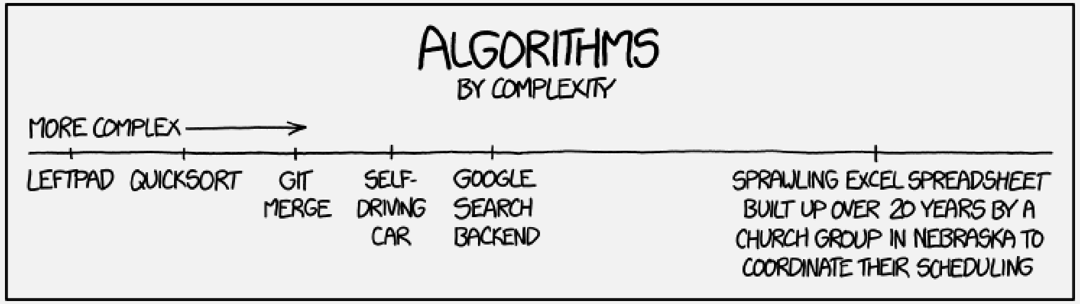
UMB CS 420

# Time Complexity

Wednesday, April 13, 2022

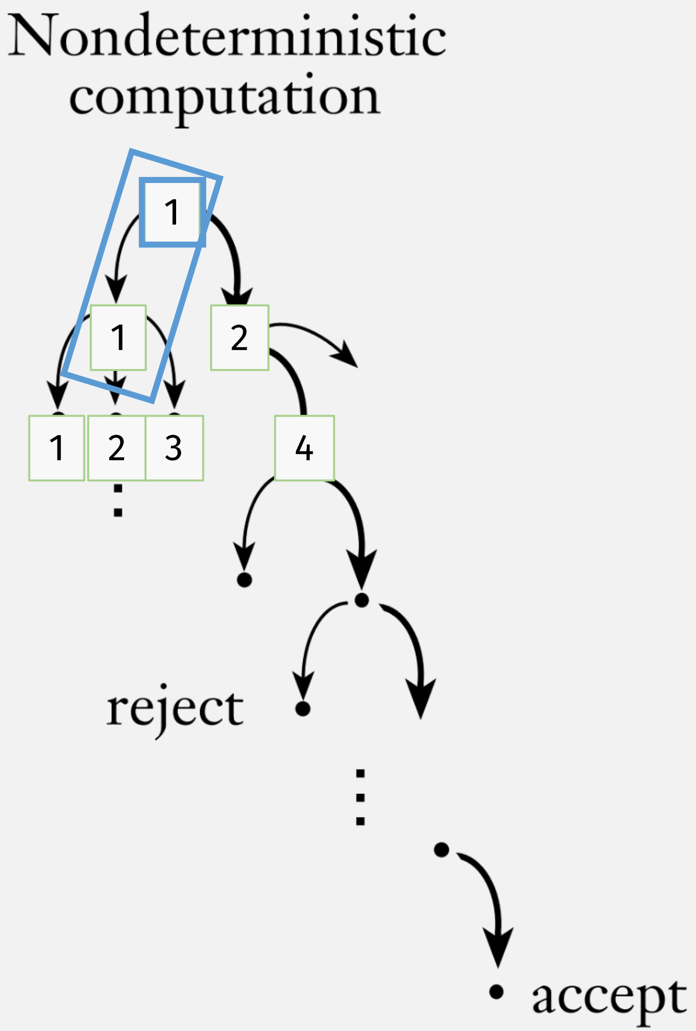


# Announcements

- HW 9 extended
  - ~~Due Sunday 4/17 11:59pm~~
  - Due Tuesday 4/19 11:59pm
- No class next Monday 4/18
  - Patriots Day
- HW 10 out next Wed 4/20
  - Due Tuesday 4/26 11:59pm

