Models of Computation

Analysis of Algorithms
Week 1, Lecture 2

Prepared by
John Reif, Ph.D.
Distinguished Professor of Computer Science
Duke University

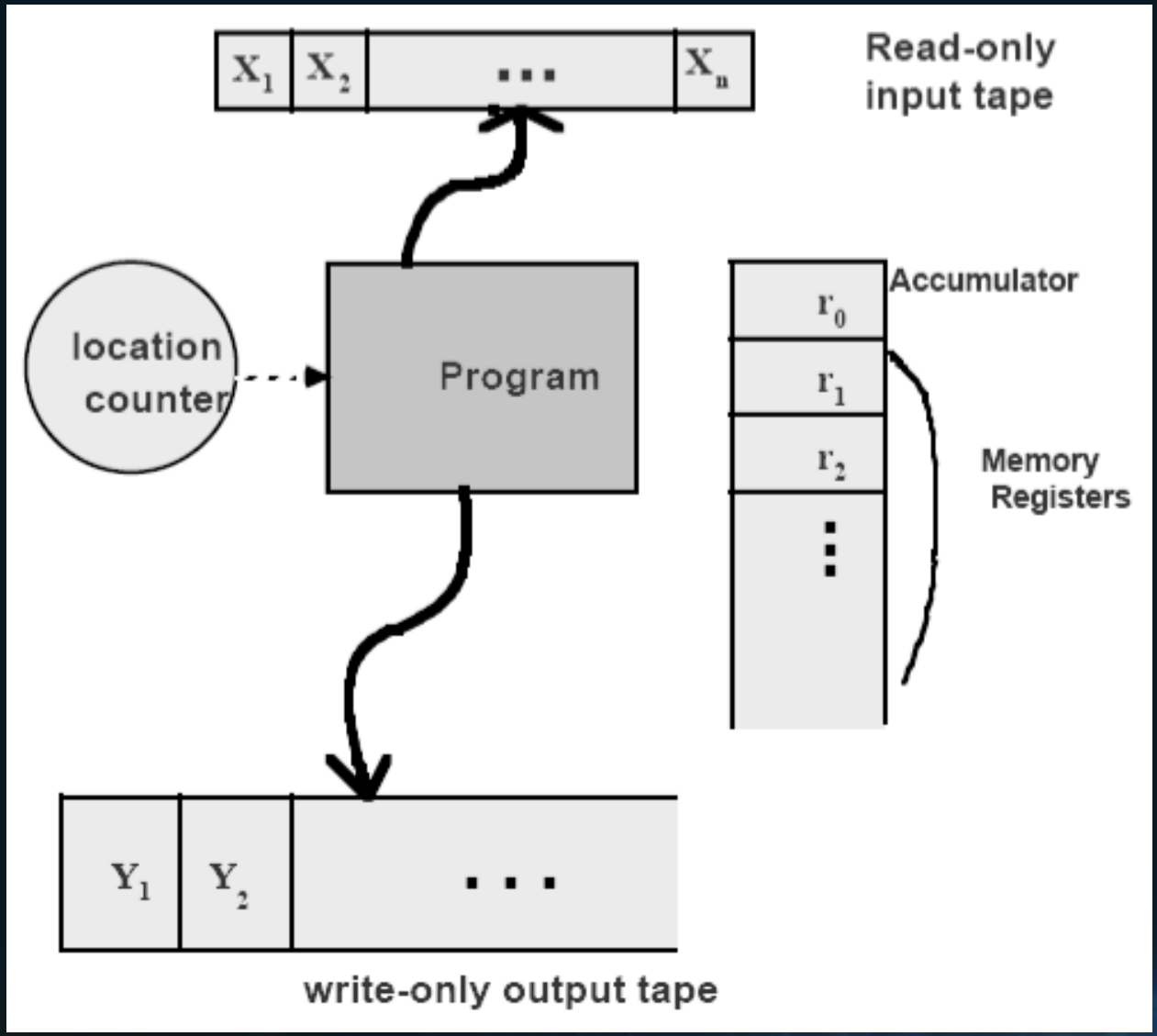
Models of Computation (RAM)

a) Random Access Machines
b) Straight Line Programs and Circuits
c) Vector Machines
d) Turing Machines
e) Pointer Machines
f) Decision Trees
g) Machines That Make Random Choices

Readings

- Main Reading Selection:
  - CLR, Chapter 1 and 2

Random Access Machine (RAM)



RAM Assumptions

1) Each register holds an integer
2) Program can't modify itself
3) Memory instructions involve simple arithmetic
    a) Addition, subtraction
    b) Multiplication, division and control states (got, if-then, etc.)

Simplified Program Style

examples:

- $r \leftarrow constant$
- $r_3 \leftarrow r_1\ op\ r_2$

  $op \in \{+, -, \times, \div\}$
- *goto* label
- *if* $r = 0$ *then goto* L
- *read* (r)
- *write* (r)

Complexity Measures of Algorithms

- $Time_A$ (X) = time cost of Algorithm A, input X
- $Space_A$ (X) = space cost of Algorithm A, input X
- Note: “time” and “space” depend on machine

Complexity Measures of Algorithms (cont' d)

- Worst case time complexity

$$T_A(n) = \max_{\{x:|x|=n\}} (\text{Time}_A(X))$$

- Average case complexity for random inputs

$$E(T_A(n)) = \sum_{\{x:|x|=n\}} \text{Time}_A(X)\text{Prob}(X)$$

- Worst case space complexity

$$S_A(n) = \max_{\{x:|x|=n\}} (\text{Space}_A(X))$$

- Average case complexity for random inputs

$$E(S_A(n)) = \sum_{\{x:|x|=n\}} \text{Space}_A(X)\text{Prob}(X)$$

Cost Criteria

- Uniform Cost Criteria
  - Time = # RAM instructions
  - space = # RAM memory registers
- Logarithmic Cost Criteria
  - Time= L(i) units per RAM instruction on integer size i
  - Space = L(i) units per RAM register on integer size i

