Learning Objectives

By the end of this worksheet, you will:

- Analyse the running time of loops whose loop counter changes differently in different iterations.
- Analyse the running time of functions that call helper functions.
- Understand and express properties about the minimum and maximum of a set of numbers.

---

1. Varying loop increments. In lecture, we saw one (complicated) example of a loop where the change in loop counter value was not the same for each iteration. In this question, you'll get some practice analyzing such loops yourself using a general technique. For each of the following functions, do the following:

   (i) Identify the *minimum* and *maximum* possible change for the loop counter in a single iteration.

   (ii) Use this to determine formula for an exact *lower bound* and *upper bound* on the value of the loop counter after k iterations.

   (iii) Use these formulas and the loop condition to find an *upper bound* and *lower bound* on the exact number of loop iterations that will occur.

   Note: a lower bound on the value of the loop counter gives you an upper bound on the number of loop iterations in the loop, and vice versa.

   (iv) Use your upper and lower bounds on the number of iterations to find Big-Oh and Omega asymptotic bounds on the running time of the function. Note that if you have the same expression for Big-Oh and Omega, then you can also conclude a Theta bound.

(a)

```
1  def varying1(n):
2      i = 0
3      while i < n:
4          if i % 3 == 0:
5              i = i + 1
6          elif i % 2 == 1:
7              i = i + 3
8          else:
9              i = i + 6
```

Solution

The minimum change in the loop is that i increases by 1; the maximum change is that i increases by 6. Let i_k be the value of i after k iterations. The previous observation tells us that $k \leq i_k \leq 6k$.

Since the loop terminates when $i \geq n$, we want to find the smallest value of k such that $i_k \geq n$. On the one hand, since $i_k \geq k$, we know that $i_k \geq n$ when $k = n$; that is, the loop can run *at most* n iterations. Since each iteration takes 1 step, this leads to an *upper bound* on the running time of $\mathcal{O}(n)$.

On the other hand, since $i_k \leq 6k$, we know that if $i \geq n$, then $6k \geq n$. This means that $k \geq \left\lceil \frac{n}{6} \right\rceil$, and so the loop must run for *at least* $\left\lceil \frac{n}{6} \right\rceil$ iterations, leading to a *lower bound* on the running time of $\Omega(n)$.

Since the upper bound and lower bounds are the same (asymptotically), we can conclude that the overall running time of this function is $\Theta(n)$.

(b)
```
def varying2(n):
    i = 1
    while i < n:
        if n % i <= i/2:
            i = 2 * i
        else:
            i = 3 * i
```

Solution

The argument is the same as the previous one, except now i increases by at least a multiplicative factor of 2, and at most a factor of 3. This means that $2^k \leq i_k \leq 3^k$, and so the number of iterations is at most $\log_2 n$ and at least $\log_3 n$ [using the same reasoning as in part (a)].
That is, the upper bound on the running time here is $\mathcal{O}(\log_2 n)$, and the lower bound is $\Omega(\log_3 n)$. Since we know that $\log_3 n \in \Theta(\log_2 n)$, we can conclude that the tight bound on the running time is $\Theta(\log n)$.

2. **Helper functions.** So far, we have analysed loops as the main mechanism for writing functions whose running time depends on the size of the function's input. Another source of non-contant running times that you often encounter are other functions that are used as helpers in an algorithm.

   For this exercise, consider having two functions `helper1` and `helper2`, which each take in a positive integer as input. Moreover, assume that `helper1`'s running time is $\Theta(n)$ and `helper2` is $\Theta(n^2)$, where n is the value of the input to these two functions.

   Your goal is to analyse the running time of each of the following functions, which make use of one or both of these helper functions. When you count costs for these function calls, simply substitute the value of the argument of the call into the function $f(x) = x$ or $f(x) = x^2$ (depending on the helper). For example, count the cost of calling `helper1(k)` as k steps, and `helper2(2*n)` as $4n^2$ steps.

(a)
```
def f1(n):
    helper1(n)
    helper2(n)
```

Solution

The call to `helper1` takes n steps, and the call to `helper2` takes n^2 steps, for a total of $n^2 + n$ steps. This is $\Theta(n^2)$.

