DAA Codes

1. Fibonacci Numbers (Recursive & Non-Recursive) :

```
# Non-Recursive Fibonacci

def fibonacci_iterative(n):

   fib = [0, 1]

   for i in range(2, n):

      fib.append(fib[i - 1] + fib[i - 2])

   return fib[:n]


# Recursive Fibonacci

def fibonacci_recursive(n):

   if n <= 1:

      return n

   else:

      return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)


# Example

n = 10

print("Iterative Fibonacci:", fibonacci_iterative(n))

print("Recursive Fibonacci:", [fibonacci_recursive(i) for i in range(n)])
```

2. Huffman Encoding (Greedy Strategy) :

```
import heapq

class Node:
  def __init__(self, char, freq):
    self.char = char
    self.freq = freq
    self.left = None
    self.right = None
  def __lt__(self, other):
    return self.freq < other.freq

def huffman_encoding(chars, freqs):
  heap = [Node(chars[i], freqs[i]) for i in range(len(chars))]
  heapq.heapify(heap)

  while len(heap) > 1:
    left = heapq.heappop(heap)
    right = heapq.heappop(heap)
    newNode = Node(None, left.freq + right.freq)
    newNode.left = left
    newNode.right = right
    heapq.heappush(heap, newNode)

  root = heap[0]
  codes = {}

  def generate_codes(node, code=""):
    if node:
      if node.char:
            codes[node.char] = code
        generate_codes(node.left, code + "0")
        generate_codes(node.right, code + "1")

    generate_codes(root)
    return codes

# Example
chars = ['a', 'b', 'c', 'd', 'e', 'f']
freqs = [5, 9, 12, 13, 16, 45]
codes = huffman_encoding(chars, freqs)
print("Huffman Codes:", codes)
```

3. Fractional Knapsack (Greedy Method) :

```
def fractional_knapsack(values, weights, capacity):
    ratio = [(values[i]/weights[i], i) for i in range(len(values))]
    ratio.sort(reverse=True)

    total_value = 0
    for r, i in ratio:
        if capacity >= weights[i]:
            total_value += values[i]
            capacity -= weights[i]
        else:
            total_value += r * capacity
            break
    return total_value
```

```
# Example
values = [60, 100, 120]
weights = [10, 20, 30]
capacity = 50
print("Maximum value:", fractional_knapsack(values, weights, capacity))
```

4. 0-1 Knapsack (Dynamic Programming) :

```
def knapsack_01(values, weights, capacity):
    n = len(values)
    dp = [[0 for _ in range(capacity + 1)] for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(1, capacity + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w])
            else:
                dp[i][w] = dp[i - 1][w]

    return dp[n][capacity]

# Example
values = [60, 100, 120]
weights = [10, 20, 30]
capacity = 50
print("Maximum value:", knapsack_01(values, weights, capacity))
```

5. n-Queens Problem (Backtracking) :

```
def print_board(board):
    for row in board:
        print(" ".join("Q" if x else "." for x in row))
    print()


def is_safe(board, row, col, n):
    for i in range(col):
        if board[row][i]:
            return False
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j]:
            return False
    for i, j in zip(range(row, n, 1), range(col, -1, -1)):
        if board[i][j]:
            return False
    return True


def solve_nqueens(board, col, n):
    if col >= n:
        print_board(board)
        return True
    res = False
    for i in range(n):
        if is_safe(board, i, col, n):
            board[i][col] = 1
            res = solve_nqueens(board, col + 1, n) or res
            board[i][col] = 0
    return res
```

```
n = 4

board = [[0]*n for _ in range(n)]

solve_nqueens(board, 0, n)
```

6. Quick Sort (Deterministic & Randomized) :

```
import random


def partition(arr, low, high):

    pivot = arr[high]

    i = low - 1

    for j in range(low, high):

        if arr[j] <= pivot:

            i += 1

            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[high] = arr[high], arr[i + 1]

    return i + 1


def quicksort(arr, low, high):

    if low < high:

        pi = partition(arr, low, high)

        quicksort(arr, low, pi - 1)

        quicksort(arr, pi + 1, high)


# Randomized version

def randomized_partition(arr, low, high):

    rand_pivot = random.randint(low, high)

    arr[high], arr[rand_pivot] = arr[rand_pivot], arr[high]

    return partition(arr, low, high)


def randomized_quicksort(arr, low, high):
    if low < high:
        pi = randomized_partition(arr, low, high)
        randomized_quicksort(arr, low, pi - 1)
        randomized_quicksort(arr, pi + 1, high)

# Example
arr1 = [10, 7, 8, 9, 1, 5]
arr2 = arr1.copy()
quicksort(arr1, 0, len(arr1)-1)
randomized_quicksort(arr2, 0, len(arr2)-1)
print("Deterministic QuickSort:", arr1)
print("Randomized QuickSort:", arr2)
```