Learning Objectives

By the end of this worksheet, you will:

- Analyse the worst-case running time of binary search.

1. **Analysing binary search.** As you’ve discussed in CSC108 and 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:

```python
def binary_search(L, x):
    """Return whether x is an element of L.
    Precondition: L is a sorted list of numbers.
    ""
    i = 0       # i is the lower bound of the search range (inclusive)
    j = len(L)  # j is the upper bound of the search range (exclusive)
    while i < j:
        mid = (i + j) // 2  # mid is the midpoint of the search range
        if L[mid] == x:
            return True
        elif L[mid] < x:
            i = mid + 1  # New search range is (mid+1) to j
        else:
            j = mid - 1  # New search range is i to (mid-1)
    # If this point is reached, then x is not in L.
    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 *L* 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 *L*. What is the initial value of \( r \), in terms of \( n \)?

(b) For what values of \( r \) will the loop terminate?
(c) Prove that at each loop iteration, if the item 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 \texttt{binary_search} is \( O(\log n) \).

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

You may assume that if the loop does not return early, then it runs for \( \Omega(\log n) \) iterations.\(^1\)

\(^1\)Something to think about: why did we explicitly mention this assumption? Why does it not follow from your work in part (c)?

\emph{Challenge}: prove the \( \Omega(\log n) \) bound.
(f) Finally, prove that the best-case running time of binary search is $O(1)$ (independent of the size of the input list). Note that proving an upper bound on the best-case is analogous to proving a lower bound on the worst case: both require you to describe a family of inputs whose running time fulfills some property.