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 and performs at least one loop. For each loop, determine the *exact* number of iterations that will occur (in terms of the size of the function's input), and then use this to determine a Theta expression for the running time of each function. When there are nested loops, remember to do the following:

   (i) First, determine an expression for the exact cost 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) The determine the total cost of the outer loop by adding up the costs of the inner loop from (i). Note that the if the cost of the inner loop is the same for each iteration, you can simply multiply this cost by the total number of iterations of the outer loop. Otherwise, you'll need set up and simplify an expression involving summation notation (Σ).

   (iii) Repeat steps (i)-(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 input size, not any loop counters.

   (a)
   ```
   def f1(n):
       i = 0
       while i < n:
           j = 0
           while j < n:
               j = j + 1
           i = i + 5
   ```

   Solution

   For a fixed iteration of the outer loop, the inner loop takes exactly n iterations, with each iteration costing a single step. Note that this is the same for each iteration of the outer loop.
   There are $\left\lceil \frac{n}{5} \right\rceil$ iterations of the outer loop. We multiply with this by the cost of each outer loop iteration to get a total cost of $\left\lceil \frac{n}{5} \right\rceil \cdot n$, which is $\Theta(n^2)$.

   (b)
   ```
   def f2(n):
       i = 4
       while i < n:
           j = 1
           while j < n:
               j = j * 3
           k = 0
           while k < n:
               k = k + 2
           i = i + 1
   ```

   Solution

   The first inner loop takes $\lceil \log_3 n \rceil$ iterations, with each iteration costing a single step. The second inner loop takes $\left\lceil \frac{n}{2} \right\rceil$ iterations, with each iteration also costing a single step. Again, this is the same for each iteration of the outer loop.
   There are $\max(n-4, 0)$ iterations of the outer loop, for a total cost of $\max(n-4, 0) \cdot \left(\lceil \log_3 n \rceil + \left\lceil \frac{n}{2} \right\rceil \right)$, which is $\Theta(n^2)$.

Note: if you want, you can simplify the cost of a *single* iteration of the outer loop to be simply $\Theta(n)$, since the calculated cost is $\lceil \log_3 n \rceil + \left\lceil \frac{n}{2} \right\rceil \in \Theta(n)$.

(c)
```
def f3(n):
    i = 0
    while i < n:
        j = n
        while j > 0:
            k = 0
            while k < j:
                k = k + 1
            j = j - 1
        i = i + 4
```

Solution

We start with the innermost loop (with k) for a fixed iteration of the other loops. This executes exactly j times, with each iteration costing a single step. Note that this cost depends on the value of j.

Next, we consider the cost of the middle loop for a fixed iteration of the outer loop. We need to add up the cost of the innermost loop for each value of j from n down to 1. (Note that the summation expression will be the same as if j went up from 1 to n.) This total cost is

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

Note that this cost does *not* depend on the iteration number of the outermost loop.

Finally, the outer loop runs $\left\lceil \frac{n}{4} \right\rceil$ times in total. This means that the total cost is $\left\lceil \frac{n}{4} \right\rceil \cdot \frac{n(n+1)}{2}$, which is $\Theta(n^3)$.

(d) 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.

```
def f4(n):
    i = 1
    while i < n:
        j = 0
        while j < i:
            j = j + 1

        i = i * 2
```

Solution

The inner loop takes exactly i iterations for a fixed iteration of the outer loop, and the cost per iteration is 1 step.

The outer loop runs $\lceil \log n \rceil$ iterations,* where i takes on the values $2^0, 2^1, 2^2, \ldots, 2^{\lceil \log n \rceil - 1}$ before the loop terminates when $i = 2^{\lceil \log n \rceil - 1}$. We have the following summation for the total running time of this function:

$$\sum_{k=0}^{\lceil \log n \rceil - 1} 2^k = 2^{\lceil \log n \rceil} - 1$$

So the total cost is $2^{\lceil \log n \rceil} - 1$. It turns out that $2^{\lceil \log n \rceil} - 1 \in \Theta(n)$, although we'll leave a proof of this as an exercise. Note that one important property of floor that's helpful here is that $\lceil \log n \rceil - 1 \leq \log n <$

$\lceil \log n \rceil + 1$.

*Correction: this used to read $\lfloor \log n \rfloor$ iterations.

2. Consider the following algorithm:

```
def subsequence_sum(L):
    """L is a list of numbers."""
    n = len(L)
    max = 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
            sum = 0
            for k in range(i, j + 1):   # Loop 3: k goes from i to j
                sum = sum + L[k]
            if max < sum:
                max = sum
    return max
```

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 $n, a, b \in \mathbb{Z}^+$):

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

Solution

As usual, we focus on the innermost loop (Loop 3). For a fixed iteration of Loops 1 and 2, the Loop 3 takes exactly $j - i + 1$ iterations, with a cost of 1 step per iteration. So the cost of the innermost loop for a fixed iteration of Loops 1 and 2 is $j - i + 1$. Note that this number of iterations depends on both i and j.

Now consider the cost of all iterations of Loop 2 for a fixed iteration of Loop 1. We need to sum the cost of Loop 3 for all values of j from i to $n - 1$. The total cost for a fixed iteration of the outer Loop 1 is:

$$\begin{aligned}
\sum_{j=i}^{n-1} j - i + 1 &= \sum_{j'=1}^{n-i} j' && \text{(change of variable } j' = j - i + 1\text{)} \\
&= \sum_{j'=1}^{n-i} j' \\
&= \frac{(n-i)(n-i+1)}{2} \\
&= \frac{n^2 - 2ni + n - i + i^2}{2} \\
&= \frac{1}{2}i^2 - \left(n + \frac{1}{2}\right) i + \left(\frac{n^2}{2} + \frac{n}{2}\right)
\end{aligned}$$

[Note: we're grouping by i here because the next step is to sum over possible values of i.]

To obtain the total cost, we have to add up the cost of each iteration of the outer i-loop (as i goes from 0 to

$n-1$):

$$\begin{aligned}
&\sum_{i=0}^{n-1}\left[\frac{1}{2}i^2-\left(n+\frac{1}{2}\right)i+\left(\frac{n^2}{2}+\frac{n}{2}\right)\right]\\
&=\frac{1}{2}\sum_{i=0}^{n-1}i^2-\left(n+\frac{1}{2}\right)\sum_{i=0}^{n-1}i+\left(\frac{n^2}{2}+\frac{n}{2}\right)\sum_{i=0}^{n-1}1\\
&=\frac{1}{2}\left(\frac{(n-1)n(2n-1)}{6}\right)-\left(n+\frac{1}{2}\right)\left(\frac{(n-1)n}{2}\right)+\left(\frac{n^2}{2}+\frac{n}{2}\right)n\\
&=\frac{(n-1)n(2n-1)}{12}-\left(\frac{(n-1)n(2n+1)}{4}\right)+\frac{n^3}{2}+\frac{n^2}{2}\\
&=-\frac{(n-1)n(n+3)}{3}+\frac{n^3}{2}+\frac{n^2}{2}\\
&=-\frac{n^3+2n^2-3n}{3}+\frac{n^3}{2}+\frac{n^2}{2}\\
&=\frac{n^3}{6}-\frac{n^2}{6}+n
\end{aligned}$$

So then the total number of basic operations is $\frac{n^3}{6}-\frac{n^2}{6}+n$, which is $\Theta(n^3)$.