Cost Criteria (cont' d)

$$\text{where } L(i) = \begin{cases} \lfloor \log |i| \rfloor & i \neq 0 \\ 1 & i = 0 \end{cases}$$

- Example

$$Z \leftarrow 2$$

$$\textit{for } k=1 \textit{ to } n \textit{ do } Z \leftarrow Z \cdot Z$$

$$\text{ouput } Z = 2^{2^n}$$

Uniform	time cost = n
Logarithmic	time cost > 2^n

Varieties of Computing Machine Models

RAMs
Straight line programs
Circuits
Bit vectors
Lisp machines
...
Turning Machines

Straight Line Programs

- Idea
  - fix n = input size
  - *unroll* each iteration loop until result is loop-free program P_n
  - For each n > 0, get a distinct program P_n

Note: this is only possible if we can *eliminate all branching* and all *indirect addressing*

Example

- Given polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

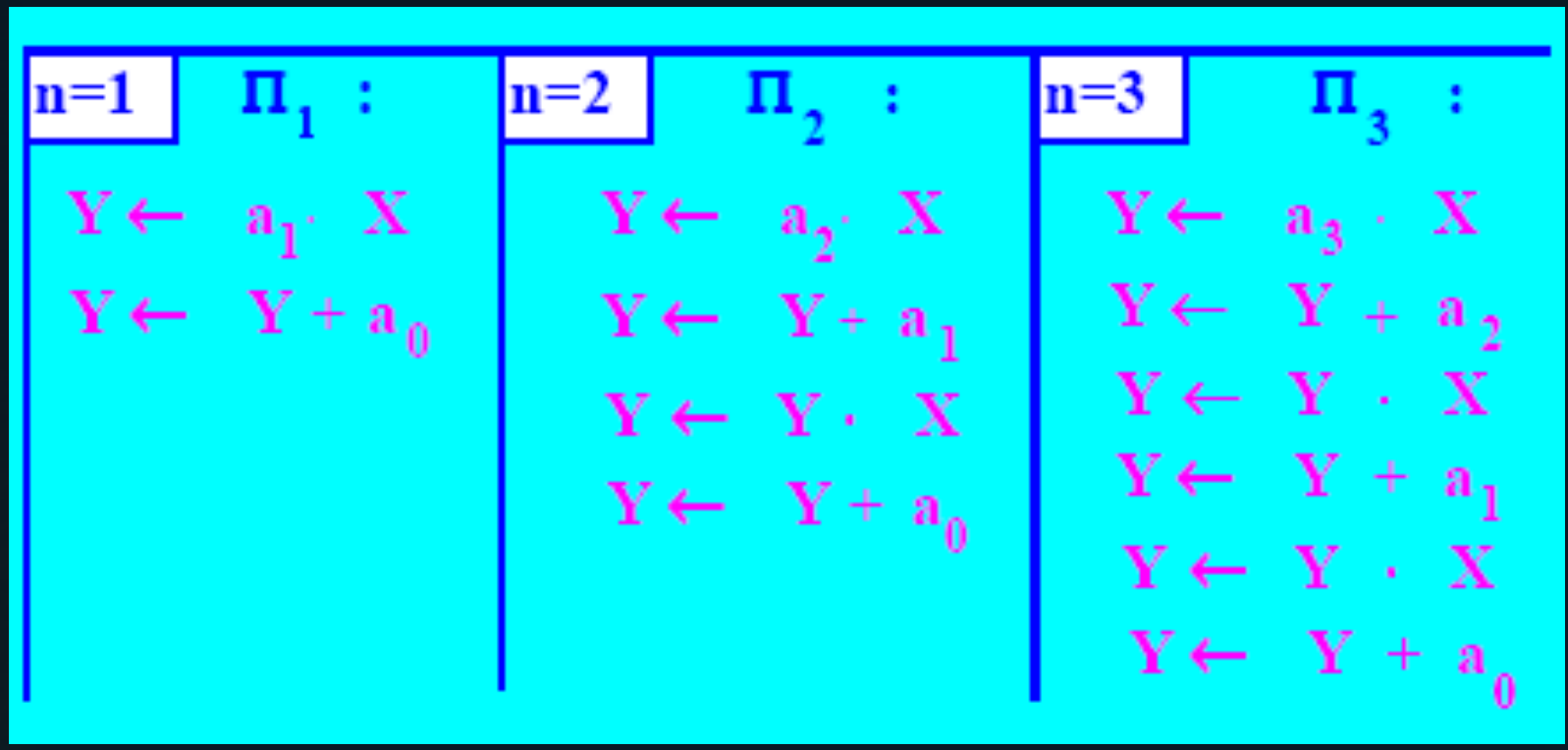
with constant coefficients $a_0, a_1, \ldots, a_n$

- Horner's Rule for Polynomial Evaluation

RAM program in 2n steps

$$\begin{cases} input\ X \\ Y \leftarrow a_n \\ for\ i = n\text{-}1\ by\ -1\ to\ 0\ do\ Y \leftarrow (Y \cdot X) + a_i \\ ouput\ Y \end{cases}$$

Example (cont' d)



