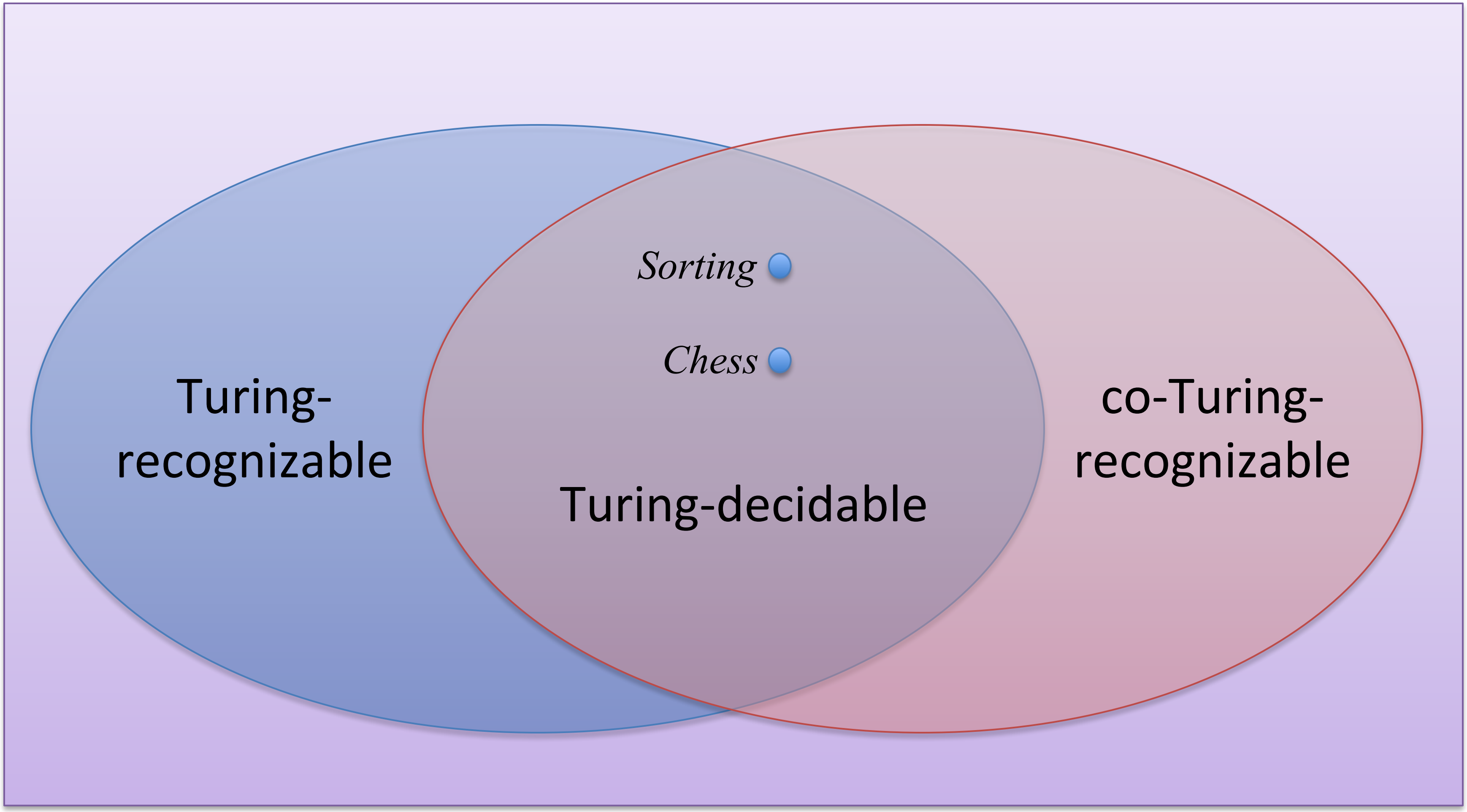
Measuring Time Complexity

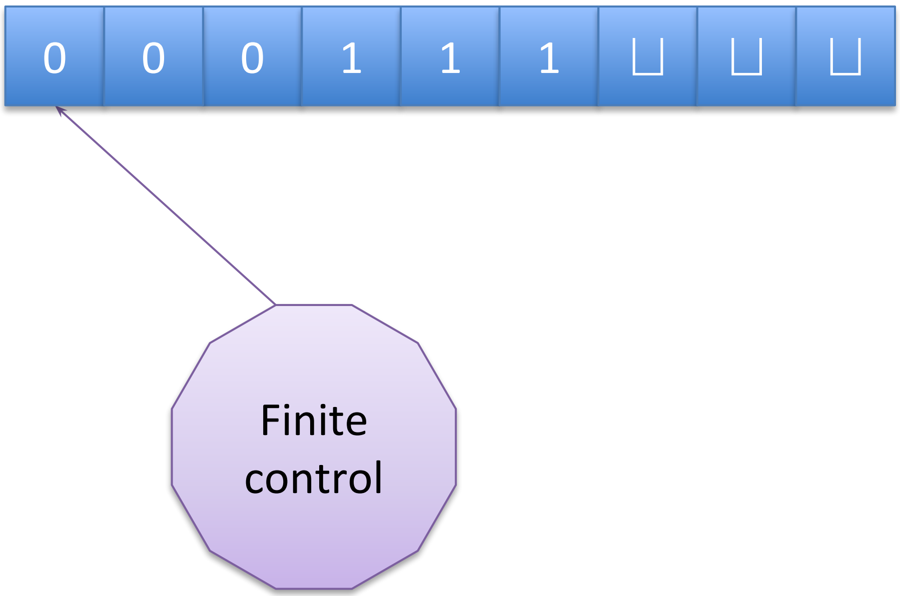
Sipser 7.1 (pages 247-256)

Solvable... in theory



Measuring Computation

- How hard is it to decide $\{0^k 1^k \mid k \geq 0\}$?

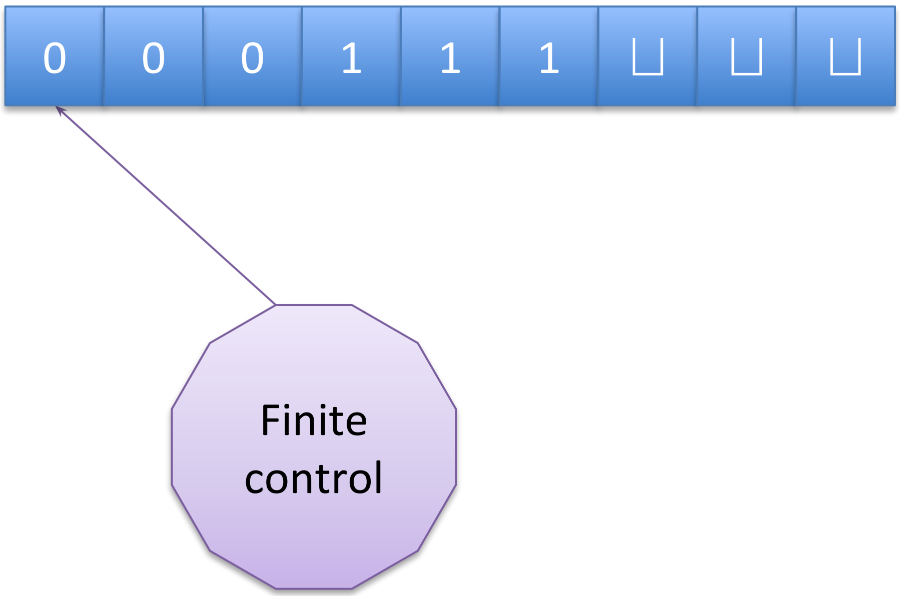


An algorithm

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 the following if both 0s and 1s remain.
3. Scan across tape, crossing off a single 0 and a single 1.
4. If either 0 or 1 remains, *reject*. Else, *accept*."

Number of steps can depend on the size of the input



Which inputs do we consider?

