Learning Objectives

By the end of this worksheet, you will:

- Analyse the running time of functions containing nested loops.

---

1. **Nested loop variations**. Each of the following functions takes as input a non-negative integer `n` and contains at least one nested loop. For each loop, determine the *exact* number of iterations that will occur (in terms of `n`), and then use this to determine a Theta expression for the running time of each function. For nested loops, do the following:

   (i) First, determine an expression for the exact cost (# of steps) of the inner loop for a fixed iteration of the outer loop. This may or may be the same for each iteration of the outer loop.

   (ii) Determine the total cost of the outer loop by adding up the costs of the inner loop from (i). If the cost of the inner loop is the same for each outer loop iteration, you can simply multiply this cost by the total number of iterations of the outer loop. Otherwise, you'll need to set up and evaluate a summation (Σ).

   (iii) Repeat steps (i) and (ii) if there is more than one level of nesting, starting from the innermost loop and working your way outwards. Your final result should depend *only* on the function input, not any loop variables.

   (a)

```
def f1(n: int) -> None:
    i = 0
    while i < n:
        j = 0
        while j < n:
            j = j + 1
        i = i + 5
```

   (b)

```
def f2(n: int) -> None:
    i = 4
    while i < n:
        j = 1
        while j < n:
            j = j * 3
        k = 0
        while k < n:
            k = k + 2
        i = i + 1
```

(c)

```
def f3(n: int) -> None:
    i = 0
    while i < n:
        j = n
        while j > 0:
            k = 0
            while k < j:
                k = k + 1
            j = j - 1
        i = i + 4
```

(d)

```
def f4(n: int) -> None:
    i = 1
    while i < n:
        j = 0
        while j < i:
            j = j + 1

        i = i * 2
```

Note: you can look up a formula for "sum of powers of 2" or "geometric series" for the analysis in this question. This analysis is trickier than the others.

2. Consider the following algorithm:

```
def max_subsequence_sum(lst: List[int]) -> int:
    n = len(lst)
    max_so_far = 0
    for i in range(n):                  # Loop 1: i goes from 0 to n-1
        for j in range(i, n):           # Loop 2: j goes from i to n-1
            s = 0
            for k in range(i, j + 1):   # Loop 3: k goes from i to j
                s = s + lst[k]
            if max_so_far < s:
                max_so_far = s
    return max_so_far
```

Determine the Theta bound on the running time of this function in terms of n, the length of the input list. For practice, do not make any approximations on the *number of iterations* of any of the three loops; that is, your analysis should actually calculate the total number of iterations of the innermost k-loop across all iterations of the outer loop. Go slow! Treat this is as a valuable exercise in performing calculations with summation notation.

You may find the following formulas helpful (valid for all $n, a, b \in \mathbb{Z}^+$):

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=a}^{b} f(i) = \sum_{i'=0}^{b-a} f(i'+a)$$