Today:

• Runtime analysis where there are more than one inputs of length n

• WORST CASE vs BEST CASE

  Avg case (cater)
We will start with an algorithm where the runtime varies as \( n \to \infty \), so we cannot get a simple \( \Theta \) expression for the runtime.

Our algorithm will be a variation on the famous Collatz function below:

\[
\text{Collatz}(n): \quad n \in \mathbb{N} \\
\text{Repeat while } n > 1 \\
\text{If } n \text{ is even, then } n = \frac{n}{2} \\
\text{If } n \text{ is odd, then } n = 3n + 1
\]

Not known if it always terminates!
Variant of Collatz function

```
Def f(n): n ∈ \mathbb{N}^+

while n > 1:
    if n mod 2 == 0 (n even):
        then n = \frac{n}{2}
    otherwise (n odd)
        = 2n - 2

when n is a power of 2
    then # iterations of while loop \\leq \log_2 n
```

Try n = 15

n ← 30 - 2 = 28
n ← 14
n ← 7
n ← 12
n ← 6
n ← 3
n ← 2
n ← 1
1. $\text{Runtime}_{f}(n) \in \Omega(\log n)$ ?

$\exists c_0 \in \mathbb{R}, \forall n \geq n_0 \quad \text{Runtime}_{f}(n) \geq c_0 \cdot \log n$

2. $\text{Runtime}_{f}(n) \in O(\ ?)$
Claim: If \( n \) is odd, after 2 iterations, it goes down by at least 1.

Ex. \( n = 15 \)

\[
n \leftarrow 2 \cdot 15 - 2 = 28
\]

\( n \leftarrow 14 \)

Proof sketch:

- If \( n \) odd:
  - \( n \rightarrow 2n - 2 \)
  - \( 2n - 2 \rightarrow \frac{2n - 2}{2} = n - 1 \)

  \[\text{after 1st iteration}\]

  \[\text{after 2nd iteration}\]
Claim \( \text{even } \Rightarrow \text{ after 2 iterations } n \text{ goes down by at least 1} \)

Proof

• After one iteration \( n \rightarrow \frac{n}{2} \)

• Case 1: \( \frac{n}{2} \) is even. Then \( \frac{n}{2} \rightarrow \frac{n}{4} \) in 2nd iteration so \( n \) goes down by at least 1 after 2 iterations

• Case 2: \( \frac{n}{2} \) is odd. Then \( \frac{n}{2} \rightarrow 2 \left( \frac{n}{2} \right) - 2 \)
  \[ = n - 2 \]
  so \( n \) goes down by at least 1 after 2 iterations
So in all cases, if $n > 1$, then after 2 iterations, $n$ goes down by at least 1.

\[ \therefore \text{after 2n iterations } n \text{ is 1} \]

So runtime of $f(n)$ is $\leq 2n = O(n)$

Since $\forall n_0 \forall n \in \mathbb{N}^+$ \hspace{1cm} (\hspace{1cm} \forall n \geq n_0 \Rightarrow \text{runtime of } f(n) = \leq c_0 \cdot n \hspace{1cm})$

$c_0 = 2$ \hspace{1cm} $n_0 = 2$
We can also show
runtime is $O(\log_2 n)$

ie. $\exists \epsilon_0, \exists c_0 \forall n \exists n_0 \ (\text{runtime of } f(n) \text{ is } \geq c_0 \cdot \log_2 n)$

↑

To prove this we have to pick $c_0, n_0$
& show that no matter what $n$ is
if $n \geq n_0$ then
runtime on $n \geq c_0 \cdot \log_2 n$

why?
Proof that Runtime is $\Omega(\log n)$ [pf sketch]

At each iteration the value of $x$ drops by at most $\sqrt{2}$

So $\text{Runtime}(n) = 1 + \text{Runtime}(\frac{n}{2})$ 

$\geq \log n$

where $\text{Runtime}(x) =$ # executions of while loop on input $n$

$\therefore \exists c_0 \in \mathbb{R}_+ \text{ where } \forall n \geq n_0, \text{Runtime}(n) \geq c_0 \cdot \log n$
So we have

1. \( \text{Runtime}(n) \in O(n) \)
2. \( \text{Runtime}(n) \in \Omega(\log n) \)

They don't match!