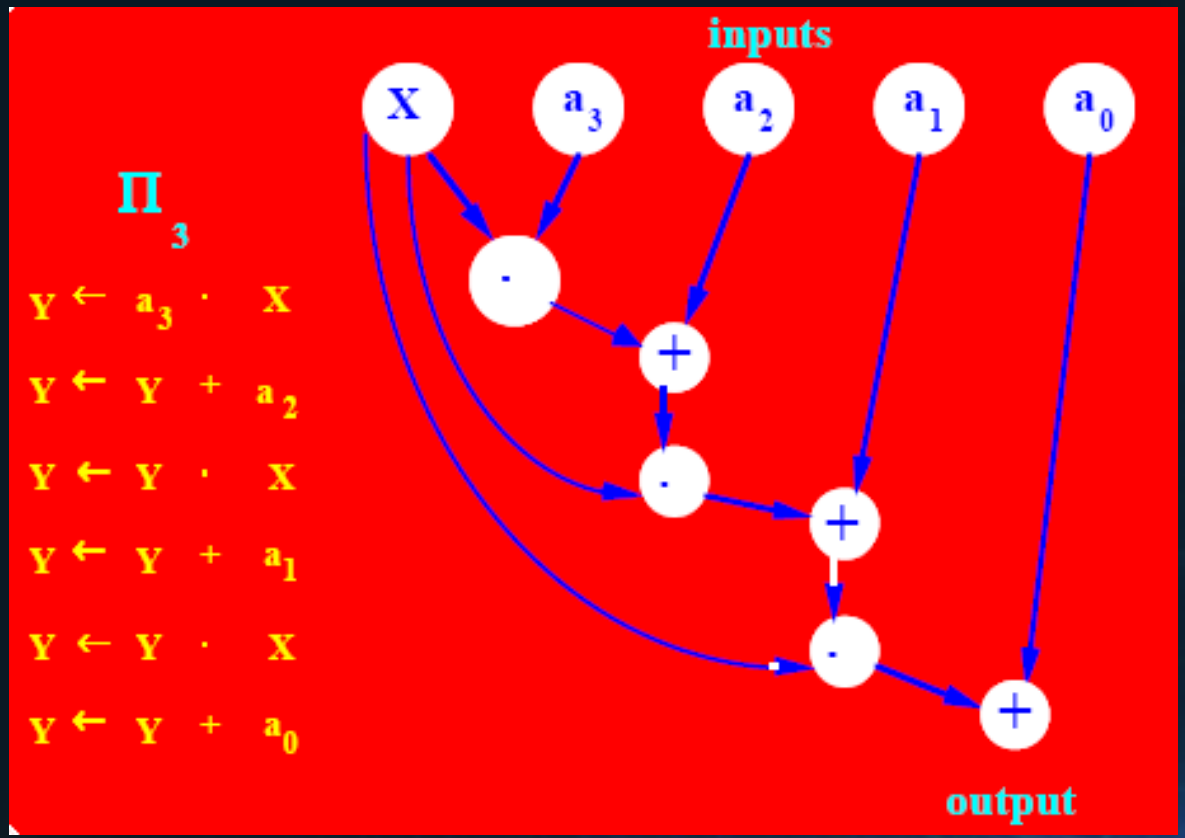
Straight Line Programs

⇔

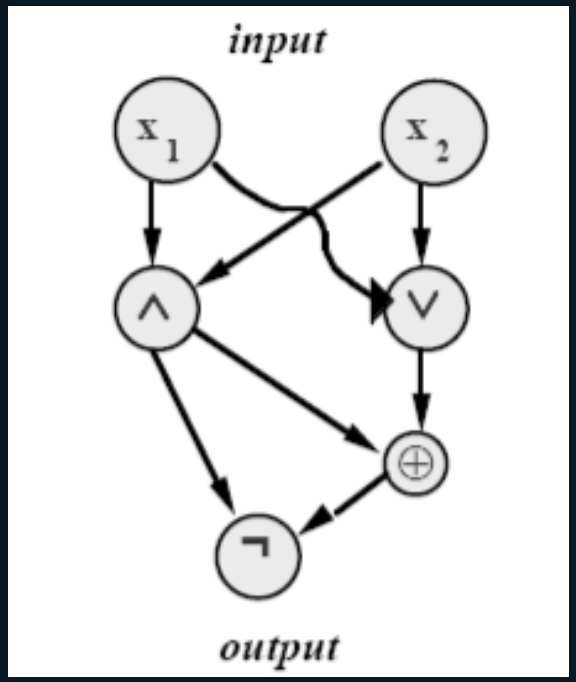
1-1 correspondence

Circuits

(DAG) graph model for straight line programs



Boolean Circuits (for VLSI Design)



- Restrictions
  1) All memory registers have value 0 or 1
  2) Use only logical operations

$\wedge$ $\vee$ $\oplus$ $\neg$

"and" "or" "parity" "not"

Vector Machines

- *logical operations* $\wedge$, $\vee$, $\oplus$, $\neg$ applied to vector elements
- memory locations hold Boolean vectors

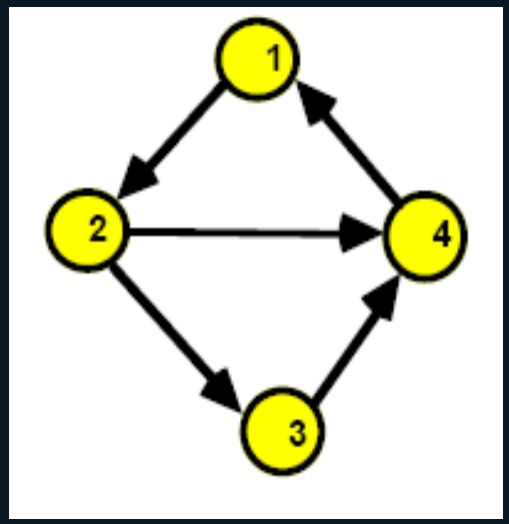
- may also allow vector shift operations