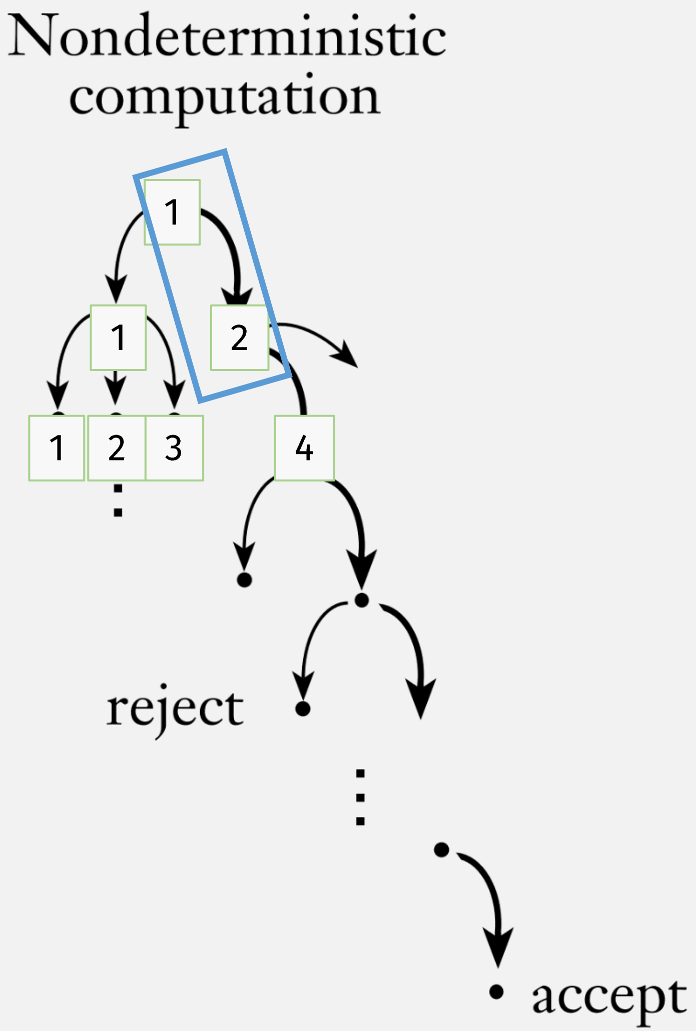
# *Flashback:* Nondet. TM → Deterministic TM

- To simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - Root node: 1
    - **1-1**



# *Flashback:* Nondet. TM → Deterministic TM

- To <u>simulate</u> NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - Root node: **1**
    - 1-1
    - **1-2**



# *Flashback:* Nondet. TM → Deterministic TM

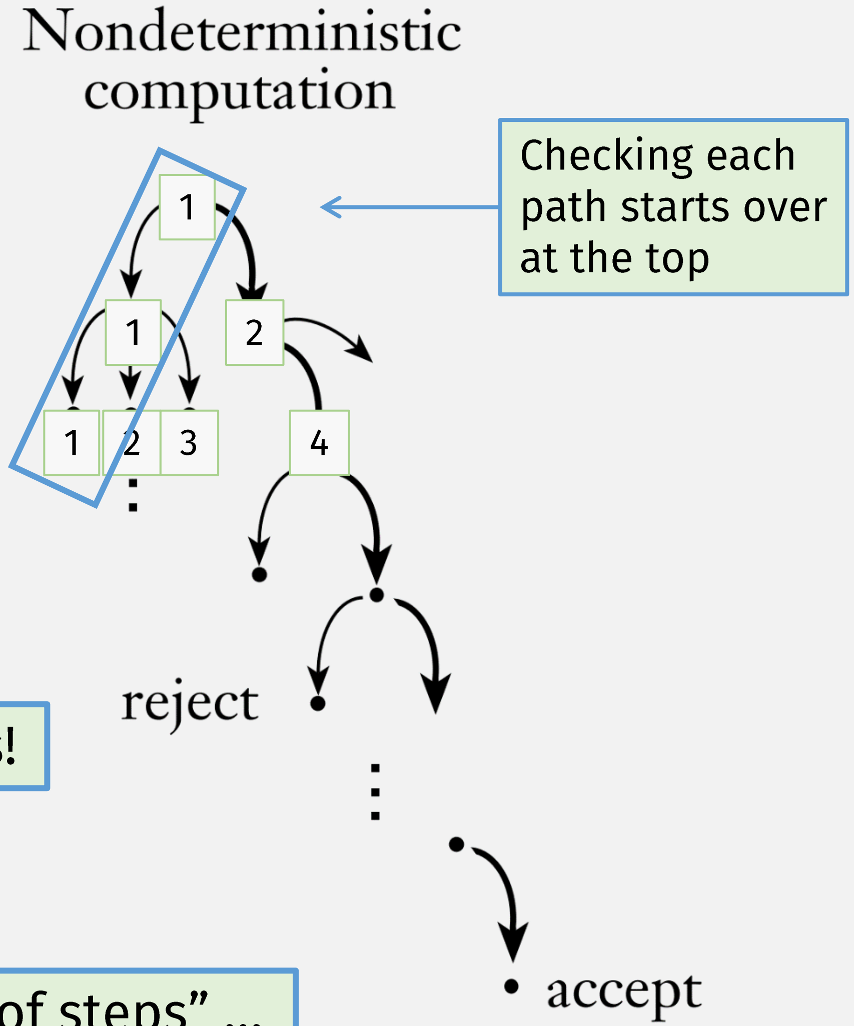
- To <u>simulate</u> NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - Root node: 1
    - 1-1
    - 1-2
    - **1-1-1**

A TM and a NTM are "equivalent" ...

