CS420
Time Complexity

Wed November 18, 2020

HW questions?

Announcements

- Graded HW6 returned
- HW9 released

Flashback: Single-tape TM “equiv to“ Nondet. TM

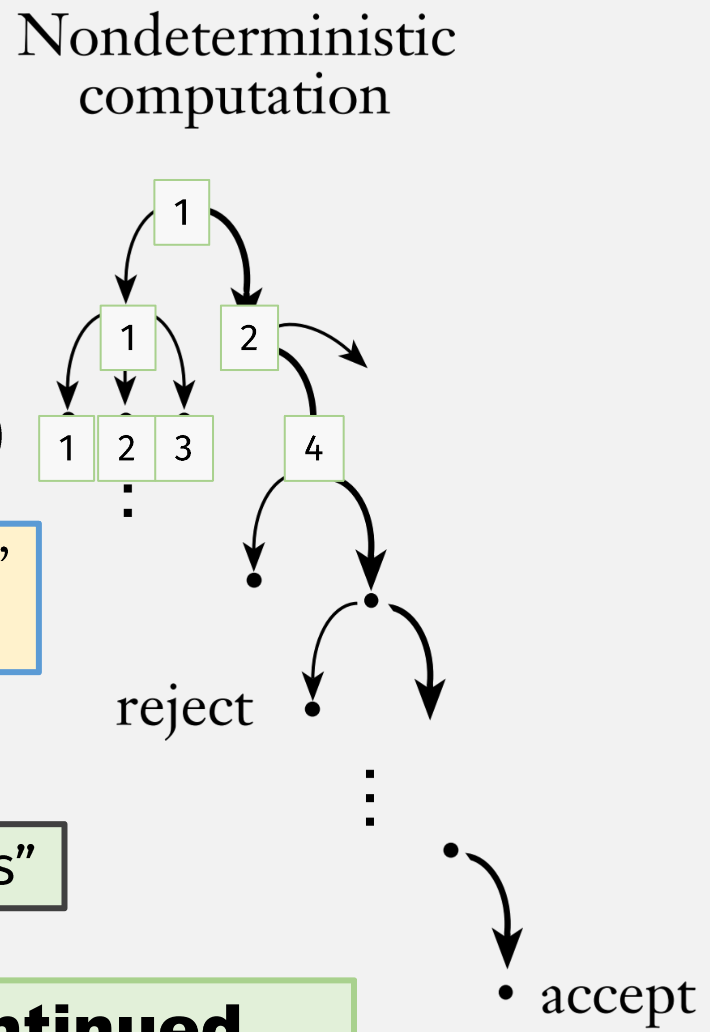
Flashback: Single-tape TM "equiv to" Nondet. TM

- Deterministic TM simulating nondeterministic TM:
  - Number the nodes at each step
  - Deterministically check every path, in breadth-first order (restart at top each time)
    - 1
    - 1-1
    - 1-2
    - 1-1-1
    - 1-1-2
    - and so on
  - Accept if accepting config found

"This is the most inefficient algorithm ever"
--- CS420 Fall2020 class

But, exactly how inefficient is it???

Now we'll start to count "# of steps"