Vector Machines (cont' d)

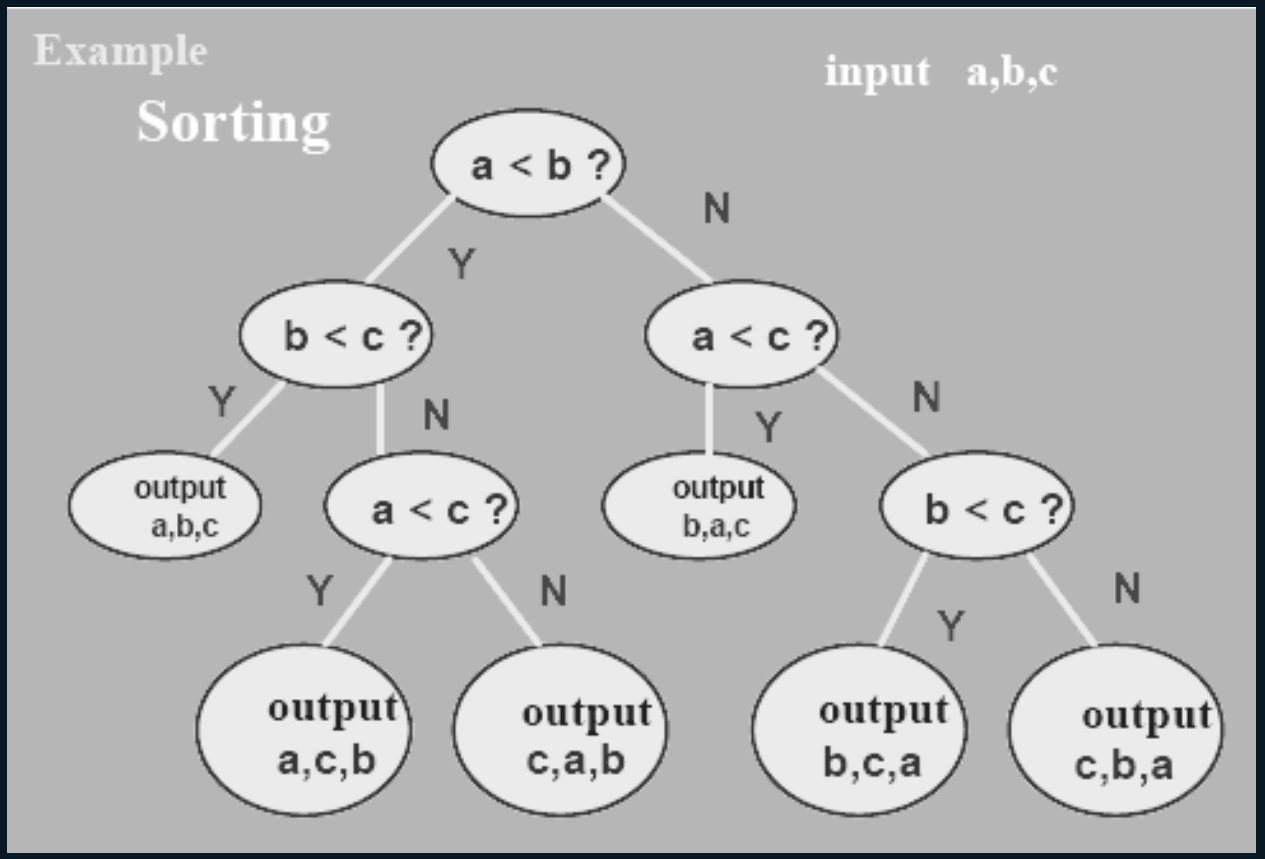
- Example

	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	0	0	0	1
4	1	0	0	0

graph G



Decision Trees



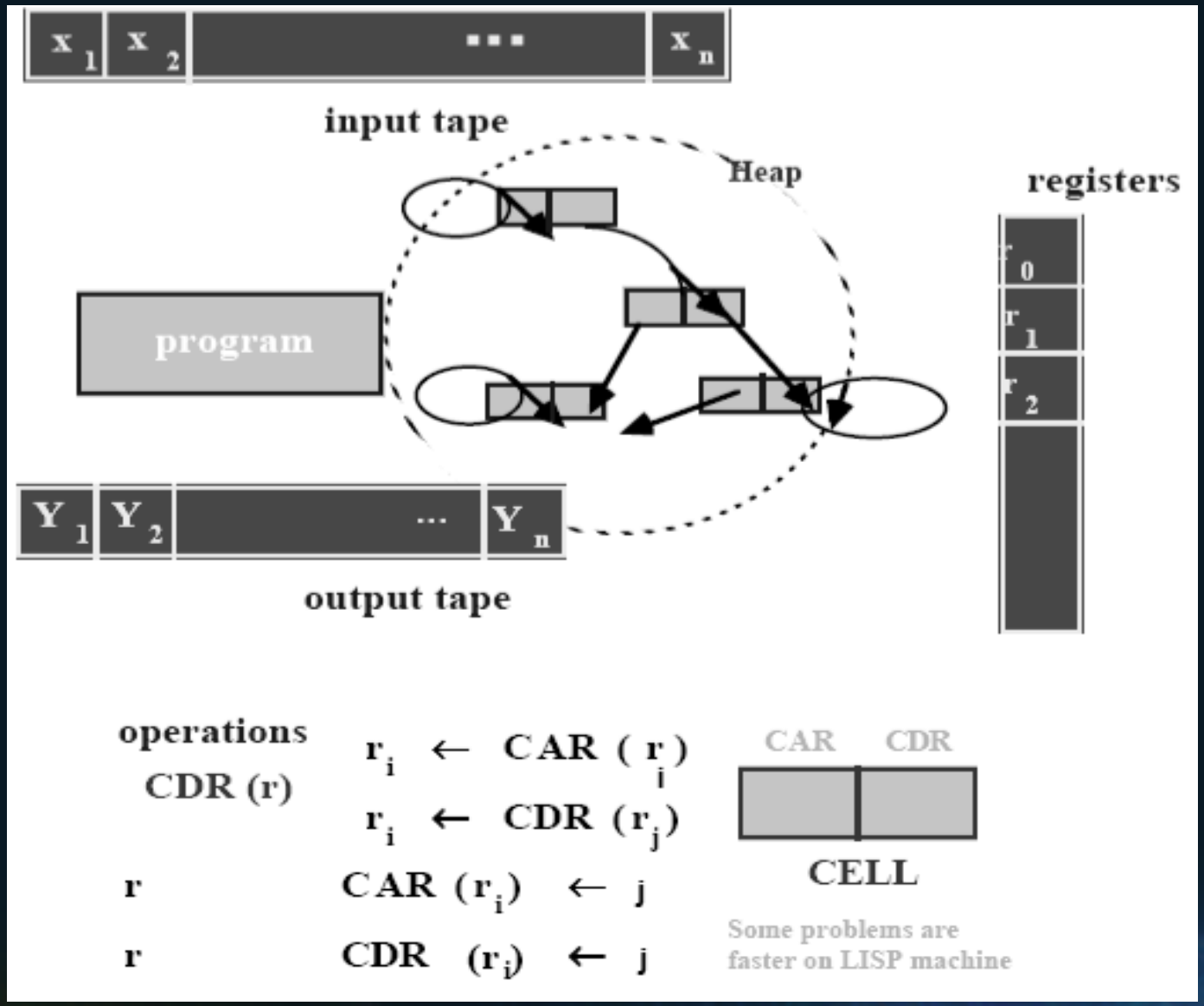
Decision Trees (cont' d)

- To sort n keys
  - Any decision tree must have n! output leaves
