**LAB 07: Dynamic Programming (Part 1)**

**IV. Exercises**

Import the library:

import time

import pylab

Measure time

def measure\_time\_1(func, N):

    runtime = []

    for n in N:

        start = time.time()

        f = func(n)

        stop = time.time()

        runtime.append(stop-start)

    return runtime

def measure\_time\_2(func, N1, N2):

    runtime = []

    for n in range (len(N1)):

        start = time.time()

        f = func(N1[n], N2[n])

        stop = time.time()

        runtime.append(stop-start)

    return runtime

def measure\_time\_3(func, N1, N2, N3):

    runtime = []

    for n in range (len(N1)):

        start = time.time()

        f = func(N1[n], N2[n], N3[n])

        stop = time.time()

        runtime.append(stop-start)

    return runtime

**Warn-up**

1. Computing a binomial coefficient

def binomial(n, k):

    C = [[0] \* (k + 1) for \_ in range(n + 1)]

    for i in range(n + 1):

        for j in range(min(i, k) + 1):

            if j == 0 or j == i:

                C[i][j] = 1

            else:

                C[i][j] = C[i - 1][j - 1] + C[i - 1][j]

    return C[n][k]

if \_\_name\_\_ == "\_\_main\_\_":

    print(binomial(5, 2))

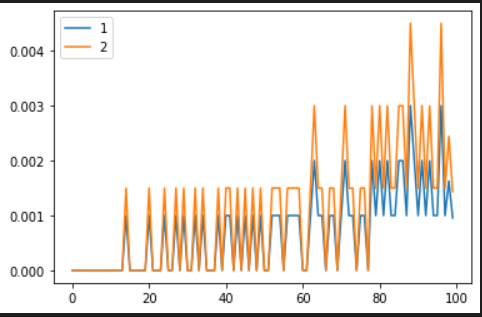
# Basic OP: addition in line 8

# Worst case: O(n^2)

# T(n)  = T(n-1) + O(n) = O(n^2)

# Time complexity: O(n^2)

Compare execution time



2. Coin-row problem

def CoinRow(coins):

    n = len(coins)

    if n == 0:

        return 0

    elif n == 1:

        return coins[0]

    elif n == 2:

        return max(coins[0], coins[1])

    else:

        A = [0] \* n

        A[0] = coins[0]

        A[1] = max(coins[0], coins[1])

        for i in range(2, n):

            A[i] = max(A[i-1], A[i-2] + coins[i])

        return A[n-1]

if \_\_name\_\_ == "\_\_main\_\_":

    coins = [5, 22, 26, 10, 4, 8]

    print(CoinRow(coins))

# Basic OP: addition in line 14

# Worst case: O(n)

# T(n)  = T(n-1) + T(n-2) = O(n)

# Time complexity: O(n)

3. Change-making problem

def ChangeMaking(C, n):

    F = [0] \* (n + 1)

    for i in range(1, n + 1):

        F[i] = min([F[i - c] for c in C if i - c >= 0]) + 1

    return F[n]

if \_\_name\_\_ == "\_\_main\_\_":

    C = [1, 3, 4]

    n = 6

    print(ChangeMaking(C, n))

# Basic OP: comparision in line 4

# Worst case: O(n^2)

# T(n)  = T(n-1) + O(n) = O(n^2)

# Time complexity: O(n^2)

**Intermediate exercise**

4. Knapsack Problem

def knapsack(W, w, v):

    F = [[0] \* (W + 1) for \_ in range(len(w) + 1)]

    for i in range(1, len(w) + 1):

        for j in range(1, W + 1):

            if j - w[i - 1] >= 0:

                F[i][j] = max(F[i - 1][j], F[i - 1][j - w[i - 1]] + v[i - 1])

            else:

                F[i][j] = F[i - 1][j]

    return F[-1][-1]

if \_\_name\_\_ == "\_\_main\_\_":

    W = 10

    w = [2, 3, 4, 5]

    v = [3, 4, 5, 6]

    print(knapsack(W, w, v))

# Basic OP: comparision in line 6

# Worst case: O(nW)

# T(n)  = T(n-1) + O(W) = O(nW)

# Time complexity: O(nW)