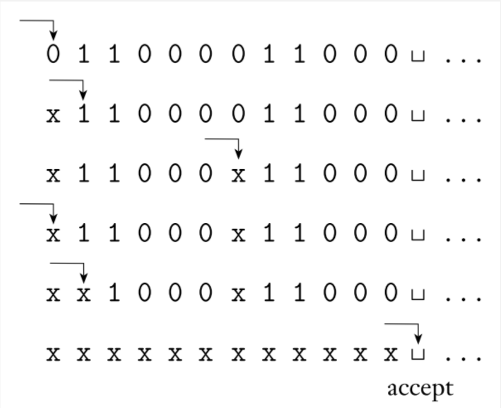


To be continued …

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

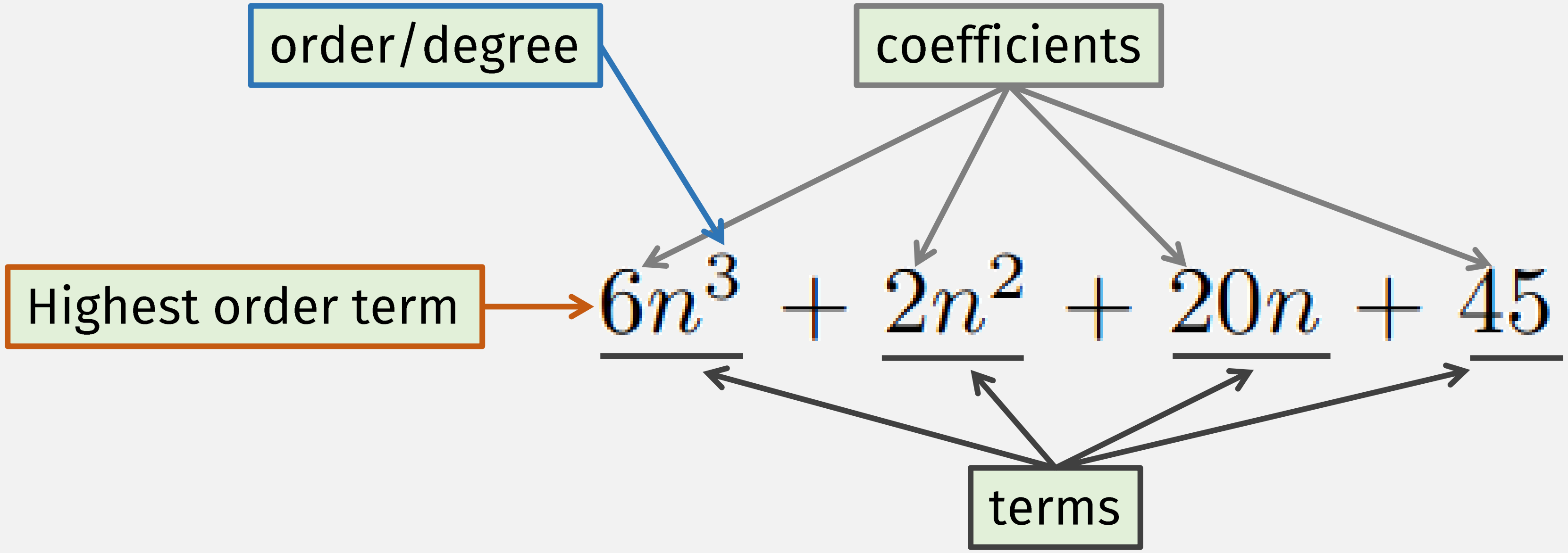
M_1 = "On input string w:

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*."



- Number 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 and 3:
    - Each scan: **n/2** steps to find **1** + **n/2** steps to return = **n** steps
    - Each scan crosses off 2 chars, so at most **n/2** scans
    - Total: **n (n/2)** = $\mathbf{n^2/2}$ steps
  - Line 4:
    - **n** steps to scan input one more time
  - Total: **2n** + $\mathbf{n^2/2}$ **+ n** = $\mathbf{n^2}$ **+3n** steps

Interlude: Polynomials



Definition: Time Complexity

n depends on kind of input

n can be other things, e.g., #states or set size, that are correlated with input length

DEFINITION 7.1

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, 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.

n is only *roughly* "length" of the input

- Machine M_1 that decides $A = \{0^k 1^k | k \geq 0\}$
  - … runs in time $\mathbf{n^2 + 3n}$

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: $\mathbf{n^2 + 3n}$
  - If $\mathbf{n} = 1$
    - $\mathbf{n^2} = 1$
    - $\mathbf{3n} = 3$
    - Total = 4
  - If $\mathbf{n} = 10$
    - $\mathbf{n^2} = 100$
    - $\mathbf{3n} = 30$
    - Total = 130
  - If $\mathbf{n} = 100$
    - $\mathbf{n^2} = 10000$
    - $\mathbf{3n} = 300$
    - Total = 10300
  - If $\mathbf{n} = 1000$
    - $\mathbf{n^2} = 1000000$
    - $\mathbf{3n} = 3000$
    - Total = 1003000
- $\mathbf{n^2 + 3n} \approx \mathbf{n^2}$ as n gets large
- asymptotic analysis only cares about large n

Definition: Big-O Notation

DEFINITION 7.2

Let f and g be functions $f, g: \mathcal{N} \longrightarrow \mathcal{R}^+$. 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.