.. but **not** if we care about the # of steps!

So how inefficient is it?

First, we need a formal way to count "# of steps" ...



# A Simpler Example: $A = \{0^k 1^k | k \geq 0\}$

$M_1$ = "On input string $w$:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape:
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, *reject*. Otherwise, if neither 0s nor 1s remain on the tape, *accept*."

# of steps (worst case), $n$ = length of input:

- TM Line 1:
  - $n$ steps to scan + $n$ steps to return to beginning = **$2n$ steps**

# A Simpler Example: $A = \{0^k 1^k \mid k \geq 0\}$

$M_1$ = "On input string $w$:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape:
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, *reject*. Otherwise, if neither 0s nor 1s remain on the tape, *accept*."

**# of steps** (worst case), ***n*** **= length of input:**

- TM Line 1:
  - ***n*** steps to scan + ***n*** steps to return to beginning = ***2n* steps**

➢ Lines 2-3 (loop):

- steps/iteration (line 3): ***n*/2** steps to find "1" + ***n*/2** steps to return = ***n*** steps
- # iterations (line 2): Each scan crosses off 2 chars, so at most ***n*/2** scans
- Total = steps/iteration * # iterations = ***n* (*n*/2)** = **$n^2/2$ steps**

# A Simpler Example: $A = \{0^k 1^k | k \geq 0\}$

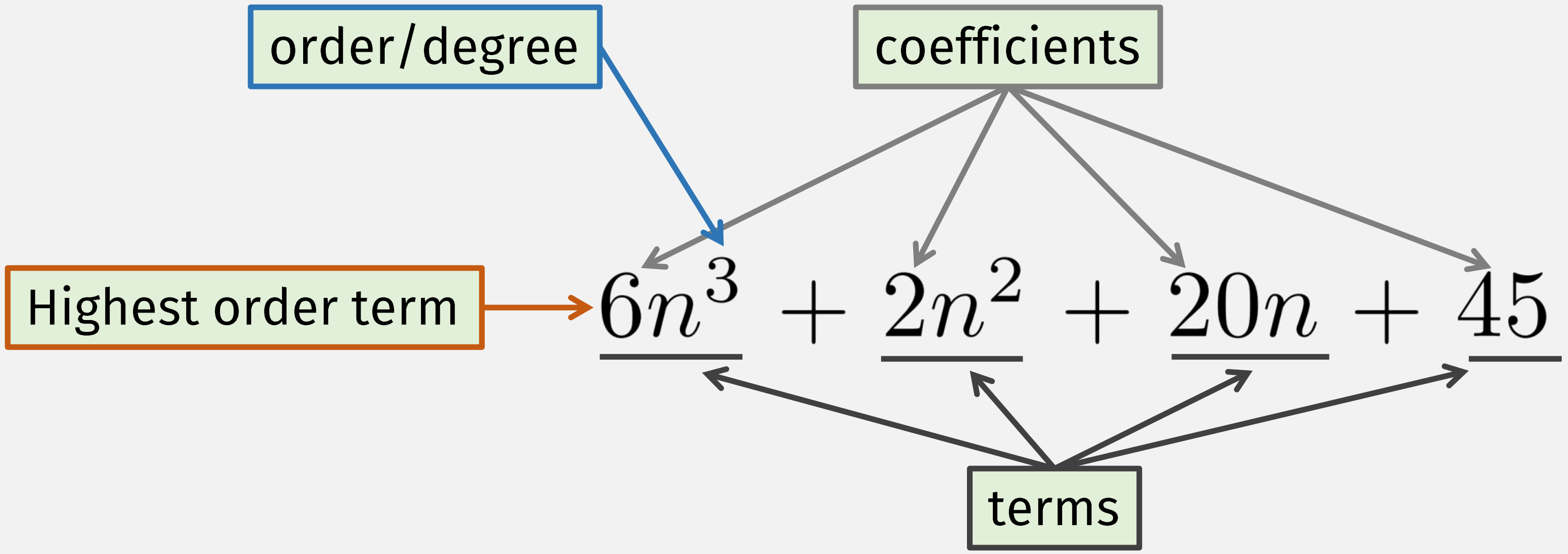
$M_1$ = "On input string $w$:
1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape:
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, *reject*. Otherwise, if neither 0s nor 1s remain on the tape, *accept*."

