Learning Objectives

By the end of this worksheet, you will:

- Analyse the worst-case running time of functions.
- Define input families for a given function that have specified asymptotic running times.

1. **Worst-case analysis.** Consider the following function, which takes in a list of numbers and determines whether the list contains any duplicates.

```python
def has_duplicate(lst: List[int]) -> bool:
    n = len(lst)
    for i in range(n):  # i goes from 0 to n-1
        for j in range(i + 1, n):  # j goes from i+1 to n-1
            if lst[i] == lst[j]:
                return True
    return False
```

(a) Find a good upper bound on the worst-case running time of this function.

**Solution**

For a fixed iteration of the outer loop, the inner loop runs at most \( n - 1 - i \) iterations (if it doesn't stop early), with each iteration taking constant time (1 step). The outer loop runs at most \( n \) iterations, for \( i \) going from 0 to \( n - 1 \). The total cost is

\[
\sum_{i=0}^{n-1} (n - 1 - i) = n(n - 1) - \sum_{i=0}^{n-1} i = \frac{n(n - 1)}{2}
\]

So the total number of steps of this algorithm is at most \( \frac{n(n - 1)}{2} \) (ignoring the +1 for the cost of the final return), which is \( \mathcal{O}(n^2) \).

(b) Prove a matching lower bound on the worst-case running time of this function, by finding an input family whose asymptotic runtime matches the bound you found in the previous part.

For an extra challenge, find an input family for which this function *does* return early (i.e., the return on line 6 executes), but the runtime is still Theta of the upper bound you found in the previous part.

**Solution**

There's actually several possibilities here! One is that the input list contains no duplicates, except that the last two elements (\( L[n - 2] \) and \( L[n - 1] \)) are equal. In this case, all iterations of the outer loop occur except the last one: when \( i = n - 2 \) and \( j = n - 1 \), the inner loop returns early. The running time in this case is

\[
\sum_{i=0}^{n-2} (n - 1 - i) = \sum_{i=0}^{n-2} (n - 1) - \sum_{i=0}^{n-2} i = (n - 1)^2 - \frac{(n - 2)(n - 1)}{2} = \frac{n(n - 1)}{2}
\]

This runtime is \( \Theta(n^2) \), matching the bound from part (a).

(c) Find an input family whose running time is \( \Theta(n) \), where \( n \) is the length of the input list, and analyse the running time of `has_duplicate` on this input family. [Note that \( \Theta(n) \) is neither the worst-case nor best-case running time!]


2. Substring matching. Here is an algorithm which is given two strings, and determines whether the first string is a substring of the second. (In Python, this would correspond to the in operation, e.g., ‘oof’ in ‘proofs are fun’). Assume that both strings are non-empty, and that the length of the second string is equal to the square of the length of the first string.

```python
def substring(s1: str, s2: str) -> bool:
    """Precondition: len(s2) = len(s1) * len(s1) -- for this analysis.""
    i = 0
    while i < len(s2) - len(s1):
        # Check whether s1 == s2[i..i+len(s1)-1]
        match = True
        for j in range(len(s1)):
            # If the current corresponding characters don’t match,
            # stop the inner loop.
            if s1[j] != s2[i + j]:
                match = False
                break

        # If a match has been found, stop and return True.
        if match:
            return True
        i = i + 1
    return False
```

(a) Let $n$ represent the length of $s1$ (and so the length of $s2$ is $n^2$). Find a good asymptotic upper bound on the worst-case running time of this function in terms of $n$.

**Solution**

Let’s analyze the running time of this function assuming there are no early returns. In this case, for a fixed iteration of the outer loop, the inner loop takes $n$ iterations, and hence this many steps (since each iteration takes constant time). The outer loop runs for $n^2 - n$ iterations, for a total cost of $n(n^2 - n)$, which is $O(n^3)$.

---

1 The algorithm certainly works even if the input string lengths don’t satisfy this requirement, we add it here to simplify some of the analysis.
(b) Find, with proof, an input family whose running time matches the upper bound you found in part (a).

**Hint:** you can pick \( s_1 \) to be a string of length \( n \) that just repeats the same character \( n \) times.

**Solution**

This input family is rather tricky to describe and analyse properly. Let \( n \in \mathbb{Z}^+ \), and let \( s_1 \) be the string of length \( n \) that only contains the character ‘a’, and let \( s_2 \) be the string of length \( n \) defined as:

\[
s_2[i] = \begin{cases} 
  b, & \text{if } n \mid i + 1 \\
  a, & \text{otherwise}
\end{cases}
\]

For example, when \( n = 4 \), we have

\[s_1 = \text{aaaa} \text{ and } s_2 = \text{aaabaabaaabaab}\]

Intuitively, since \( s_1 \) and \( s_2 \) are so similar, the inner loop has to run for many iterations until it finds a mismatch.

We leave the analysis of the running time of `substring` on this input family as an exercise, with one hint: the outer loop will run \( n^2 - n \) times in total; rather than trying to sum up over all of these iterations, break it up into \( n \) groups of \( n \) consecutive iterations. You should find that the running time of the first \( n \) iterations (from \( i = 0 \) to \( n - 1 \)) is more straightforward to analyse, and each subsequent group of \( n \) iterations has the same cost.