  - (n! = # permutations of n keys) hence height of tree is

$$\geq \log_2(n!) \geq c\ n \log n$$

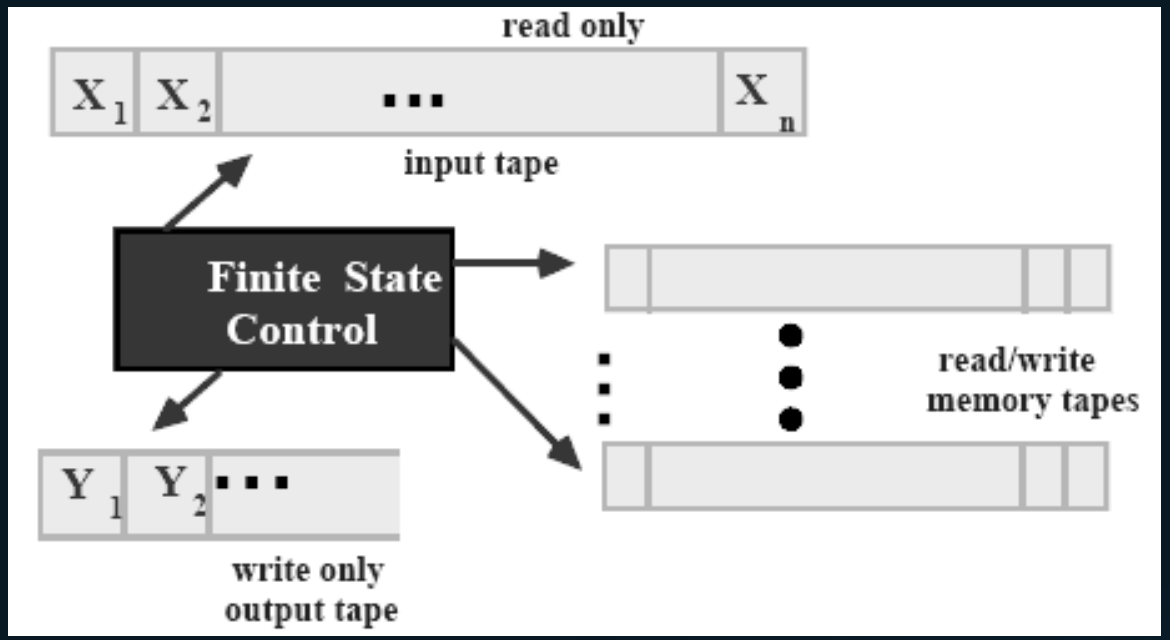
Pointer Machines

Based on LISP programming language



The Turing Machine (TM): The VW of Machines

- Invented by Turing (a Cambridge logician)
  - Built by British for WWII cryptography !