- In English: Keep only highest order term, drop all coefficients
- Machine M_1 that decides $A = \{0^k 1^k \mid k \geq 0\}$
  - Is an $\mathbf{n^2 + 3n}$ time Turing machine
  - Is an $O(\mathbf{n^2})$ time Turing machine
  - Has asymptotic upper bound $O(\mathbf{n^2})$

Definition: Small-o Notation (less used)

DEFINITION 7.5

Let f and g be functions $f, g: \mathcal{N} \longrightarrow \mathcal{R}^+$. 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 : <

DEFINITION 7.2

Let f and g be functions $f, g: \mathcal{N} \longrightarrow \mathcal{R}^+$. 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

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

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

Definition: Time Complexity Classes

DEFINITION 7.7

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.

- Machine M_1 that decides $A = \{0^k 1^k | \, k \geq 0\}$
  - Is an $O(\mathbf{n^2})$ time Turing machine
  - And A is in $\mathrm{TIME}(\mathbf{n^2})$

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*."

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*."

- Number of steps (worst case), n = length of input:
  - Line 1:
    - **n** steps to scan + **n** steps to return to beginning = $O(\mathbf{n})$ steps
  - Lines 2 and 3:
    - Each scan: $O(\mathbf{n})$ steps
    - Each scan crosses off *half* the chars, so at most $O(\mathbf{log\ n})$ scans
    - Total: $O(\mathbf{n})\ O(\mathbf{log\ n}) = O(\mathbf{n\ log\ n})$ steps
  - Line 4:
    - $O(\mathbf{n})$ steps to scan input one more time
  - Total: $O(\mathbf{n}) + O(\mathbf{n\ log\ n}) + O(\mathbf{n}) = O(\mathbf{n\ log\ n})$ steps

Interlude: Logarithms

- $2^x = y$

- $\log_2 y = x$

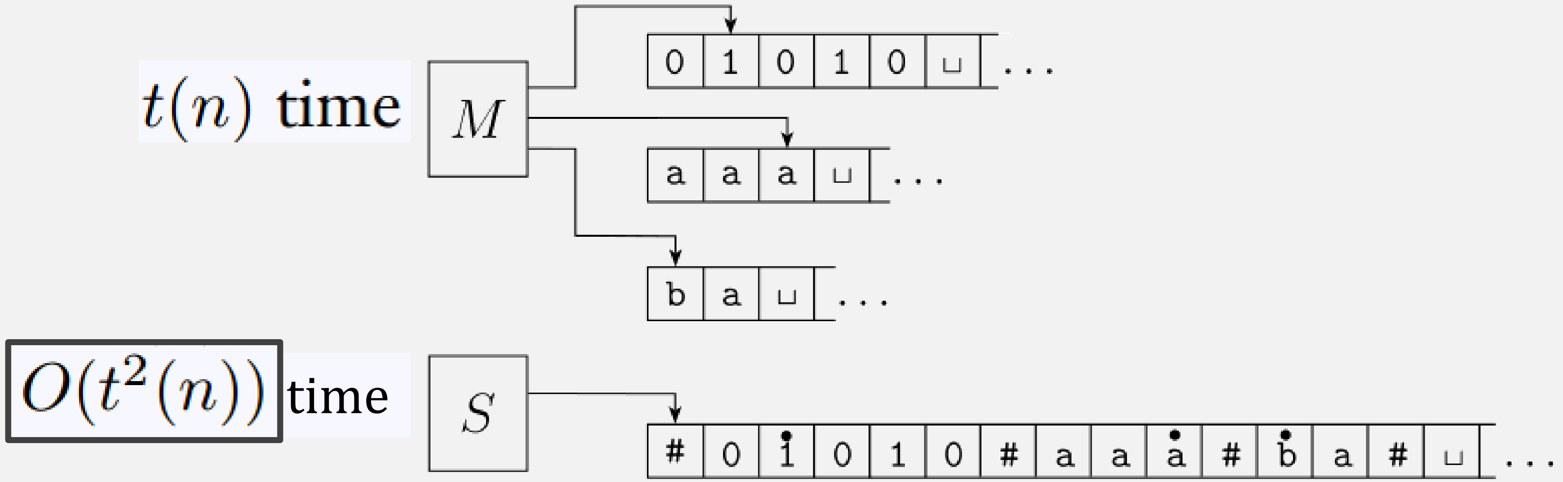
- (In computer science, base-2 is the only base)

