1.Graph colouring:

Program:

```
def is_safe(graph, vertex, color, c):
    for i in range(len(graph)):
        if graph[vertex][i] == 1 and color[i] == c:
            return False
    return True


def graph_coloring(graph, m, color, v):
    if v == len(graph):
        return True
    for c in range(1, m+1):
        if is_safe(graph, v, color, c):
            color[v] = c
            if graph_coloring(graph, m, color, v+1):
                return True
            color[v] = 0


    return False


def graph_coloring_solver(graph, m):
    color = [0] * len(graph)
    if not graph_coloring(graph, m, color, 0):
```

```
        print("No solution exists")
        return False
    print("Solution exists. Colors assigned to vertices:")
    print(color)
    return True


# Example Usage
graph = [[0, 1, 1, 1],
         [1, 0, 1, 0],
         [1, 1, 0, 1],
         [1, 0, 1, 0]]
colors = 3
graph_coloring_solver(graph, colors)
```

output:

```
[1, 2, 3, 2]
```

2. find minimum and maximum

Program:

```
a=[2,4,6,8,10,12,14,18]
a.sort()
print(a)
b=max(a)
c=min(a)
print("max :",b,"min :",c)
```

output:

max = 18

min = 2

3. Robbery planning

Program:

```
def maxLoot(hval,n):
   if (n < 0):
      return 0
   if (n == 0):
      return hval[0]
   pick = hval[n] + maxLoot(hval, n - 2)
   notPick = maxLoot(hval, n - 1)
   return max(pick, notPick)
hval = [1, 2, 3, 1]
n = len(hval)
print("Maximum loot possible : ",maxLoot(hval, n - 1));
```

output: 4

4. single source shortest path : dijkstra's algorithm:

Program:

```
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)
```

5. selection sort :

Program:

```
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


# Example Usage
my_list = [5, 2, 9, 1, 5, 6]
sorted_list = selection_sort(my_list)
print("Sorted list:", sorted_list)
```

output:

```
[1,  2,  5,  5,  6,  9]
```

6. sequential sort:

Program:

```
def sequential_search(target, lst):
    for i in range(len(lst)):
```

```
        if lst[i] == target:
            return i
    return -1

my_list = [1,2,3,4]
target_value = 2
result = sequential_search(target_value, my_list)
if result != -1:
    print(f"Target found at index: {result}")
else:
    print("Target not found in the list")
```

output: 1

7. binary search

Program:

```
def binary_search(arr, element):
  low = 0
  high = len(arr) - 1
  while low <= high:
    mid = (low + high) // 2
    if arr[mid] == element:
      return mid
    elif arr[mid] < element:
      low = mid + 1
```

```
    else:
      high = mid - 1
  return -1
arr = [5,10,15,20,25,30,35,40,45]
element_to_find = 20
index = binary_search(arr, element_to_find)
if index != -1:
  print(f"Element {element_to_find} found at index {index}")
else:
  print(f"Element {element_to_find} not found in the array")
```

output:

Element 20 found at index 3

8.Combination sum:

Program:

```
def combinationSum(candidates, target):
    def backtrack(start, path, target):
        if target == 0:
            result.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()

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

candidates = [2, 3, 6, 7]
target = 7
print(combinationSum(candidates, target))
```

output:

[[2, 2, 3], [7]]

9. Merge sort :

Program:

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
```

```
    right = merge_sort(arr[mid:])
    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
    result.extend(left[i:])
    result.extend(right[j:])
    return result

nums = [31, 23, 35, 27, 11, 21, 15, 28]
sorted_nums = merge_sort(nums)
print(sorted_nums)
```

output:

```
[11, 15, 21, 23, 27, 28, 31, 35]
```

10. Divide and conquer:

Program:

```
import math


def distance(point):
   return math.sqrt(point[0]*2 + point[1]*2)


def kClosest(points, k):
   points.sort(key=distance)
   return points[:k]


input_points = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k = 2
output_points = kClosest(input_points, k)
print("\n")
print(output_points)
print(output_points)
```

output:

```
[[-2, 2], [0, 1]]
[[-2, 2], [0, 1]]
```