The Turing Machine (cont'd)

T(n) = time cost = max steps of TM

S(n) = space cost = max cells written by TM on memory tapes

Reductions Between TM and RAM Models

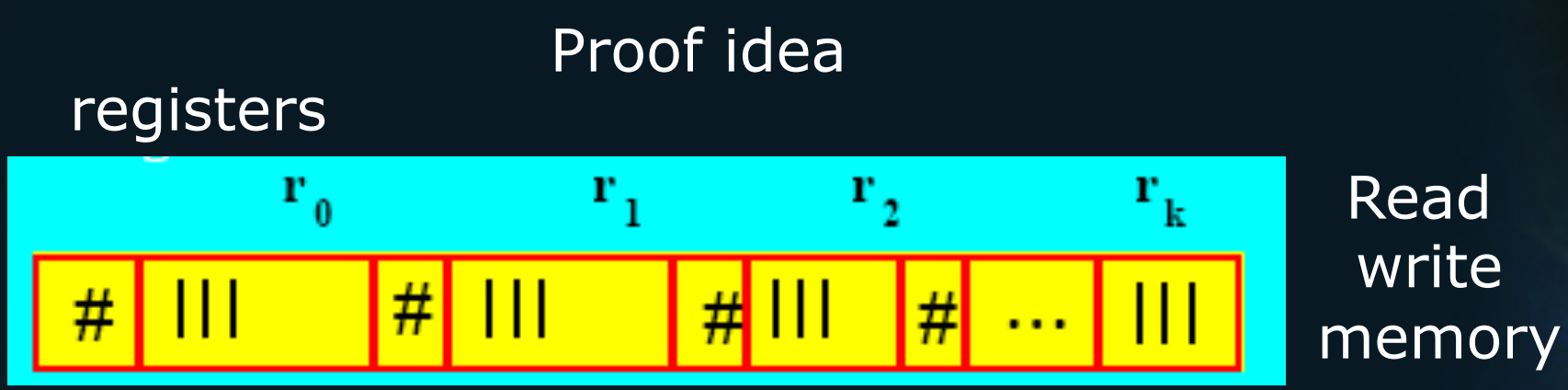
1) Given TM time cost T(n) then
   ∃ equivalent RAM (obvious)

   Time cost { c T(n) if uniform cost
   { c T(n) (log n) if log cost

2) Given RAM time cost T(n) with logarithmic cost then
   ∃ equivalent TM with time cost c' $T(n)^2$

Reductions Between TM and RAM Models (cont' d)

Proof idea



- Do arithmetic by Grammar School Method

Extensions of RAMS

- Reasonable:
  - (0) Modifiable Program
  - (1) Random Choices
  - (2) Non-uniformity
- Not Reasonable:
  - (3) Non-deterministic Choices

RASP Machine

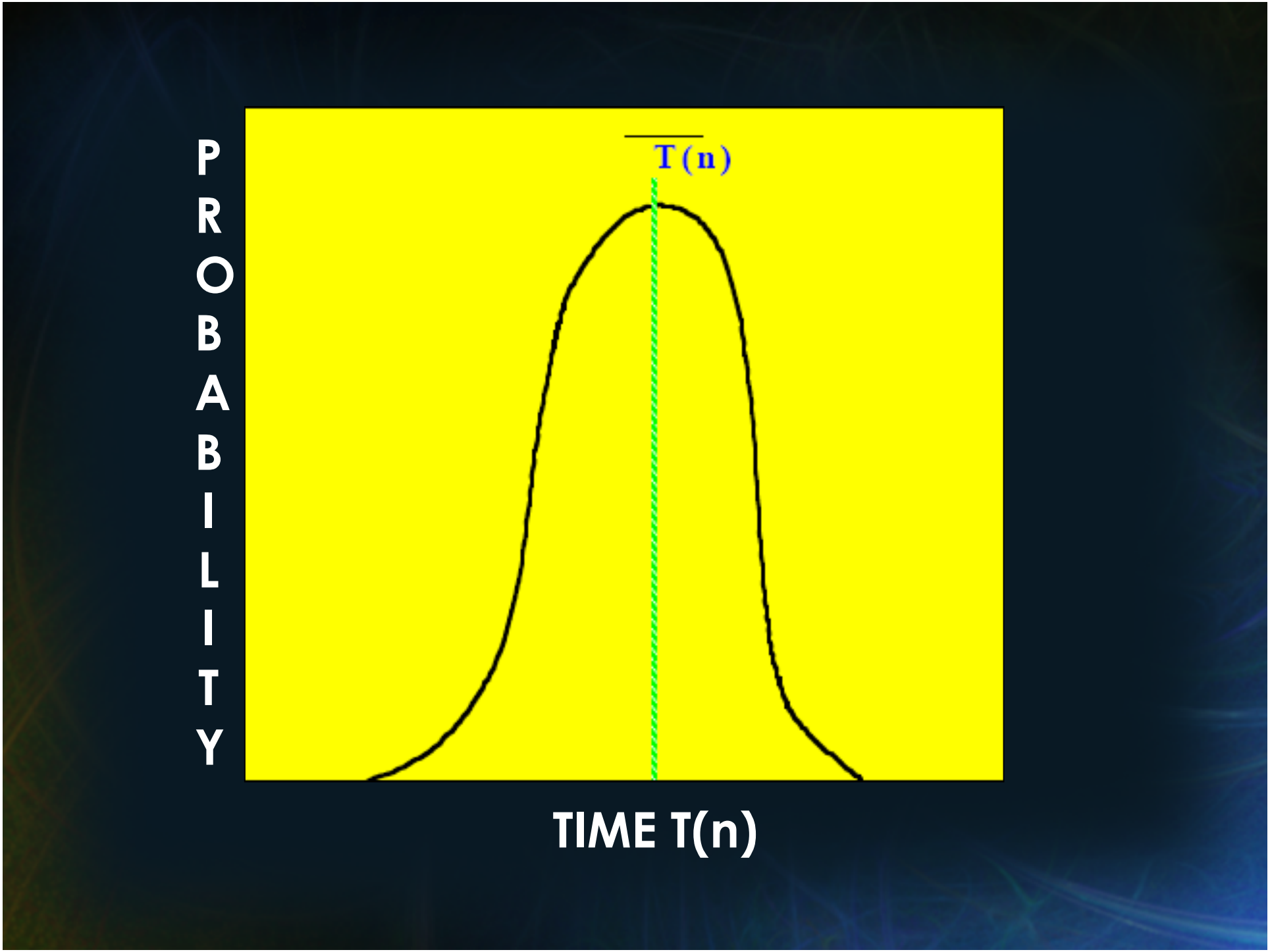
- Same as RAM but allow
  program to change itself
- Same power as RAM
- Proof idea (due to Von Neumann)
  - Let RAM use memory registers to store modifiable program of RASP