$n^2/2 + 3n$

**# of steps** (worst case), $n$ = length of input:

- TM Line 1:
  - $n$ steps to scan + $n$ steps to return to beginning = **$2n$ steps**
- Lines 2-3 (loop):
  - steps/iteration (line 3): $n/2$ steps to find "1" + $n/2$ steps to return = $n$ steps
  - # iterations (line 2): Each scan crosses off 2 chars, so at most $n/2$ scans
  - Total = steps/iteration * # iterations = $n(n/2)$ = **$n^2/2$ steps**
- ➢ Line 4:
  - **$n$ steps** to scan input one more time
- Total: $2n + n^2/2 + n$ = **$n^2/2 + 3n$ steps**

# Interlude: Polynomials



# Definition: Time Complexity

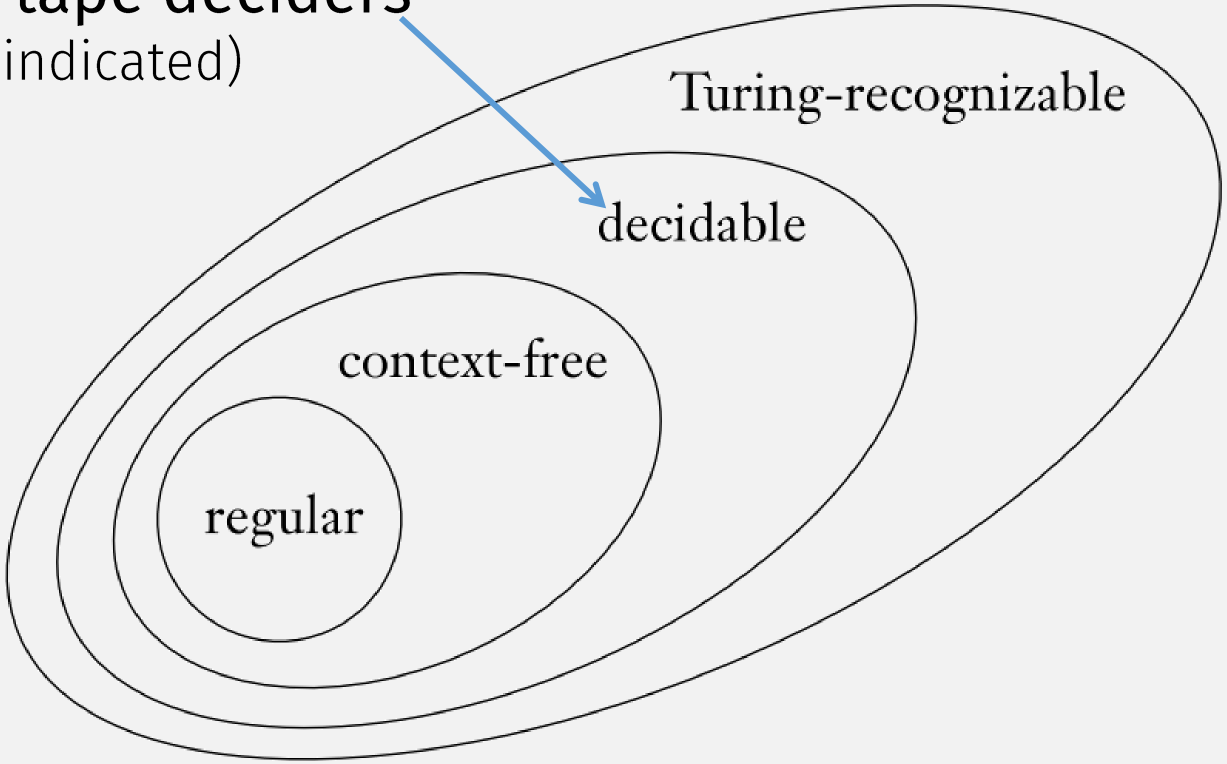
i.e., a decider (algorithm)

Let $M$ be a deterministic Turing machine that halts on all inputs. The ***running time*** or ***time complexity*** of $M$ is the function $f\colon \mathcal{N} \longrightarrow \mathcal{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$. If $f(n)$ is the running time of $M$, we say that $M$ runs in time $f(n)$ and that $M$ is an $f(n)$ time Turing machine. Customarily we use $n$ to represent the length of the input.