Show by a better analysis we can show

\( \text{Runtime}(n) \in O(\log n) \)

So \( \text{Runtime}(n) = \Theta(\log n) \)
To show runtime \( n(n) \in O(\log n) \)

consider 3 consecutive iterations of algorithm + show \( n \rightarrow \frac{n}{2} \)

4 cases

1. \( n \) even  \( \rightarrow \) odd  \( \rightarrow \) even  \( \rightarrow \)
   \( n \rightarrow \frac{n}{2} \rightarrow 2(\frac{n}{2}) - 2 \rightarrow \frac{n-2}{2} < \frac{n}{2} \)

2. \( n \) even  \( \rightarrow \) even  \( \rightarrow \) odd  \( \rightarrow \)
   \( n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow 2(\frac{n}{4}) - 2 = \frac{n}{2} - 2 < \frac{n}{2} \)

3. \( n \) even  \( \rightarrow \) even  \( \rightarrow \) even  \( \rightarrow \)
   \( n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \)

4. \( n \) odd  \( \rightarrow \) even  \( \rightarrow \) odd  \( \rightarrow \)
   \( n \rightarrow \frac{n}{2} \rightarrow 2n - 2 \rightarrow n - 1 \)

5. \( n \) odd  \( \rightarrow \) even  \( \rightarrow \) even  \( \rightarrow \)
   \( n \rightarrow \frac{n}{2} \rightarrow 2n - 2 \rightarrow n - 1 \rightarrow \frac{n-1}{2} < \frac{n}{2} \)

:. In all cases \( n \rightarrow \frac{n}{2} \) after 3 iterations.

so runtime \( \leq 3 \cdot \log_2 n = O(\log n) \)
Now we want to consider algorithms that have multiple inputs of length n. How to measure runtime?
Now we want to consider algorithms that have multiple inputs of length \( n \). How to measure runtime?

\[ A(n) \]
Example

```python
from numbers import numbers

def has_even(numbers):
    for number in numbers:
        if number % 2 == 0:
            return True
    return False

def numbers:
    return [3, 7, 9, 2, 5, 4, 8]

size = n = 7

Ex numbers: 3, 7, 9, 2, 5, 4, 8

Let runtime be the number of executions of the for-loop
```
Define worst-case runtime

\[ WC_f(n) = \max \left\{ \# \text{ of steps when executing } f(x) \mid x \text{ has length } n \right\} \]

\[ BC_f(n) = \min \left\{ \# \text{ of steps when executing } f(x) \mid x \text{ has length } n \right\} \]

\[ \text{Ex} \quad WC_{\text{has-even}}(n) = n = \Theta(n) \quad \text{← we hope for a } \Theta(g(n)) \]

\[ BC_{\text{has-even}}(n) = 1 = \Theta(1) \quad \text{← and } \Theta(h(n)) \]
More formally, to show \( WC_f(n) \leq O(g(n)) \):

we have to show
\[
\exists n_0, c_0 \text{ s.t. } \forall n \geq n_0 \quad WC_f(n) \leq c_0 \cdot g(n)
\]

\[
= \exists n_0, c_0 \forall n \geq n_0 \quad \max \{ \text{Runtime}_f(x) \} \leq c_0 \cdot g(n)
\]

inputs \( x \) of length \( n \)

\[
= \exists n_0, c_0 \forall n \geq n_0 \quad \forall x \quad [ \text{If } x \text{ has length } n \Rightarrow \text{Runtime}_f(x) \leq c_0 \cdot g(n) ]
\]

In general, to prove \( WC_f(n) \in O(g(n)) \) you usually do a static analysis of code.

For \( WC \) has even \( (n) \in O(n) \)

since \( n_0 = 1, c_0 = 1 \). Argue for any list of length \( n \), the for loop executes at most \( n \) times

so runtime on all \( x, |x| = n \) is \( n \).
More formally, to show \( WC_f(n) \leq O(g(n)) \):
we have to show
\[
\exists n_0, c_0 \ \forall n \geq n_0 \ WC_f(n) \leq c_0 \cdot g(n)
\]
\[
\equiv \exists n_0, c_0 \ \forall n \geq n_0 \ \left[ \max_{\|x\| = n} \{ \text{Runtime}_f(x) \} \leq c_0 \cdot g(n) \right]
\]
\[
\equiv \exists n_0, c_0 \ \forall n \geq n_0 \ \forall x \ (\|x\| = n \Rightarrow \text{Runtime}_f(x) \leq c_0 \cdot g(n))
\]

