Learning Objectives

By the end of this worksheet, you will:

- Determine the exact number of iterations of loops with a variety of loop counter behaviours.
- Find the asymptotic running time of programs containing loops.

---

1. **Loop variations**. Each of the following functions takes as input a non-negative integer and performs at least one loop. For each loop, determine the *exact* number of iterations that will occur (in terms of the size of the function's input), and then use this to determine the simplest Theta expression[1] for the running time of each function. You do *not* need to prove any "$g \in \Theta(f)$" statements here.

   Note: each loop body runs in $\Theta(1)$ time in this question. While this won't always be the case, such examples allow you to focus on just counting loop iterations here.

   (a)

```python
def f1(n):
    i = 0
    while i < n:
        print(i)
        i = i + 5
```

   Solution

   There are $\left\lceil \frac{n}{5} \right\rceil$ loop iterations. Since each iteration takes constant time, the total runtime of this function is $\Theta(n)$.

   (b)

```python
def f2(n):
    i = 4
    while i < n:
        print(i)
        i = i + 1
```

   Solution

   There are $\max(n-4, 0)$ loop iterations. Since each iteration takes constant time, the total runtime of this function is also $\Theta(n)$.

   (c)

```python
def f3(n):
    # Assume n > 0 here.
    i = 0
    while i < n:
        print(i)
        i = i + (n / 10)
```

   Solution

   There are exactly 10 loop iterations. Since each iteration takes constant time, the total runtime of this function is $\Theta(1)$.

[1] By "simplest," we mean ignoring constants and slower-growth terms. For example, write $\Theta(n)$ instead of $\Theta(2n + 0.3)$.

(d)
```python
def f4(n):
    i = 20
    while i < n*n:
        print(i)
        i = i + 3
```

Solution

There are $\max\left(\left\lceil \frac{n^2 - 20}{3} \right\rceil, 0\right)$ loop iterations. Since each iteration takes constant time, the total runtime of this function is $\Theta(n^2)$.

(e)
```python
def f5(n):
    i = 20
    while i < n*n:
        print(i)
        i = i + 3

    j = 0
    while j < n:
        print(j)
        j = j + 0.01
```

Solution

The first loop takes $\Theta(n^2)$ time (this is a previous part). The second loop takes $\Theta(n)$ time. Since $n \in \mathcal{O}(n^2)$, the total runtime of this function is $\Theta(n^2)$.

2. **Multiplicative increments**. Consider the following function, which takes in a positive integer

```
def f(n):
    i = 1
    while i < n:
        print(i)
        i = i * 2
```

though this looks similar to previous examples, the fact that the loop variable i changes by a multiplicative rather than additive factor requires a more principled approach in determining the number of loop iterations.

(a) Let i_0 be the value of i when 0 loop iterations have occurred, i_1 be the value of i right after 1 loop iteration has occurred, and in general i_k to be the value of i right after k loop iterations have occurred. For example, $i_0 = 1$ (the initial value of i) and $i_1 = 2$.

Determine the values of i_2, i_3, i_4, and a general formula for i_k.[2]

Solution

The general formula is $i_k = 2^k$.

(b) Determine the **exact** number of loop iterations that occur in terms of n. Use your work from part (a); note that you have a formula for i in terms of the number of iterations.

Solution

The loop terminates when $i \geq n$. We want to find the smallest value of k such that $i_k \geq n$, i.e., $2^k \geq n$. Since k must be an integer, the smallest value it can be is $\lceil \log n \rceil$. So then the loop runs $\lceil \log n \rceil$ times.

(c) Determine the Theta running time for the function f.

Solution

Since each loop iteration takes $\Theta(1)$ time, the total running time is $\Theta(\log n)$.

(d) Why did we not initialize i = 0 in this function?

[2]Of course, if n is small then not a lot of loop iterations occur. You can think of i_k as representing the value of i after k loop iterations, *if* k iterations occur.

3. **A more unusual increment**. Consider the following function, which takes a positive integer

```python
def f(n):
    i = 2
    while i < n:
        print(i)
        i = i * i
```

Analyse the running time of this function using the same technique as the previous question. You may assume that $n \geq 2$ here.

Solution

The hardest part is finding a general formula for i_k, the value of variable `i` after k iterations. This turns out to be $i_k = 2^{2^k}$ (the best way to find this is by computing the first few values of `i` by hand). We leave the rest of the analysis as an exercise.