Learning Objectives

By the end of this worksheet, you will:

- Analyse the worst-case running time of an algorithm.
- Find, with proof, an input family for a given algorithm that has a specified asymptotic running time.

---

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

```
1 def has_duplicates(lst: list) -> bool:
2     n = len(lst)
3     for i in range(n):              # Loop 1: i goes from 0 to n-1
4         for j in range(i + 1, n):  # Loop 2: j goes from i+1 to n-1
5             if lst[i] == lst[j]:
6                 return True
7     return False
```

   (a) Find a tight asymptotic 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 (with proof) 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, with proof, an input family whose running time is $\Theta(n)$, where n is the length of the input list.

2. **Analysing binary search**. As you've discussed in CSC108 and/or CSC148, one of the most fundamental advantages of sorted data is that it makes it easier to search this data for a particular item. Rather than searching sequentially (item by item) through the entire list of data, we can employ the *binary search algorithm*:

```
1  def binary_search(lst: List[int], x: int) -> bool:
2      i = 0                    # i is the lower bound of the search range (inclusive)
3      j = len(lst)             # j is the upper bound of the search range (exclusive)
4      while i < j:
5          mid = (i + j) // 2   # mid is the midpoint of the search range
6          if lst[mid] == x:
7              return True
8          elif lst[mid] < x:
9              i = mid + 1      # New search range is (mid+1) to j
10         else:
11             j = mid          # New search range is i to mid
12
13     return False
```

The intuition behind analysing the running time of binary search is to say that at each loop iteration, the size of the range being searched decreases by a factor of 2. At the same time, our more formal techniques of analysis seem to have trouble. We don't have a predictable formula for the values of variables i and j after k iterations, since how the search range changes depends on the contents of *lst* and the item being searched for.

We can reconcile the intuition with our more formal approach by explicitly introducing and analysing the behaviour of a new variable. Specifically: let $r = j - i$ be a variable representing the size of the search range.

(a) Let n represent the length of the input list *lst*. What is the initial value of r, in terms of n?

(b) For what value(s) of r will the loop *terminate*?

(c) Prove that at each loop iteration, if the item `x` is not found then the value of r decreases by at least a factor of 2. More precisely, let r_k and r_{k+1} be the values of r immediately before and after the k-th iteration, respectively, and prove that $r_{k+1} \leq \frac{1}{2} r_k$. You can use external properties of floor/ceiling in this question.

(d) Find the exact maximum number of iterations that could occur (in terms of n), and use this to show that the worst-case running time of `binary_search` is $\mathcal{O}(\log n)$.

(e) Prove that the worst-case running time of `binary_search` is $\Omega(\log n)$. Note that your description of the input family should talk about both the input list, lst, and the item being searched for, x.