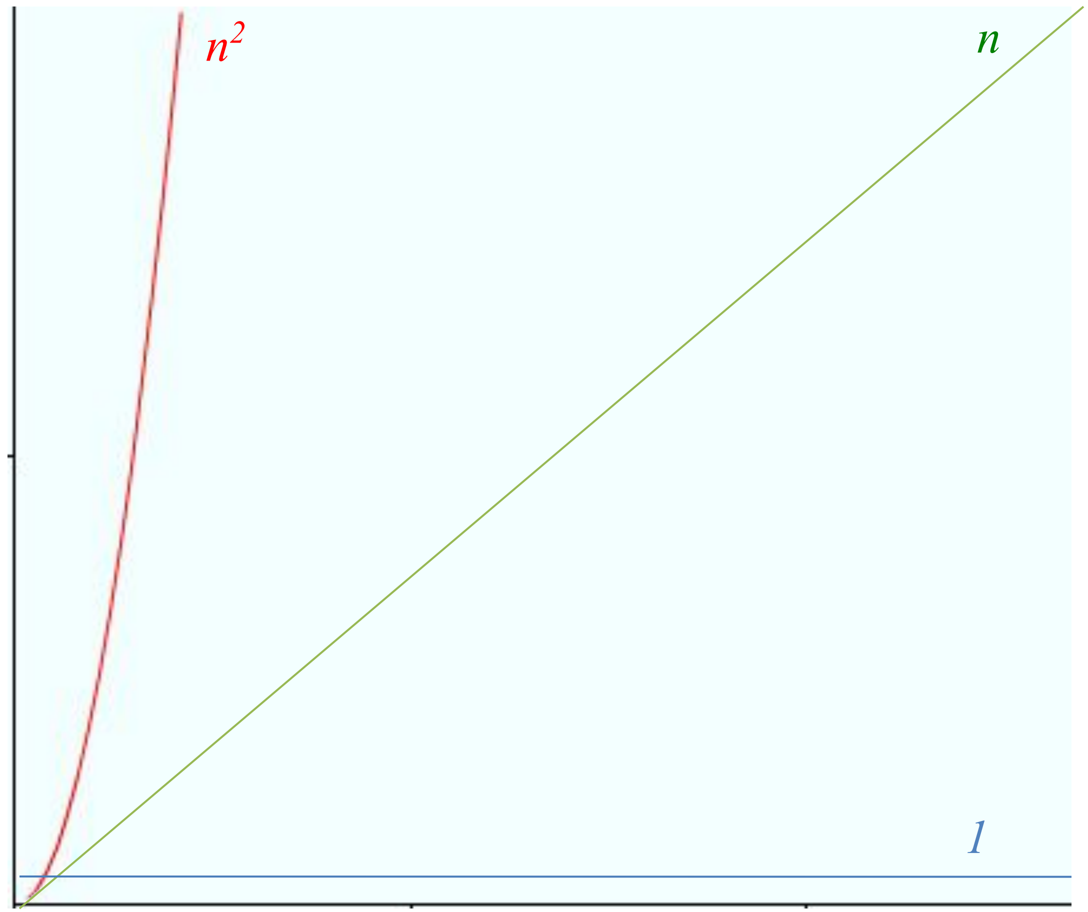
- For a particular input length:
  - *Worst-case analysis*: longest running time of all inputs
  - *Average-case analysis*: average of running times of all inputs

Time complexity

- Definition 7.1

  The **time complexity** of TM M is the function $f: \mathbf{N} \rightarrow \mathbf{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n.

Asymptotic analysis



Big-O and little-o

- Let $f, g: N \rightarrow R^+$.
- Definition 7.2

  We say that $f(n) = O(g(n))$ if positive integers c and n_0 exist so that for every $n \geq n_0$

  $$f(n) \leq c\, g(n)$$
- Definition 7.5

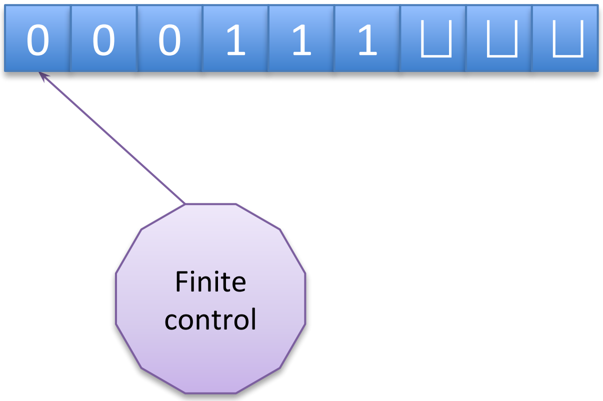
  We say that $f(n) = o(g(n))$ if

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

So now…

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 the following if both 0s and 1s remain.
3. Scan across tape, crossing off a single 0 and a single 1.
4. If either 0 or 1 remains, *reject*. Else, *accept*."