Randomized Machines

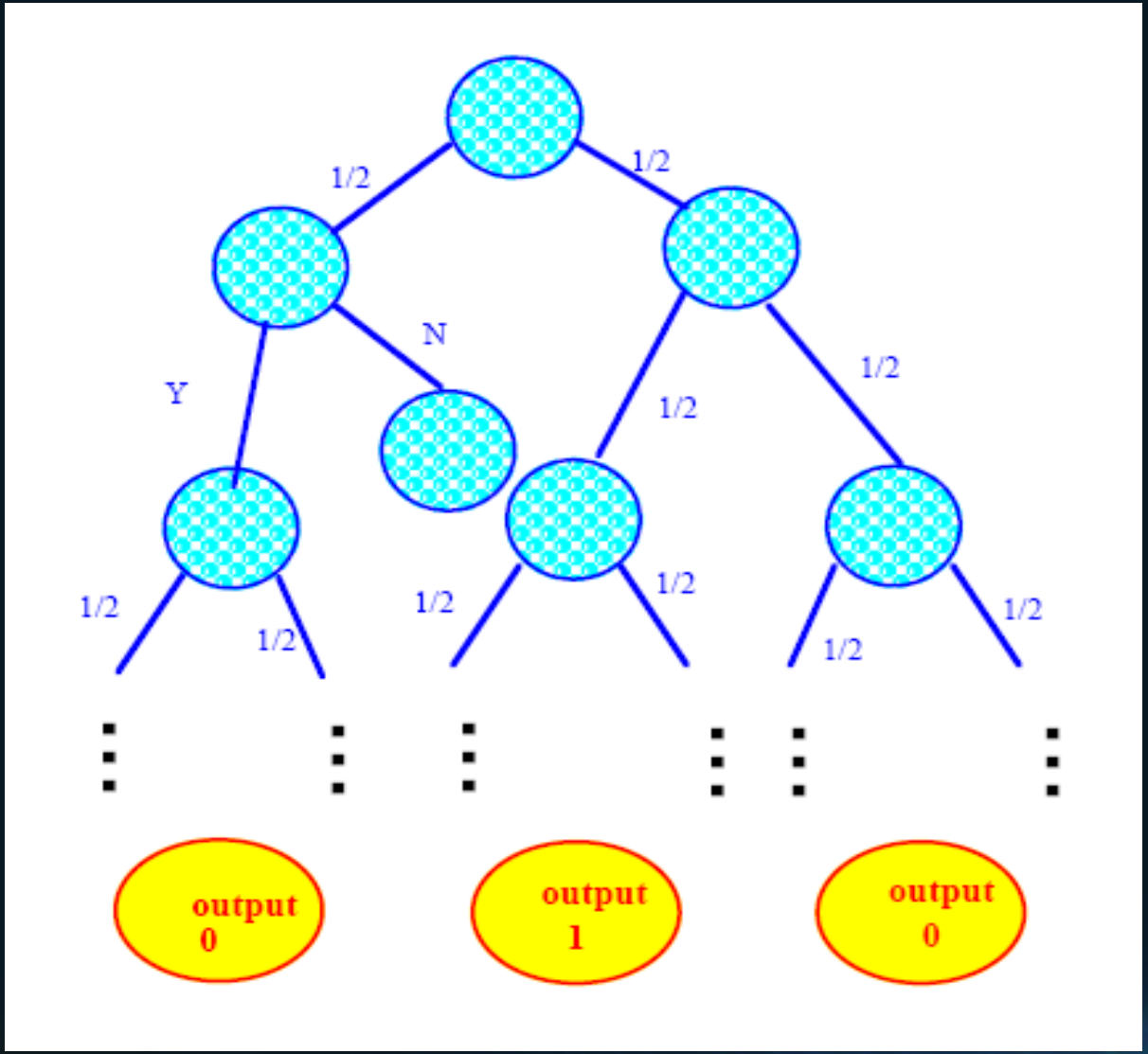
- Extend RAM instructions to include
  - r ← RANDOM(k) gives a random k bit number
  - Let $A_R(x)$ denote randomized algorithm with input x, random choices R

$$\frac{\text{Expected Time}}{\text{Time (X)}} = \sum_{\forall R}^{\text{input X}} \text{Time (X) Prob (R)}$$

- Expected Time Complexity
  - T(n) = max Time (x)
    {x|n = |x|}


PROBABILITY
$\overline{T(n)}$
TIME T(n)

A Randomized Computation



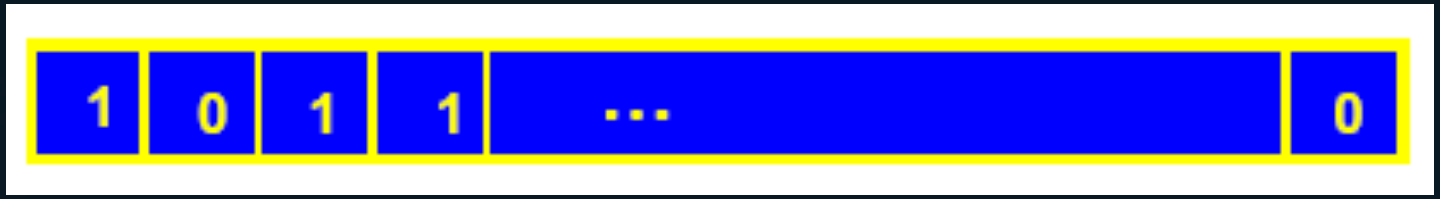
1) If machine outputs value v with prob > ½ then v is considered its output.