- $\log_2 n = O(\mathbf{log\ n})$
  - "divide and conquer" algorithms = $O(\mathbf{log\ n})$
  - E.g., binary search

Terminology: Categories of Bounds

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

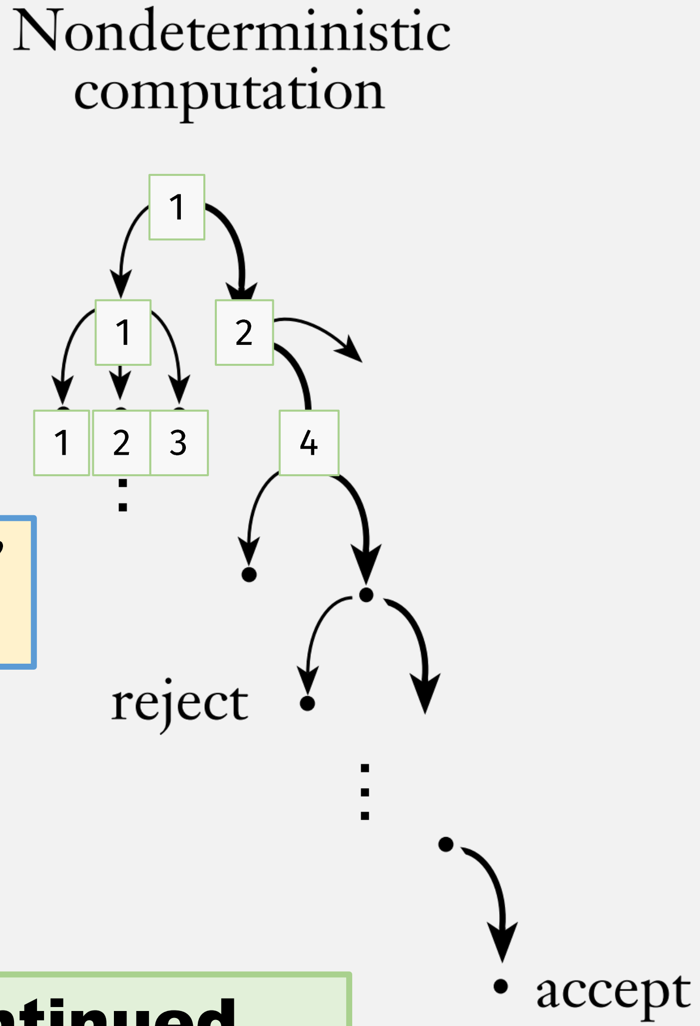
Multi-tape vs Single-tape TMs: # of Steps



- For S to simulate 1 step of M:
  - Scan to find all "heads"
  - "Execute" transition of at all the heads
  - Max single-tape steps to do 1 multitape step = O(length of all M's tapes)
    - = $O(t(n))$ (If M spends all its steps expanding its tapes)
- <u>Total steps (single tape)</u>: $O(t(n))$ per step × $t(n)$ steps = $\boldsymbol{O(t^2(n))}$

Single-tape TM vs Nondet. TM: # of steps

- Deterministic TM simulating nondeterministic TM:
  - Number the nodes at each step
  - Deterministically check every path, in breadth-first order (restart at top each time)
    - 1
    - 1-1
    - 1-2
    - 1-1-1
    - 1-1-2
    - and so on
  - Accept if accepting config found



“This is the most inefficient algorithm ever”
--- CS420 Fall2020 class

But, exactly how inefficient is it???