**Running Time** / **Time Complexity** is a property of a (Turing) Machine

# Where Are We Now?

**We are back in here now:**
deterministic, single-tape deciders
(unless otherwise indicated)



# Definition: Time Complexity

**NOTE:** $n$ has no units, it's only *roughly* "length" of the input

$n$ can be:
# characters,
# states,
# nodes, …

We can use any $n$ that is correlated with the input length

Let $M$ be a deterministic Turing machine that halts on all inputs. The ***running time*** or ***time complexity*** of $M$ is the function $f: \mathcal{N} \longrightarrow \mathcal{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$. If $f(n)$ is the running time of $M$, say that $M$ runs in time $f(n)$ and that $M$ is an $f(n)$ time Tur- machine. Customarily we use $n$ to represent the length of the ut.

- Machine $M_1$ that decides $A = \{0^k 1^k | k \geq 0\}$
  - Running time / Time Complexity: $n^2/2+3n$

$M_1$ = "On input string $w$:
1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape:
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, *reject*. Otherwise, if neither 0s nor 1s remain on the tape, *accept*."

# Interlude: Asymptotic Analysis

Total: $n^2 + 3n$

- If $n = 1$
  - $n^2 = 1$
  - $3n = 3$
  - Total = 4
- If $n = 10$
  - $n^2 = 100$
  - $3n = 30$
  - Total = 130
- If $n = 100$
  - $n^2 = 10{,}000$
  - $3n = 300$
  - Total = 10,300
- If $n = 1{,}000$
  - $n^2 = 1{,}000{,}000$
  - $3n = 3{,}000$
  - Total = 1,003,000

$n^2 + 3n \approx n^2$ as n gets large

asymptotic analysis only cares about **large** $n$

# Definition: Big-$O$ Notation

Let $f$ and $g$ be functions $f, g\colon \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $\boldsymbol{f(n) = O(g(n))}$ if positive integers $c$ and $n_0$ exist such that for every integer $n \geq n_0$,

$$f(n) \leq c\,g(n).$$

"only care about **large** $n$"

When $f(n) = O(g(n))$, we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

In other words: **Keep only highest order term, drop all coefficients**

- Machine $M_1$ that decides $A = \{0^k 1^k \mid k \geq 0\}$
  - is an $\boldsymbol{n^2 + 3n}$ time Turing machine
  - is an $O(\boldsymbol{n^2})$ time Turing machine
  - has **asymptotic upper bound** $O(\boldsymbol{n^2})$

# Definition: Small-$o$ Notation (less used)

Let $f$ and $g$ be functions $f, g\colon \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $\boldsymbol{f(n) = o(g(n))}$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

In other words, $f(n) = o(g(n))$ means that for any real number $c > 0$, a number $n_0$ exists, where $f(n) < c\, g(n)$ for all $n \geq n_0$.

Analogy: Big-$O$ : ≤ :: small-$o$ : <

Let $f$ and $g$ be functions $f, g\colon \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $\boldsymbol{f(n) = O(g(n))}$ if positive integers $c$ and $n_0$ exist such that for every integer $n \geq n_0$,

$$f(n) \leq c\, g(n).$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

# Big-$O$ arithmetic

- $O(\boldsymbol{n^2}) + O(\boldsymbol{n^2})$
  $= O(\boldsymbol{n^2})$

- $O(\boldsymbol{n^2}) + O(\boldsymbol{n})$
  $= O(\boldsymbol{n^2})$

- $\boldsymbol{2n} = O(\boldsymbol{n})$ ?
  - TRUE

- $\boldsymbol{2n} = O(\boldsymbol{n^2})$ ?
  - TRUE

- $\boldsymbol{1} = O(\boldsymbol{n^2})$ ?
  - TRUE

- $\boldsymbol{2^n} = O(\boldsymbol{n^2})$ ?
  - FALSE

NOTE: **Other courses** might use Big-$\Theta$ notation, which is a tighter bound where some of these equalities won't be true, e.g., $\boldsymbol{2n} \neq \Theta(\boldsymbol{n^2})$

NOTE: **In this course,** we use Big-$O$ only, not Big-$\Theta$ (so do not confuse the two)

# Definition: Time Complexity Classes

Let $t\colon \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. Define the ***time complexity class***, $\mathbf{TIME}(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

Remember: **TMs** have a time complexity (i.e., a running time), **languages** are in a time complexity class

The complexity class of a **language** is determined by the time complexity (running time) of its deciding **TM**

A **language** could be in more than one **time complexity class**

- Machine $M_1$ decides language $A = \{0^k 1^k | k \geq 0\}$
  - $M_1$ has time complexity (running time) of $O(n^2)$
  - $A$ is in time complexity class $\mathbf{TIME}(n^2)$

# A Faster Machine? $A = \{0^k 1^k \mid k \geq 0\}$

$M_2$ = "On input string $w$:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

Previously:

$M_1$ = "On input string $w$:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape:
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, *reject*. Otherwise, if neither 0s nor 1s remain on the tape, *accept*."

