CSC110 Lecture 23: More Running-Time Analysis

Print this handout

Exercise 1: Analysing running time of while loops

Your task here is to **analyse the running time of each of the following functions**. Recall the technique from lecture for calculating the number of iterations a while loop takes:

1. Find i_0, the value of the loop variable at the start of the first iteration. (You may need to change the notation depending on the loop variable name.)
2. Find a pattern for i_0, i_1, i_2, etc. based on the loop body, and a formula for a general i_k, the value of the loop variable after k loop iterations, assuming that many iterations occurs.
3. Find the *smallest* value of k that makes the loop condition False. This gives you the number of loop iterations.

   You'll need to use the floor/ceiling functions to get the correct exact number of iterations.

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

1.

```python
def f1(n: int) -> None:
    """Precondition: n >= 3."""
    i = 3
    while i < n:
        print(i)
        i = i + 5
```

Iteration	Loop variable `i`
0	$i_0 =$
1	$i_1 =$
2	$i_2 =$
3	$i_3 =$
...	...
k	$i_k =$

Write your running-time analysis below.

2.

```python
def f2(n: int) -> None:
    """Preconditions: n > 0 and n % 10 == 0."""
    i = 0
    while i < n:
        print(i)
        i = i + (n // 10)
```

Iteration	Loop variable `i`
0	$i_0 =$
1	$i_1 =$
2	$i_2 =$
3	$i_3 =$
...	...
k	$i_k =$

Write your running-time analysis below.

3.

```python
def f3(n: int) -> None:
    """Precondition: n >= 5."""
    i = 20
    while i < n * n:
        print(i)
        i = i + 3
```

Iteration	Loop variable `i`
0	$i_0 =$
1	$i_1 =$
2	$i_2 =$
3	$i_3 =$
...	...
k	$i_k =$

Write your running-time analysis below.

Exercise 2: Analysing nested loops

Remember, to analyse the running time of a nested loop:

1. First, determine an expression for the exact running time of the inner loop(s) for a fixed iteration of the outer loop. This may or may be the same for each iteration of the outer loop.
2. Then determine the total running time of the outer loop by adding up the steps of the inner loop(s) from Step 1. Note that if the number of steps of the inner loop(s) is the same for each iteration, you can simply multiply this number by the total number of iterations of the outer loop. Otherwise, you'll need to set up and simplify an expression involving summation notation (Σ).
3. Repeat Steps (1) and (2) if there is more than one level of nesting, starting from the innermost loop and working your way outwards. Your final result should depend *only* on the function input, not any loop variables.

You will also find the following formula helpful:

$$\forall n \in \mathbb{N},\ \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

Each of the following functions takes as input a non-negative integer and performs at least one loop. Analyse the running time of each function.

4.

```python
def f4(n: int) -> None:
    """Precondition: n >= 0"""
    i = 0
    while i < n:                 # Loop 1
        for j in range(0, n):    # Loop 2
            print(j)
        i = i + 5
```

5.

```python
def f6(n: int) -> None:
    """Precondition: n >= 4"""
    i = 4
    while i < n:                 # Loop 1
        j = 1
        while j < n:             # Loop 2
            j = j * 3
        for k in range(0, i):    # Loop 3
            print(k)
        i = i + 1
```

6.

```python
def f6(n: int) -> None:
    """Precondition: n >= 0"""
    i = 0
    while i < n:                     # Loop 1
        j = n
        while j > 0:                 # Loop 2
            for k in range(0, j):    # Loop 3
                print(k)
            j = j - 1
        i = i + 4
```

Additional exercises

1. Analyse the running time of the following function.

```python
def extra(n: int) -> None:
    """Precondition: n >= 1"""
    i = 1
    while i < n:         # Loop 1
        j = 0
        while j < i:     # Loop 2
            j = j + 1

        i = i * 2
```

Hint: the actual calculation for this function is a little trickier than the others. You may need a formula from Appendix C.1 Summations and Products. Note: you can look up a formula for "sum of powers of 2" or "geometric series" for the analysis in this question. This analysis is trickier than the others.

2. Consider the following algorithm:

```python
def subsequence_sum(lst: list[int]) -> int:
    n = len(lst)
    max_so_far = 0
    for i in range(0, n):                # Loop 1: i goes from 0 to n-1
        for j in range(i, n):            # Loop 2: j goes from i to n-1
            s = 0
            for k in range(i, j + 1):    # Loop 3: k goes from i to j
                s = s + lst[k]
            if max_so_far < s:
                max_so_far = s
    return max_so_far
```

Analyse the running time of this function (in terms of n, the length of the input list). For practice, do not make any approximations on the *number of iterations* of any of the three loops; that is, your analysis should actually calculate the total number of iterations of the innermost k-loop across all iterations of the outer loop. Go slow! Treat this is as a valuable exercise in performing calculations with summation notation.

You may find the following formulas helpful (valid for all $n, a, b \in \mathbb{Z}^+$):

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=a}^{b} f(i) = \sum_{i'=0}^{b-a} f(i'+a)$$