2) Machine accepts input X if outputs 1 with prob > ½

3) Has 1-sided error if when not accepting 1 outputs only 0.

Non-Uniformity

- For each input size n, allow the program a distinct, finite, "advice tape" to read



- Note:
  - If advice tape length 2^n can solve any Boolean output problem with n Boolean inputs (obvious).

Surprising Result (Adelman)

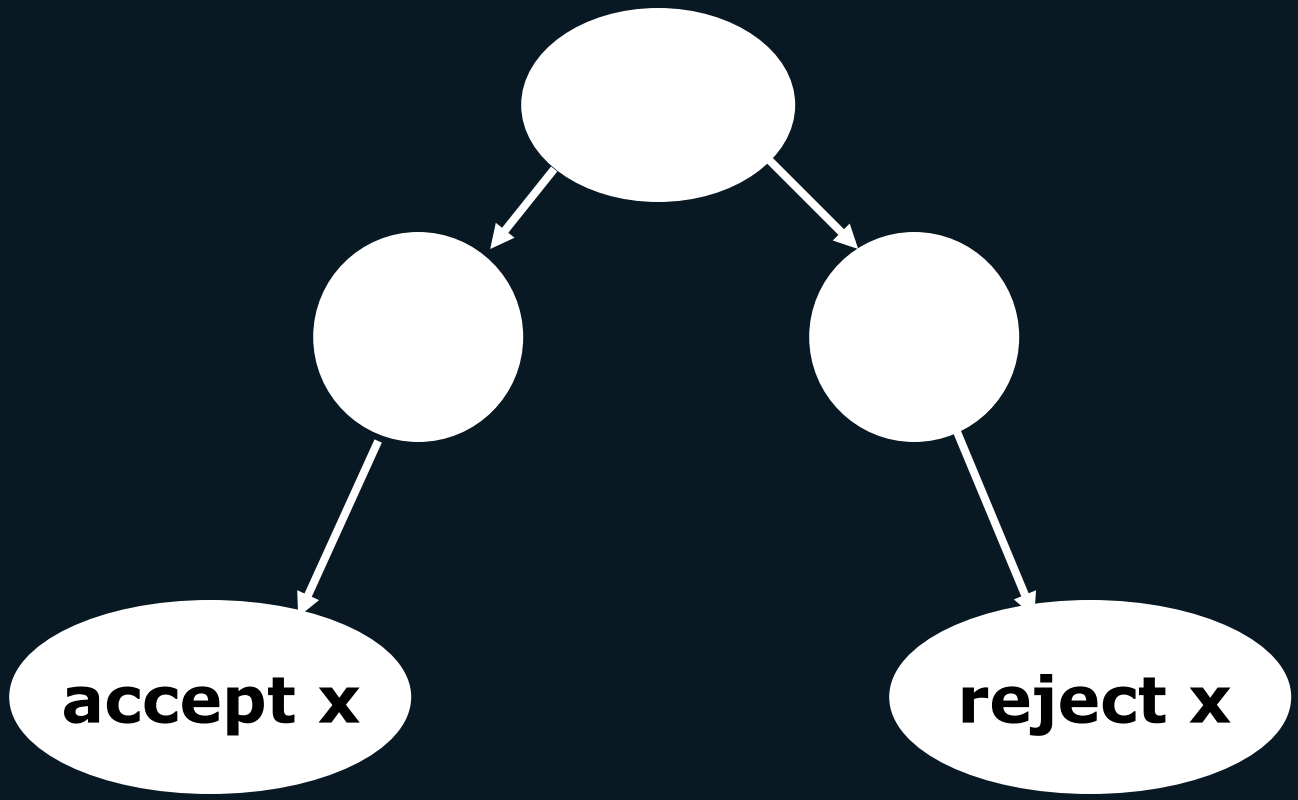
- Given any polynomial time randomized algorithm with 1 side error,

  $\exists$ a non-uniform deterministic algorithm with polynomial time and polynomial advice!

  - Gives way of derandomizing a randomized algorithm

Nondeterministic Machines

- Allow "nondeterministic choice" branches



- If any sequence of choices succeed to accept x, then computation accepts.

NP and P

- NP = languages accepted by polynomial time nondeterministic TM machines.
  - Includes many hard problems:
    1) Traveling Salesman Problem
    2) Propositional Satisfiability
    3) Integer Programming
- P = languages accepted by polynomial time deterministic machines

$$P \stackrel{?}{=} NP$$

Not known probably no

Another Surprising Result (Levin)

- If P=NP but we don't know the proof (i.e., the polynomial time algorithm for NP)
  - find an optimal algorithm to find the solution of any solvable search problem, in polynomial time!

- Proof depends on assumption that there is a finite length program for NP search problems, running in poly time)

Conclusion

1) There are many possible machine models

2) Most (but not Nondeterministic) are "constructable" – so might be used if we have efficient algorithms to execute on machines.

3) New machine models can help us invent new algorithms, and vice versa!

Models of Computation

Analysis of Algorithms
Week 1, Lecture 2

Prepared by
John Reif, Ph.D.
Distinguished Professor of Computer Science
Duke University