
k-th smallest element: 7
```

8. Meet in 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:]

    first_half_sums = set(sum(comb) for r in range(len(first_half)+1) for comb in
combinations(first_half, r))
    second_half_sums = set(sum(comb) for r in range(len(second_half)+1) for comb in
combinations(second_half, r))
```

```
    for s in first_half_sums:
        if target - s in second_half_sums:
            return True
    return False
arr = [3, 34, 4, 12, 5, 2]
target = 9
result = meet_in_the_middle(arr, target)
print("Subset with given sum exists:", result)
```

output

Subset with given sum exists: True