```
def find_max_min(arr):
    max_val = arr[0]
    min_val = arr[0]

    for num in arr[1:]:
        if num > max_val:
            max_val = num
        elif num < min_val:
            min_val = num

    return max_val, min val
arr = [3, 5, 1, 2, 4, 8]
max_val, min_val = find_max_min(arr)
print(f"Maximum value: {max_val}, Minimum value: {min_val}")

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
arr = [12, 11, 13, 5, 6, 7]
merge_sort(arr)
print(f"Sorted array: {arr}")

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(f"Sorted array: {quick_sort(arr)}")

def binary_search(arr, x):
    left, right = 0, len(arr) - 1
    while left <= right:
```

```
        mid = (left + right) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            left = mid + 1
        else:
            right = mid - 1
    return -1

arr = [2, 3, 4, 10, 40]
x = 10
result = binary_search(arr, x)
print(f"Element {x} is at index {result}")

import numpy as np

def strassen(A, B):
    n = len(A)
    if n == 1:
        return A * B
    else:
        mid = n // 2
        A11, A12, A21, A22 = A[:mid, :mid], A[:mid, mid:], A[mid:, :mid], A[mid:, mid:]
        B11, B12, B21, B22 = B[:mid, :mid], B[:mid, mid:], B[mid:, :mid], B[mid:, mid:]

        M1 = strassen(A11 + A22, B11 + B22)
        M2 = strassen(A21 + A22, B11)
        M3 = strassen(A11, B12 - B22)
        M4 = strassen(A22, B21 - B11)
        M5 = strassen(A11 + A12, B22)
        M6 = strassen(A21 - A11, B11 + B12)
```

```
        M7 = strassen(A12 - A22, B21 + B22)

        C11 = M1 + M4 - M5 + M7
        C12 = M3 + M5
        C21 = M2 + M4
        C22 = M1 - M2 + M3 + M6

        C = np.vstack((np.hstack((C11, C12)), np.hstack((C21, C22))))
        return C
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
print(f"Strassen's Matrix Multiplication result:\n{strassen(A, B)}")

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

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

    high1, low1 = divmod(x, 10**m)
    high2, low2 = divmod(y, 10**m)

    z0 = karatsuba(low1, low2)
    z1 = karatsuba((low1 + high1), (low2 + high2))
    z2 = karatsuba(high1, high2)

    return (z2 * 10**(2*m)) + ((z1 - z2 - z0) * 10**m) + z0
x = 1234
y = 5678
print(f"Karatsuba multiplication result: {karatsuba(x, y)}")
```

```
import math

def dist(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def closest_pair_of_points(points):
    def closest_pair_rec(points_sorted_x, points_sorted_y):
        if len(points_sorted_x) <= 3:
            return min([dist(points_sorted_x[i], points_sorted_x[j]) for i in range(len(points_sorted_x)) for j in range(i+1, len(points_sorted_x))], default=float('inf'))

        mid = len(points_sorted_x) // 2
        Qx = points_sorted_x[:mid]
        Rx = points_sorted_x[mid:]
        midpoint = points_sorted_x[mid][0]
        Qy = list(filter(lambda x: x[0] <= midpoint, points_sorted_y))
        Ry = list(filter(lambda x: x[0] > midpoint, points_sorted_y))

        delta = min(closest_pair_rec(Qx, Qy), closest_pair_rec(Rx, Ry))

        strip = [point for point in points_sorted_y if abs(point[0] - midpoint) < delta]

        min_dist = delta
        for i in range(len(strip)):
            for j in range(i+1, min(i+7, len(strip))):
                min_dist = min(min_dist, dist(strip[i], strip[j]))

        return min_dist

    points_sorted_x = sorted(points, key=lambda x: x[0])
    points_sorted_y = sorted(points, key=lambda x: x[1])
```

```
    return closest_pair_rec(points_sorted_x, points_sorted y)
points = [(2.1, 3.4), (12.3, 30.5), (40.1, 50.2), (5.0, 1.5), (3.4, 4.5)]
print(f"Closest pair distance: {closest_pair_of_points(points)}")

def partition(arr, low, high, pivot):
    i = low - 1
    for j in range(low, high):
        if arr[j] < pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
        elif arr[j] == pivot:
            arr[j], arr[high] = arr[high], arr[j]
            j -= 1
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

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

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

    pivot_index = partition(arr, 0, n - 1, median_of_median)

    if k < pivot_index:
        return median_of_medians(arr[:pivot_index], k)
    elif k > pivot_index:
        return median_of_medians(arr[pivot_index + 1:], k - pivot_index - 1)
    else:
```

```
        return arr[pivot_index]
arr = [12, 3, 5, 7, 4, 19, 26]
k = 2
print(median of medians)
```