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.
(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.

(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 $n$.

---

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.