# A Faster Machine? $A = \{0^k 1^k | k \geq 0\}$

$M_2$ = "On input string $w$:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

**Number of steps** (worst case), ***n*** **= length of input:**

- Line 1:
  - ***n*** steps to scan + ***n*** steps to return to beginning = $O(n)$ steps

# A Faster Machine? $A = \{0^k 1^k | k \geq 0\}$

$M_2$ = "On input string $w$:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

**Number of steps** (worst case)**,** $n$ = **length of input:**

- Line 1:
  - $n$ steps to scan + $n$ steps to return to beginning = $O(n)$ steps
- Lines 2-4 (loop):
  - steps/iteration (lines 3-4): a scan takes $O(n)$ steps
  - # iters (line 2): Each iter crosses off *half* the chars, so at most $O(\log n)$ scans
  - Total: $O(n) * O(\log n)$ = $O(n \log n)$ steps

# Interlude: Logarithms (dual to exponentiation)

- If $2^x = y$ …
- … then $\log_2 y = x$
- $\log_2 n = O(\mathbf{log}\ \boldsymbol{n})$
  - "divide and conquer" algorithms = $O(\mathbf{log}\ \boldsymbol{n})$
  - E.g., binary search
- (In computer science, **base-2 is the only base!**)

# A Faster Machine? $A = \{0^k 1^k \mid k \geq 0\}$

$M_2$ = "On input string $w$:

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

**Number of steps** (worst case), ***n*** **= length of input:**

- Line 1:
  - ***n*** steps to scan + ***n*** steps to return to beginning = $O(n)$ steps
- Lines 2-4 (loop):
  - steps/iteration (lines 3-4): a scan takes $O(n)$ steps
  - # iters (line 2): Each iter crosses off *half* the chars, so at most $O(\log n)$ scans
  - Total: $O(n) * O(\log n)$ = $O(n \log n)$ steps

➢ Line 5:
  - $O(n)$ steps to scan input one more time

# A Faster Machine? $A = \{0^k1^k | k \geq 0\}$

$M_2$ = "On input string $w$:
1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and some 1s remain on the tape:
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, *reject*.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, *accept*. Otherwise, *reject*."

$O(n \log n)$

Prev: $n^2/2 + 3n = O(n^2)$

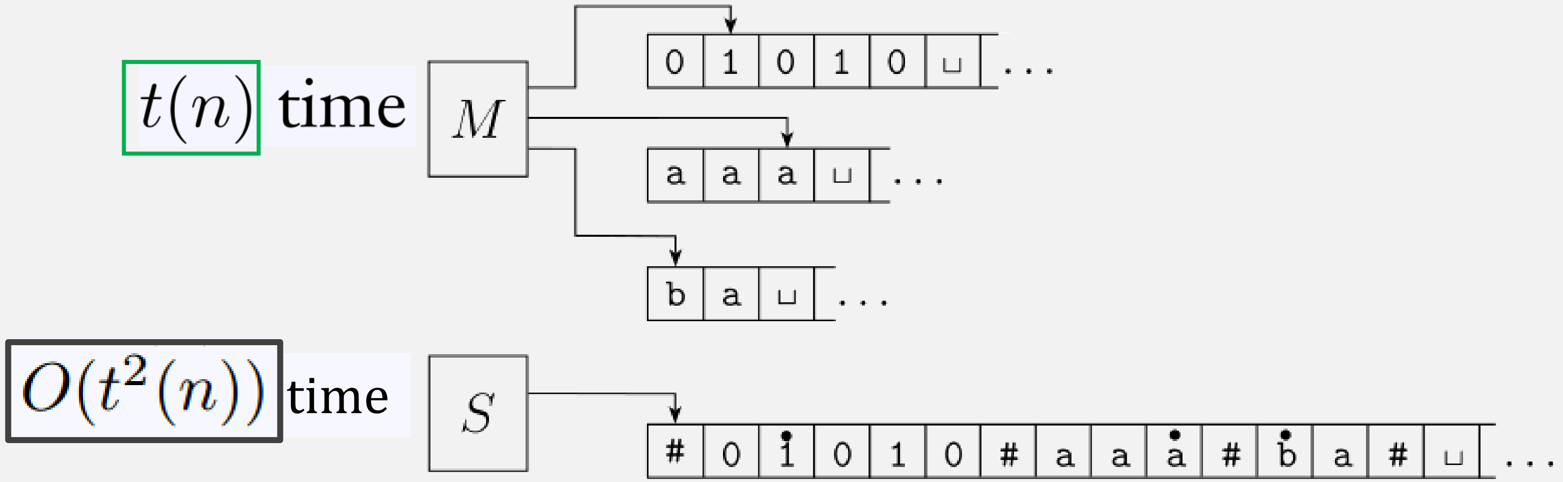
**Number of steps** (worst case), $n$ = **length of input:**

