#selectionsort

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]
my_list = [64, 25, 12, 22, 11]
selection_sort(my_list)
print("Sorted list:", my_list)
```

#MERGESORT

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left, right = arr[:mid], arr[mid:]
        merge_sort(left)
        merge_sort(right)
        merged = []
        i = j = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                merged.append(left[i])
                i += 1
            else:
                merged.append(right[j])
                j += 1
        merged.extend(left[i:])
        merged.extend(right[j:])
        arr[:] = merged
my_list = [64, 25, 12, 22, 11]
merge_sort(my_list)
print("Sorted list:", my_list)
```

#NQUEENS

```
from itertools import permutations
def solve_nq(n):
    for queens in permutations(range(n)):
        if all(abs(i - j) != abs(queens[i] - queens[j]) for i in range(n) for j in range(i + 1, n)):
            return queens
    return None
if __name__ == "__main__":
    solution = solve_nq(4)
    if solution:
        print(solution)
    else:
        print("Solution does not exist")
```

#QUICKSORT

```
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)
my_list = [99,11,22,66,77,44,1,22,3]
sorted_list = quick_sort(my_list)
print("Sorted list:", sorted_list)
```

#MAXMIN

```
def find_max_min(lst):
    if not lst:
        return None, None
    def helper(lst, start, end):
        if start == end:
            return lst[start], lst[start]
        mid = (start + end) // 2
        return max(left_max, right_max), min(left_min, right_min)
    return helper(lst, 0, len(lst) - 1)
my_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
max_value, min_value = find_max_min(my_list)
print("Maximum value:", max_value)
print("Minimum value:", min_value)
```

#ADJACENCYMATRIX

```
num_vertices = int(input("Enter the number of vertices: "))
num_edges = int(input("Enter the number of edges: "))
adjacency_matrix = [[0] * num_vertices for _ in range(num_vertices)]
print("Enter edges (vertex1 vertex2):")
for _ in range(num_edges):
    vertex1, vertex2 = map(int, input().split())
    adjacency_matrix[vertex1][vertex2] = 1
    adjacency_matrix[vertex2][vertex1] = 1
print("\nAdjacency Matrix:")
for row in adjacency_matrix:
    print(row)
```

#INDEGREEOURDEGREE

```
MAX = 100
def indegree_outdegree(adj, n):
    in_degree = [0] * MAX
    out_degree = [0] * MAX
    for i in range(n):
        for j in range(n):
            in_degree[i] += adj[j][i]
            out_degree[i] += adj[i][j]
    for i in range(n):
        print(f"Vertex: {i+1} In-Degree: {in_degree[i]} Out-Degree: {out_degree[i]}")
    print("Adjacency matrix is:")
    for i in range(n):
        for j in range(n):
            print(adj[i][j], end=" ")
        print()
print("Enter the number of vertices:")
n = int(input())
print("Enter the adjacency matrix (1 for edge, 0 for no edge):")
adj = []
for i in range(n):
    row = list(map(int, input().split()))
    adj.append(row)
indegree_outdegree(adj, n)
```

#BACKTRACKING

```
def subset_sum(nums, target_sum, s=[], c=0):
    if c == target_sum: result.append(s[:])
    for i, num in enumerate(nums):
        subset_sum(nums[i+1:], target_sum, s + [num], c + num)
result = []
subset_sum([2, 4, 6, 8, 10], 16)
print("Subsets with sum equal to 16 are:", result)
```

#JOBSEQ

```
def job_sequencing_with_deadline(jobs):
    jobs.sort(key=lambda x: x[2], reverse=True)
    result = []
    total_profit = 0
    for job in jobs:
        deadline = job[1]
        while deadline > 0 and result.count(None) < deadline:
            result.append(job[0])
            total_profit += job[2]
            break
    return result, total_profit
jobs = [('J1', 2, 60), ('J2', 1, 100), ('J3', 3, 20), ('J4', 2, 40)]
schedule, profit = job_sequencing_with_deadline(jobs)
print("Job Sequence:", schedule)
print("Total Profit:", profit)
```

#KNAPSACK

```
def knapsack(weights, profit, capacity):
    n = len(weights)
    dp = [0] * (capacity + 1)
    for i in range(n):
        for w in range(capacity, weights[i] - 1, -1):
            dp[w] = max(dp[w], dp[w - weights[i]] + profit[i])
    return dp[capacity]
weights = [10, 20, 30]
profit = [60, 100, 120]
capacity = 50
max_value = knapsack(weights, profit, capacity)
print("Maximum value:", max_value)
```