(b)
```
def f2(n):
    i = 0
    while i < n:
        helper1(n)
        i = i + 2

    j = 0
    while j < 10:
        helper2(n)
        j = j + 1
```

Solution

The first loop takes $\left\lceil \frac{n}{2} \right\rceil$ iterations, and each iteration requires n steps for the call to `helper1`. As with nested loops, we ignore the lower-order cost of the loop counter increment `i = i + 2`. So the total cost of this loop is $\left\lceil \frac{n}{2} \right\rceil \cdot n$.

The second loop runs for 10 iterations, and each iteration requires n^2 steps for the call to `helper2`. So we count the cost for this loop as $10n^2$.

The total cost is $\left\lceil \frac{n}{2} \right\rceil \cdot n + 10n^2$, which is $\Theta(n^2)$.

(c)

```
def f2(n):
    i = 0
    while i < n:
        helper1(i)
        i = i + 1

    j = 0
    while j < 10:
        helper2(j)
        j = j + 1
```

Solution

The first loop takes n iterations, but now the cost of the call to `helper1` changes at each iteration. For a fixed iteration of this loop, the cost of calling `helper1(i)` is i, and so the total cost over all iterations of this loop is $\sum_{i=0}^{n-1} i = \frac{(n-1)n}{2}$ (note that this is the same as when we analysed one of the nested loop examples from lecture).

Similarly, the cost of the second loop is $\sum_{j=0}^{9} j^2$; this is one, however, is a *constant* cost with respect to n.

So the first loop has a running time of $\Theta(n^2)$, and the second has a running time of $\Theta(1)$. The overall running time is the sum of these two, which is $\Theta(n^2)$.

3. **Set maximum and minimum**. Students learning about worst-case and best-case running times are often confused about exactly how (and why) to prove upper and lower bounds on these functions. To help your understanding, in this exercise we're going to take a steps back and look at a simpler problem: determining properties of the maximum and minimum of a single set of numbers. For this problem, we'll use the notation $\max(S)$ to denote the maximum value in set of numbers S, and $\min(S)$ to denote its minimum.

   (a) Let S be a non-empty, finite subset of $\mathbb{R}$, and assume that every element of S is less than or equal to 165. What can you conclude about $\max(S)$ and/or $\min(S)$?

   Solution

   We know that $\max(S) \leq 165$ and $\min(S) \leq 165$.

   (b) Express the idea from part (a) by completing the following formula in predicate logic:

   $$\forall S \subseteq \mathbb{R},\ \forall M \in \mathbb{R},\ \big(\forall x \in S,\ x \leq M\big) \Rightarrow$$

   Solution

   $$\forall S \subseteq \mathbb{R},\ \forall M \in \mathbb{R},\ \big(\forall x \in S,\ x \leq M\big) \Rightarrow \max(S) \leq M \wedge \min(S) \leq M$$

   (c) Now suppose instead that *at least one* element of S is less than or equal to 165. What can you conclude about $\max(S)$ and/or $\min(S)$?

   Solution

   We know that $\min(S) \leq 165$. However, we can't conclude anything about $\max(S)$.

   (d) Express the idea from part (a) by completing the following formula in predicate logic:

   $$\forall S \subseteq \mathbb{R},\ \forall M \in \mathbb{R},\ \big(\exists x \in S,\ x \leq M\big) \Rightarrow$$

   Solution

   $$\forall S \subseteq \mathbb{R},\ \forall M \in \mathbb{R},\ \big(\exists x \in S,\ x \leq M\big) \Rightarrow \min(S) \leq M$$

   (e) Finally, write the analogous statements to the previous parts that use the inequality $x \geq M$ instead of $x \leq M$. Your four statements combined should describe upper and lower bounds on both $\min(S)$ and $\max(S)$.

   Solution

   $$\forall S \subseteq \mathbb{R},\ \forall M \in \mathbb{R},\ \big(\forall x \in S,\ x \geq M\big) \Rightarrow \max(S) \geq M \wedge \min(S) \geq M$$

   $$\forall S \subseteq \mathbb{R},\ \forall M \in \mathbb{R},\ \big(\exists x \in S,\ x \geq M\big) \Rightarrow \max(S) \geq M$$