March 9-13

Topics this week:

Hour 1 - Worst-case runtime analysis (will be on midterm)
Hour 2 - Worksheets: worst-case - has_duplicates / analyzing binary search
Hour 3 - Finish Hour 1 analysis / More runtime: twisty example from notes
Hour 4 - Worksheets: varying loop increments, helper functions, and more twisty analysis details,

CSC165 Student Questions / Announcements

- **Problem Set 4 should be out later this week**
- **Information about Midterm 2 is posted on the course website + old midterms for studying**
  - You are expected to be able to use the theorems from the course notes / lecture to jump from an exact step expression to a Theta/Big-Oh/Omega expression when doing running time analysis, but you don’t need to cite the specific theorems (e.g. "By Theorem 5.1…")
  - In proofs, the same principle applies as Midterm 1 (and problem sets): the only external facts you can use are the ones given in the question.

Go to **menti.com** and enter the code 64 20 49 (For Q+A during lecture)
Tight bounds:

Suppose we have proven that $\text{RT}_{\text{fun}} \in O(f(n))$.

This bound is tight if $\text{RT}_{\text{fun}} \in \Omega(f(n))$.

[\text{ie } \text{RT}_{\text{fun}} \in \Theta(f(n))]$

2 Types of Questions:

1. "Find a tight upper bound", prove that runtime is $O(f(n))$, such that you could also prove $\Omega(f(n))$.

2. "Prove the bound is tight" show $\Theta(f(n))$.

Worst-case runtime:

Now we will consider cases where the runtime depends on the input size and the actual input value.
Review: Definitions from pages 104-105:

\( f(n) \) is an upper bound on the worst-case runtime: for \( f(n) \):

\[
\forall n \in \mathbb{N}, \forall x \in \mathcal{I}_{\text{fun}_n}, \text{RT}_{\text{fun}}(x) \leq f(n).
\]

\( g(n) \) is a lower bound on the worst-case runtime: for \( g(n) \):

\[
\forall n \in \mathbb{N}, \exists x \in \mathcal{I}_{\text{fun}_n}, \text{RT}_{\text{fun}}(x) \geq g(n).
\]

Example:

```python
def is_palindrome(s: str) -> bool:
    # Return whether s is a palindrome.
    n = len(s)
    for i in range(n):
        if not s[i] == s[n - 1 - i]:
            return False
    return True
```

 palindrome:
 e.g. 'abba' ✓
 'abcab' ✗
Analysis of the worst-case runtime of is_palindrome:

Part 1: Upper bound: \( \mathcal{O}(n) \).

Let \( n \in \mathbb{N} \). Let \( s \) be an arbitrary string of length \( n \). We'll find an upper bound on \( RT_{is\_pal}(s) \).

Ignoring the early return in the loop, we know that there will be at most \( n \) iterations of the loop. And each loop iteration takes constant time (1 step), so the runtime for \( s \) is \( O(n) \).

The worst-case runtime is \( O(n) \).

Part 2: Lower bound:

Let \( n \in \mathbb{N} \). Let \( s = a \ldots a \), \( \underbrace{}_{n} \)

For this specific \( s \), the "if" condition is always false, so the loop runs \( n \) times.

So the runtime for this \( s \) is \( \Omega(n) \).

So worst-case runtime is \( \Omega(n) \).
Example (page 108 course notes)

```python
def palindrome_prefix(s: str) -> int:
    n = len(s)
    for prefix_length in range(n, 0, -1):  # goes from n down to 1
        # Check whether s[0:prefix_length] is a palindrome
        is_palindrome = True
        for i in range(prefix_length):
            if s[i] != s[prefix_length - 1 - i]:
                is_palindrome = False
                break

        # If a palindrome prefix is found, return the current length.
        if is_palindrome:
            return prefix_length
```

$O(n^2)$ \quad \exists x \in \mathbb{Z}^+ \quad \subseteq O(n^2)$
Next class:

Analysis of Worst-case runtime for palindrome_prefix:

Part 1 Upper bound:

Let $n \in \mathbb{N}$. Let $s$ be an arbitrary string of length $n$. break/return.

Ignoring early "loop stops" (and early returns).

We know Loop 2 iterates at most $\text{prefix length}$ times. And Loop 1 iterates at most $n$ times, for $\text{prefix length} \leq n, n-1, \ldots, 1$.

\[
\sum_{i=1}^{\text{prefix length}} \leq \frac{n(n+1)}{2} \sim O(n^2)
\]

Part 2 Lower bound:

Let $n \in \mathbb{N}$. Let $s$ be a string of length $n$, where every character is an 'a', except $s[\lfloor \frac{n}{2} + 1 \rfloor] = \beta$.

For this $s$, Loop 1 will iterate until $\text{prefix length} = \lfloor n/2 \rfloor$. 

\[
s = \text{adabada}
\]
For a given prefix length, Loop 2 will iteratively until the '0' is seen. (if condition met to break).

The body of Loop 2 takes constant time, so there are nested loops. Take: \( \frac{n^2}{2} \)

\[
\sum_{i=0}^{\frac{n^2}{2}} (\frac{n^2}{2} - i) \sim \Omega \left( n^2 \right)
\]

WC runtime of findlen is \( \Omega \left( n^2 \right) \).

CSC165 Student Questions / Announcements

- Problem Set 4 should be out now.
- Midterm 2 next week (see course website Exam tab for more information)

Go to menti.com and enter the code 113313 (For Q+A during lecture)

64 20 49
CSC 165 Lecture 17

Continuation from last lecture...

More Runtime Analysis

Recap of the 2 approaches - exact count / bounds

\[ \Theta(\cdot), \Omega(\cdot), \omega(\cdot), \Theta(\cdot) = \Theta(\cdot) \]
Claim: For all values of $x$ greater than 2, after two iterations the value decreases by at least 1.

Proof:

Let $x_0$ be the current $x$.

Assume $x_0 > 2$.

Let $x_1$ be $x$ after 1 iteration, $x_2$ after 2 iterations.

We want to prove that $x_2 \leq x_0 - 1$.

Let $x_0 = 4q + r$ where $r \in \{0, 1, 2, 3\}$.

We have $x_1 = 2x_0 - 2 = 2(4q + r) - 2 = 8q + 2r - 2$.

We have $2k = x_0$.

We have $2k + 1 = x_0$.

Q.E.D.
Cases on $r$, for

$3k, 4k+r = x_0$

Case $r = 2$:

So $x_0$ is even ($4k+2$), so branch 1 executes and

$$x_1 = \frac{x_0}{2} = \frac{4k+2}{2} = 2k+1$$

Then $x_1$ is odd, so

$$x_2 = 2(x_1-1) = 2(2k+1) - 1 = 4k+1$$

Analysis of twisty:

Using the claim we proved, after $2k$ iterations,

$$x_{2k} \leq x_0 - k$$

$$x_0 - k \leq 1$$

So after $2k = 2(n-1)$ iterations the function ends... $O(n)$.