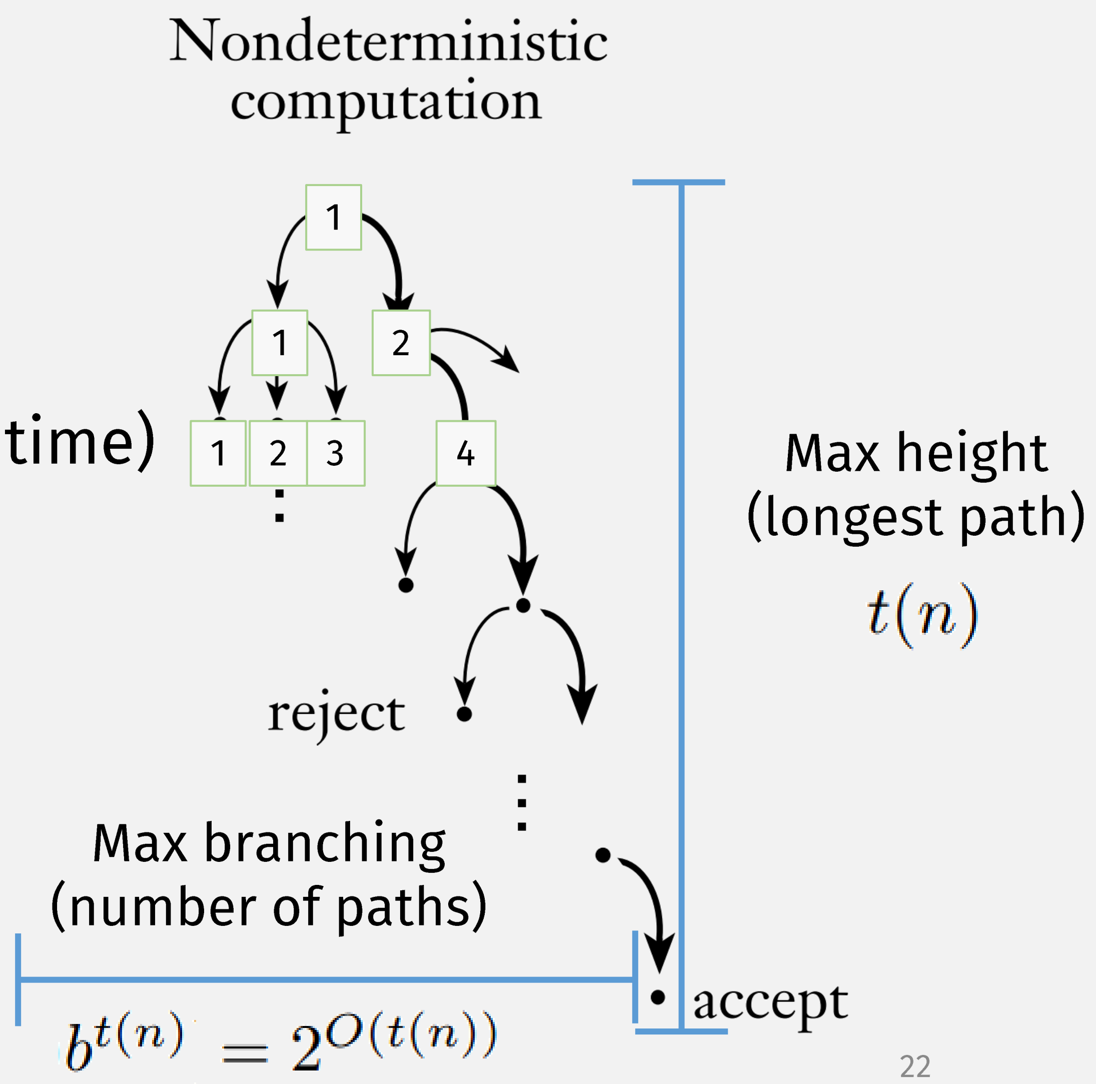
Now we’ll start to count “steps”

To be continued …

Single-tape TM vs Nondet. TM: # of steps

$2^{O(t(n))}$ time

- Deterministic TM simulating nondeterministic TM: $t(n)$ time
  - Number the nodes at each step
  - Deterministically check every path, in breadth-first order (restart at top each time)
    - 1
    - 1-1
    - 1-2
    - 1-1-1
    - 1-1-2
    - and so on
  - Accept if accepting config found



Summary

- 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 11/18

On gradescope

End of Class Survey 11/18

See course website

Polynomial Time

Monday November 23, 2020

Check-in Quiz 11/23

On gradescope

End of Class Survey 11/23

See course website