Similarly to show \( WC_f(n) \in \Omega(g(n)) \)
we have to show:
\[
\exists n_0, c_0 \ \forall n \geq n_0 \ WC_f(n) \geq c_0 \cdot g(n)
\]
\[
\equiv \exists n_0, c_0 \ \forall n \geq n_0 \ \left[ \max_{\|x\| = n} \{ \text{Runtime}_f(x) \} \geq c_0 \cdot g(n) \right]
\]
\[
\equiv \exists n_0, c_0 \ \forall n \geq n_0 \ \exists x \ [ \|x\| = n \land \text{Runtime}_f(x) \geq c_0 \cdot g(n)]
\]
Now to show $BC_f(n) \in O(g(n))$

we have to show

$$\exists n_0, c_0 \ \forall n \geq n_0 \ \ BC_f(n) \leq c_0 \cdot g(n)$$

$$\exists n_0, c_0 \ \forall n \geq n_0 \ \left[ \min_{x, |x| = n} \{ \text{Runtime}_f(x) \} \leq c_0 \cdot g(n) \right]$$

$$\exists n_0, c_0 \ \forall n \geq n_0 \ \exists x \ (|x| = n \land \text{Runtime}_f(x) \leq c_0 \cdot g(n))$$

To show $BC_f(n) \in \Omega(g(n))$

we have to show

$$\exists n_0, c_0 \ \forall n \geq n_0 \ \ BC_f(n) \geq c_0 \cdot g(n)$$

$$\exists n_0, c_0 \ \forall n \geq n_0 \ \left[ \min_{x, |x| = n} \{ \text{Runtime}_f(x) \} \geq c_0 \cdot g(n) \right]$$

$$\exists n_0, c_0 \ \forall n \geq n_0 \ \forall x \ (|x| = n \Rightarrow \text{Runtime}_f(x) \geq c_0 \cdot g(n))$$
\[WC_f(n) \in O(g(n)):\]
\[
\exists n_0, c_0 \, \forall n \geq n_0 \, \forall x \ [\|x\| = n \Rightarrow \text{Runtime}_f(x) \leq c_0 \cdot g(n)]
\]

\[WC_f(n) \in \Omega(g(n)):\]
\[
\exists n_0, c_0 \, \forall n \geq n_0 \, \exists x \ [\|x\| = n \land \text{Runtime}_f(x) \geq c_0 \cdot g(n)]
\]

\[BC_f(n) \in O(g(n)):\]
\[
\exists n_0, c_0 \, \forall n \geq n_0 \, \exists x \ [\|x\| = n \land \text{Runtime}_f(x) \leq c_0 \cdot g(n)]
\]

\[BC_f(n) \in \Omega(g(n)):\]
\[
\exists n_0, c_0 \, \forall n \geq n_0 \, \forall x \ [\|x\| = n \Rightarrow \text{Runtime}_f(x) \geq c_0 \cdot g(n)]
\]
Example: Palindromes

A palindrome is a string (sequence of letters) that reads the same forwards as backwards.

Ex. MOM, DAD, abbcbbba, abbcbbba

\[ \text{not palindromes} \]

Our algorithm: find longest prefix of an input string that is a palindrome.
Def Pal(s):  # s is a string  s[1] ... s[n]

n = len(s)

For k = n, n-1, ..., 2, 1

    is_pal = True
    For j = k, k-1, ..., 1
        # check if s[i ... j] is a palindrome
        If s[j] != s[k-j+1]
            then is_pal = False

If is_pal = true:
    return k

\[ \sum_{i=1}^{n} \sum_{j=1}^{\frac{n}{2}} i \cdot \frac{n}{2} = O(n^2) \]
Worst Case Runtime

(∀n, ∀x (x has length n ⇒ Runtime_{par}(x) ≤ ?))

The inner loop executes at most

\[ \frac{n + n-1 + \ldots + 1}{2} \] times

\[ = \frac{n(n+1)}{2} \in O(n^2) \]
\[ 2 \quad WC_{\text{pal}}(n) \in \mathcal{O}(n^2) \]

\( \forall n \exists x \ (x \text{ has length } n \text{ and } \text{runtime}_{\text{pal}}(x) \geq ?) \)

Let \( x = abbb \ldots b \)

Pal on this string \( x \)
executes inner loop
\[ n + n - 1 + \ldots + 1 \text{ time } \in \mathcal{O}(n^2) \]

\[ \therefore WC_{\text{pal}}(n) = \Theta(n^2) \]
Best Case Runtime of Pal

1. \( BC_{\text{pal}}(n) \in O(\ ?) \)

\( \forall n \exists x \ (x \text{ has length } n \text{ and } \text{Runtime}_{\text{pal}}(x) \leq \ ?) \)

Let \( x \) be a string of \( n \) \( a \)'s

on this \( x \), alg takes time \( n \)

so \( BC_{\text{val}}(n) \in O(n) \)
\[ B_{\text{par}}(n) \leq \mathcal{O}(n) \]

Show that if \( x \) has length \( n \), then runtime \( (x) \geq ? \)