Time Complexity Classes

- Definition 7.7
  Let $t{:}\boldsymbol{N} \rightarrow \boldsymbol{N}$ be a function. Define the **time complexity class, *TIME(t(n))*,** to be the collection of all languages that are decidable by an $O(t(n))$ time TM.
- Example:
  The language $\{0^k 1^k \mid k \geq 0\} \in TIME(n^2)$.*

*And $TIME(n^3)$ and $TIME(n^4)$ and…

Losing time complexity

M_2 ="On input string w:

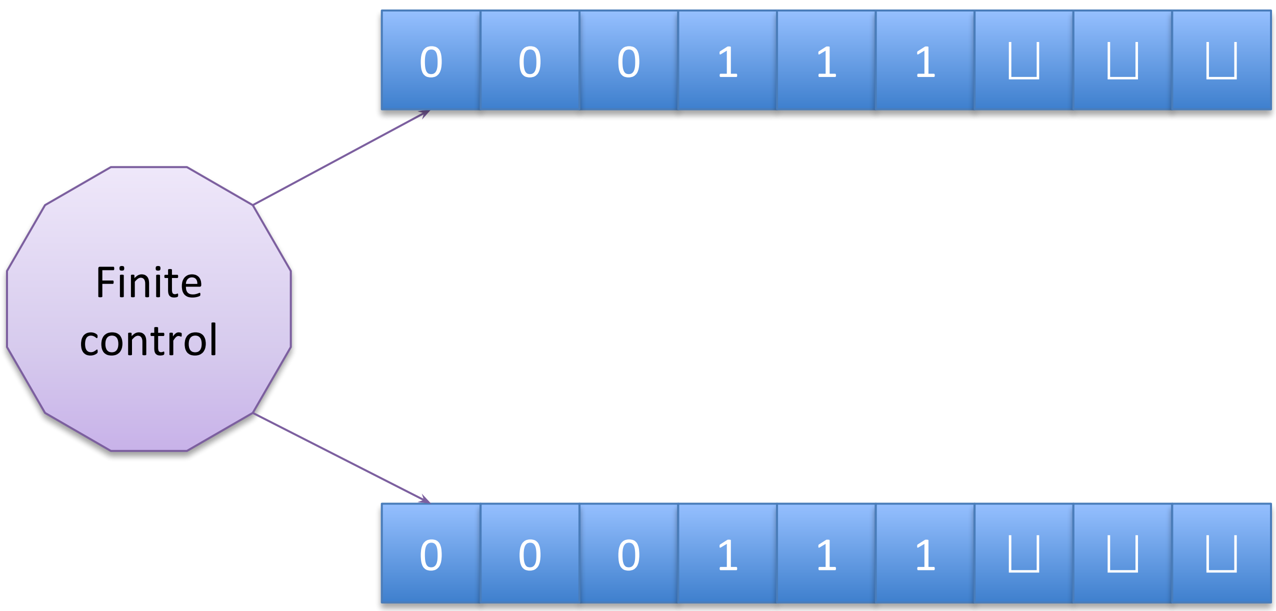
1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat the following if both 0s and 1s remain.
3. Scan across tape, checking whether the total number of 0s and 1s remaining on the tape is even or odd. If odd, *reject*.
4. Scan again across tape, crossing off every other 0, and every other 1.
5. If no 0s or 1s remain, *accept*. Else, *reject*."

Can we do better?

- Theorem: Let $f: \mathbf{N} \rightarrow \mathbf{N}$ be any function where $f(n) = o(n \log n)$.

  $TIME(f(n))$ contains only regular languages.

Well… what if we had two tapes?



A 2-tape algorithm

- M_3 = "On input w:
  1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
  2. Scan across the 0s on tape 1 until the first 1. At the same time, copy the 0s onto tape 2.
  3. Scan across the 1s on tape 1 until the end of the input. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all the 1s are read, *reject*.
  4. If all the 0s have now been cross off, *accept*. If any 0s remain, *reject*.

How to compare?

- Theorem 7.8
  Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

- Proof:
  Compare the time complexity of the given multitape machine with the single tape equivalent given in Theorem 3.13.

Seems more powerful, but…

- Theorem 3.13: Every multitape Turing machine has an equivalent single-tape Turing machine.