- Line 1:
  - $n$ steps to scan + $n$ steps to return to beginning = $O(n)$ steps
- Lines 2-4 (loop):
  - steps/iteration (lines 3-4): a scan takes $O(n)$ steps
  - # iters (line 2): Each iter crosses off *half* the chars, so at most $O(\log n)$ scans
  - Total: $O(n) * O(\log n)$ = $O(n \log n)$ steps
- Line 5:
  - $O(n)$ steps to scan input one more time
- Total: $O(n) + O(n \log n) + O(n)$ =

# *Terminology:* Categories of Bounds

- Exponential time
  - $O(2^{n\wedge c})$, for $c > 0$, or $2^{O(n)}$ (always base 2)
- Polynomial time
  - $O(n^c)$, for $c > 0$
- Quadratic time (special case of polynomial time)
  - $O(n^2)$
- Linear time (special case of polynomial time)
  - $O(n)$
- Log time
  - $O(\log n)$

# Multi-tape vs Single-tape TMs: # of Steps




- For single-tape TM to simulate 1 step of multi-tape:
  1. Scan to find all "heads" = $O$(length of all $M$'s tapes)
  2. "Execute" transition at all the heads = $O$(length of all $M$'s tapes)
- # single-tape steps to simulate 1 multitape step (worst case)
  - = $O$(length of all $M$'s tapes)
  - = $O(t(n))$, If $M$ spends all its steps expanding its tapes
- Total steps (single tape): $O(t(n))$ per step × $t(n)$ steps =

# *Flashback:* Nondet. TM → Deterministic TM

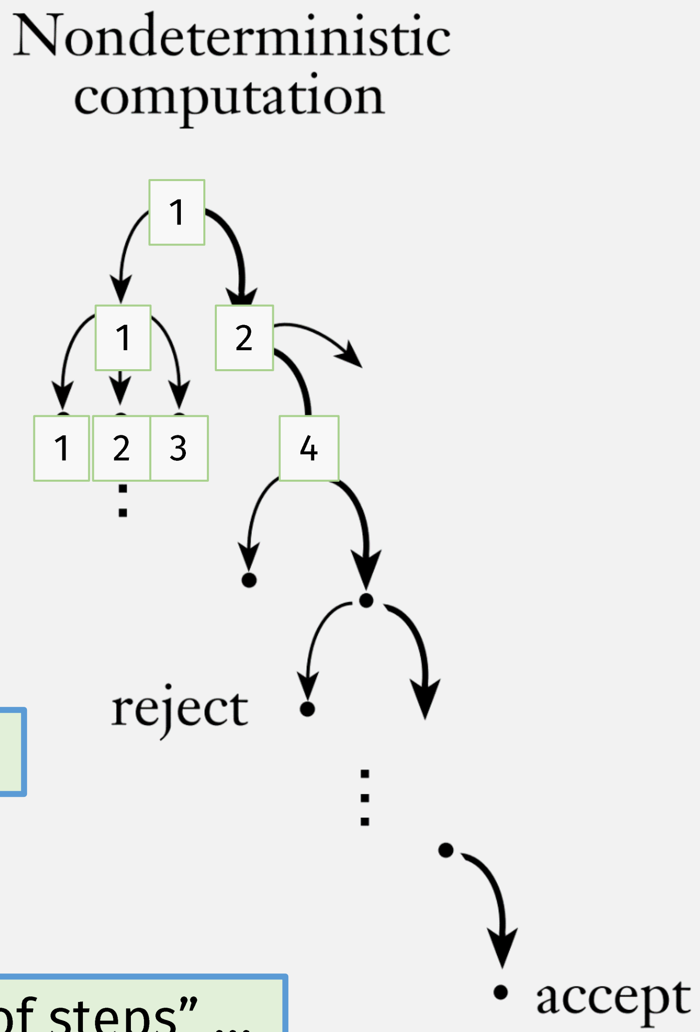
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - 1-2
    - **1-1-1**