No matter what \( x \) is (of length \( n \))

The outer loop executes at least once, when \( K = n \)

... then the inner loop executes \( n \) times

So runtime on every \( x \) of length \( n \) is \( \geq n \)

\[ \therefore B_{\text{par}}(n) = \Theta(n) \]
Next let's modify our algorithm so that inner loop breaks/quit as soon as it discovers the current prefix is not a palindrome.
Define `Pal(s):` # s is a string

```python
n = len(s)

For k = n, n-1, ..., 2, 1

    is_pal = True

    For j = k, k-2, ..., 1
        # check if s[1..k] is a palindrome
        If s[j] != s[k-j+1]
            then is_pal = False

    Break out of inner loop

If is_pal = true
    return K
```

Now: Analysis
with the
`break` stmt

```python
acaacabb
```
Show \( WC_{\text{PAL}}(n) \in O(n^2) \)

Outer loop: executes at most \( n \) times

1. \( k = n \): 2 cases:
   1. Never execute "Break" statement \( \Rightarrow \) find palindrome
   2. We do eventually "Break" \( \Rightarrow \) this must happen after \( \leq n-1 \) executions of inner loop
   \( \Rightarrow n \)

2. \( k = n-1 \)
   \( \Rightarrow \) inner loop executes after \( \leq n-2 \) iterations
   \( \Rightarrow \) so on: \( (n-1)+(n-2)+\ldots+1 \in O(n^2) \)
To prove $\text{WC}_{\text{pal}}(n) \notin \Omega(n^2)$ is trickier with the "Break" statement.

We want to show:

\[ \exists x \text{ s.t. runtime of algorithm on } x \text{ is } \Omega(n^2) \]

Our old string: $\text{abbbbbb...}$

- $k = n$
- $k = n - 1$
- $\vdots$
- $k = 2$
- $k = 1$

All runtime is only \( n \).
Our old family of strings won't work anymore since they have runtime $\Theta(n)$.

So we need to find a new family of strings (one string of length $n$ for every $n$) that take time $\Omega(n^2)$. 
We need to find a string $x$, $|x|=n$ s.t. runtime is $\sim n^2$.

$n = \text{even}$

\[
\begin{array}{cccccccccccc}
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\frac{n}{2} \text{a}'s & \uparrow & b & \frac{n-1}{2} \text{a}'s
\end{array}
\]

$n = \text{odd}$

\[
\begin{array}{cccccccccccc}
\text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
\left\lfloor \frac{n}{2} \right\rfloor \text{a}'s & \uparrow & b & \frac{n-1}{2} \text{a}'s
\end{array}
\]
Runtime on $x$, $n$ even:

\[
\frac{n}{2} + \frac{n}{2} - 1 + \frac{n}{2} - 2 + \ldots + \frac{n}{2} - \frac{n}{2}
\]

\[
= \frac{n}{2} + \ldots + 2 + 1
\]

\[
= \frac{(\frac{n}{2})(4 - 1)}{2} = \Theta(n^2)
\]

Note that for all $K$, $n \geq K \geq \frac{n}{2} + 1$, the prefix $X[1] \ldots X[K]$ is not a palindrome.

# executions of inner loop is at least

Runtime for $x$, $x$ even: similar analysis gives $\Omega(n^2)$

\[\therefore W_{RC_{PAL}}(n) = \Theta(n^2)\]
What about Best Case complexity?

$BC_{PAL}(n) \in O(n)$

Show: An $x$ of length $n$

such that algorithm on $x$

takes time $\leq \Theta(n)$

Example: Let $x$ be a palindrome, say $x = \text{all } a$'s

Then alg takes $\Theta(n)$ time
Now show $BC_{\text{PAL}}(n) = \mathcal{O}(n)$:

Now we need to show

For all $x$ if $x$ has length $n$

then runtime on $x$ is $\mathcal{O}(n)$

Need to show that inner loop always executes a total of $\mathcal{O}(n)$ times
Look at cases:

1. $x$ is a palindrome.
   \[\text{Then outer loop executes once when } k=n\]
   \[\text{inner loop executes } n \text{ times}\]

2. Not a palindrome: let $x \ldots x_i$ be the largest prefix that is a palindrome
   \[\text{Then}\]
   \[\text{outer loop executes } (n-i-1) \text{ times unsuccessfully}\]
   \[\text{in each case inner loop executes at least once}\]
   \[\text{outer loop executes successfully when } k=i\]
   \[\text{inner loop executes } i \text{ times}\]

In total: $(n-i-1)+i = n-1 = \mathcal{O}(n)$