GRAPH COLOURING:

```
def graph_coloring(adj_list):
    colors = {}
    max_color = 0

    for node in adj_list:
        neighbor_colors = set(colors.get(nei, 0) for nei in adj_list[node])
        color = 1
        while color in neighbor_colors:
            color += 1
        colors[node] = color
        max_color = max(max_color, color)

    return max_color

# Adjacency list representation of the graph
adj_list = {
    0: [1, 2, 3],
    1: [0, 2],
    2: [1, 3, 0],
    3: [2, 0]
}

max_regions_colored = graph_coloring(adj_list)
print(max_regions_colored)
```

MAXIMUM AND MINIMUM VALUE:

```
array1 = [2, 4, 6, 8, 10, 12, 14, 18]
```

```
array2 = [11,13,15,17,19,21,23,35,37]
min_value = min(array1)
max_value = max(array1)
min_value2 = min(array2)
max_value2 = max(array2)


print("Input Array:", array1)
print("Minimum Value:", min_value)
print("Maximum Value:", max_value)
print("Input Array:", array2)
print("Minimum Value:", min_value2)
print("Maximum Value:", max_value2)
```

ROBBERY:

```
def rob(nums):
    if not nums:
        return 0
    if len(nums) <= 2:
        return max(nums)

    def rob_helper(nums):
        dp = [0] * len(nums)
        dp[0] = nums[0]
        dp[1] = max(nums[0], nums[1])

        for i in range(2, len(nums)):
            dp[i] = max(dp[i-1], dp[i-2] + nums[i])
        return dp[-1]

    return max(rob_helper(nums[1:]), rob_helper(nums[:-1]))

# Test the function with example inputs
print(rob([2, 3, 2]))
print(rob([1, 2, 3, 1]))
```

DIJAKRASTRA’S:

```
import sys

def dijkstra(graph, source):
    n = len(graph)
    dist = [sys.maxsize] * n
    dist[source] = 0
    visited = [False] * n

    for _ in range(n):
        u = min_distance(dist, visited)
        visited[u] = True

        for v in range(n):
            if not visited[v] and graph[u][v] != sys.maxsize and dist[u] + graph[u][v] < dist[v]:
                dist[v] = dist[u] + graph[u][v]

    return dist

def min_distance(dist, visited):
    min_dist = sys.maxsize
    min_index = -1

    for v in range(len(dist)):
        if not visited[v] and dist[v] < min_dist:
            min_dist = dist[v]
            min_index = v

    return min_index

graph = [
    [0, 10, 3, sys.maxsize, sys.maxsize],
    [sys.maxsize, 0, 1, 2, sys.maxsize],
    [sys.maxsize, 4, 0, 8, 21],
    [sys.maxsize, sys.maxsize, sys.maxsize, 0, 6],
    [sys.maxsize, sys.maxsize, sys.maxsize, sys.maxsize, 0]
]
source = 0
result = dijkstra(graph, source)
print(result)

graph = [
    [0, 5, sys.maxsize, 10],
    [sys.maxsize, 0, 3, sys.maxsize],
    [sys.maxsize, sys.maxsize, 0, 1],
    [sys.maxsize, sys.maxsize, sys.maxsize, 0]
]
source = 3
result = dijkstra(graph, source)
print(result)
```

SELECTION SORT:

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

random_array = [52, 9, 1, 5, 6]
sorted_random_array = selection_sort(random_array)
print(sorted_random_array)



reverse_sorted_array = [12, 8, 6, 4, 2]
sorted_reverse_array = selection_sort(reverse_sorted_array)
print(sorted_reverse_array)

already_sorted_array = [1, 2, 3, 4, 5]
sorted_already_sorted_array = selection_sort(already_sorted_array)
print(sorted_already_sorted_array)
```

SEQUENTIAL SEARCH:

```
def findKthPositive(arr, k):
    missing = []
    i = 1
    while len(missing) < k:
        if i not in arr:
            missing.append(i)
        i += 1
    return missing[-1]



arr1 = [2, 3, 4, 7, 11]
k1 = 5
output1 = findKthPositive(arr1, k1)
print(output1)

arr2 = [1, 2, 3, 4, 14, 15]
k2 = 2
output2 = findKthPositive(arr2, k2)
print(output2)
```

BINARY SEARCH:

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0
    count = 0

    while low <= high:
        mid = (high + low) // 2
        count += 1

        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return mid, count

    return -1, count

arr = [5, 10, 15, 20, 25, 30, 35, 40, 45]
x = 20
result, comparisons = binary_search(arr, x)

print("Index of element 20:", result)
print("Number of comparisons made:", comparisons)
```

COMBINATION SUM:

```
def combinationSum(candidates, target):
    def backtrack(start, path, target):
        if target == 0:
            res.append(path[:])
            return
        for i in range(start, len(candidates)):
            if candidates[i] > target:
                continue
            path.append(candidates[i])
            backtrack(i, path, target - candidates[i])
            path.pop()

    res = []
    candidates.sort()
    backtrack(0, [], target)
    return res

candidates1 = [2, 3, 6, 7]
target1 = 7
print(combinationSum(candidates1, target1))

candidates2 = [2, 3, 5]
target2 = 8
print(combinationSum(candidates2, target2))

candidates3 = [2]
target3 = 1
print(combinationSum(candidates3, target3))
```

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


def print_list(arr):

    for i in range(len(arr)):

        print(arr[i], end=" ")

    print()


arr1 = [31, 23, 35, 27, 11, 21, 15, 28]

merge_sort(arr1)

print("Sorted array:")

print_list(arr1)


arr2 = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]

merge_sort(arr2)

print("Sorted array:")

print_list(arr2)
```

DIVIDE AND CONQUER:

```
import heapq

def kClosest(points, k):
    heap = []
    for x, y in points:
        dist = -(x*x + y*y)
        if len(heap) == k:
            heapq.heappushpop(heap, (dist, x, y))
        else:
            heapq.heappush(heap, (dist, x, y))
    return [(x, y) for (dist, x, y) in heap]

points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k1 = 2
print(kClosest(points1, k1))

points2 = [[1, 3], [-1, 2], [5, -1]]
k2 = 2
print(kClosest(points2, k2))

points3 = [[3, 3], [5, -1], [2, 4]]
k3 = 2
print(kClosest(points3, k3))
```