A TM and a NTM are "equivalent" …

.. but **not if we care about the # of steps**

How **inefficient is it?**

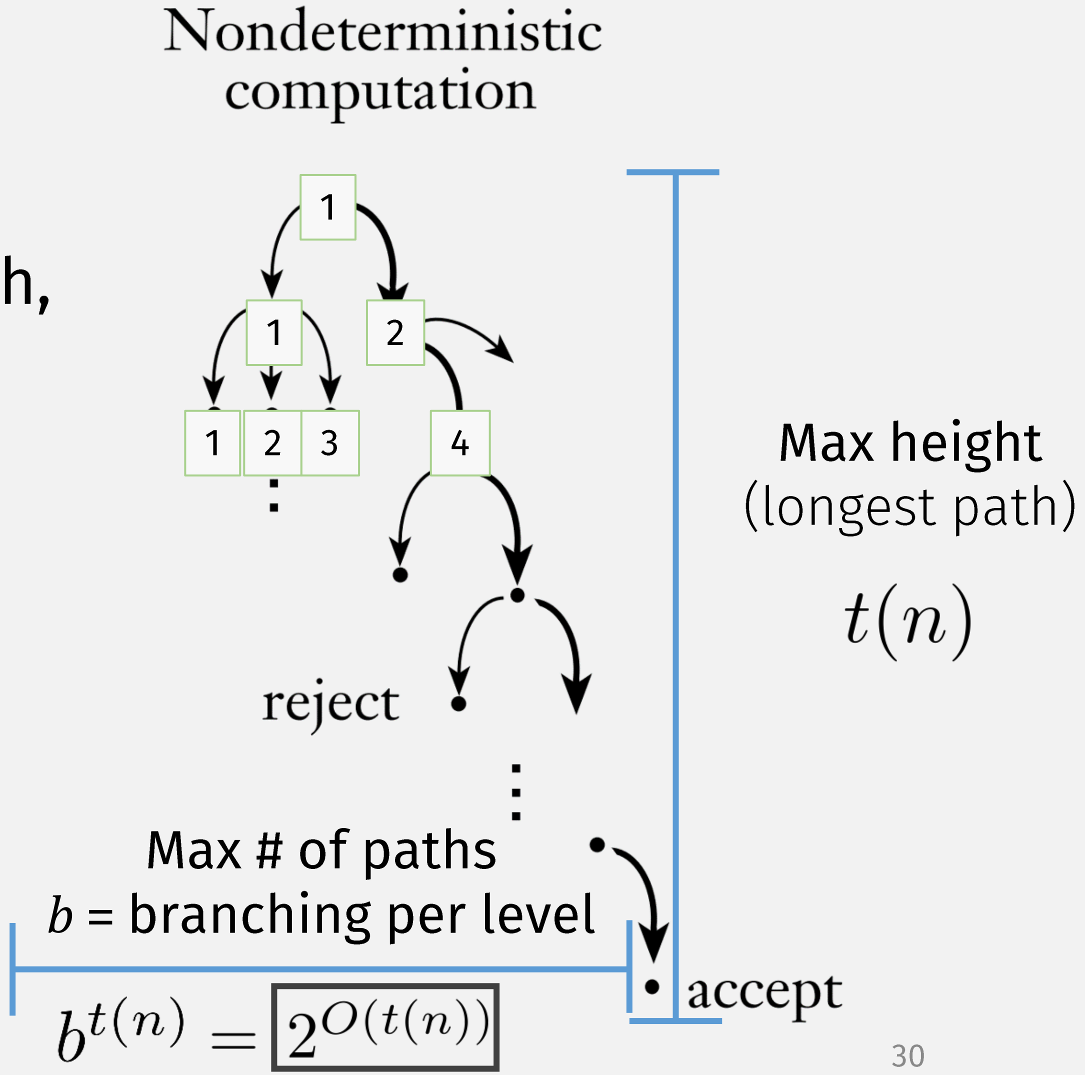
First, **we need a formal way to count "# of steps" …**



# *Flashback:* Nondet. TM → Deterministic TM

$t(n)$ time → $\boxed{2^{O(t(n))}}$ time

- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - 1-2
    - **1-1-1**



# Summary: TM Variations

- If **multi-tape TM:** $t(n)$ time
- Then **equivalent single-tape TM:** $O(t^2(n))$
  - **Quadratically** slower

- If **non-deterministic TM:** $t(n)$ time
- Then **equivalent single-tape TM:** $2^{O(t(n))}$
  - **Exponentially** slower

# Check-in Quiz 4/13

On gradescope