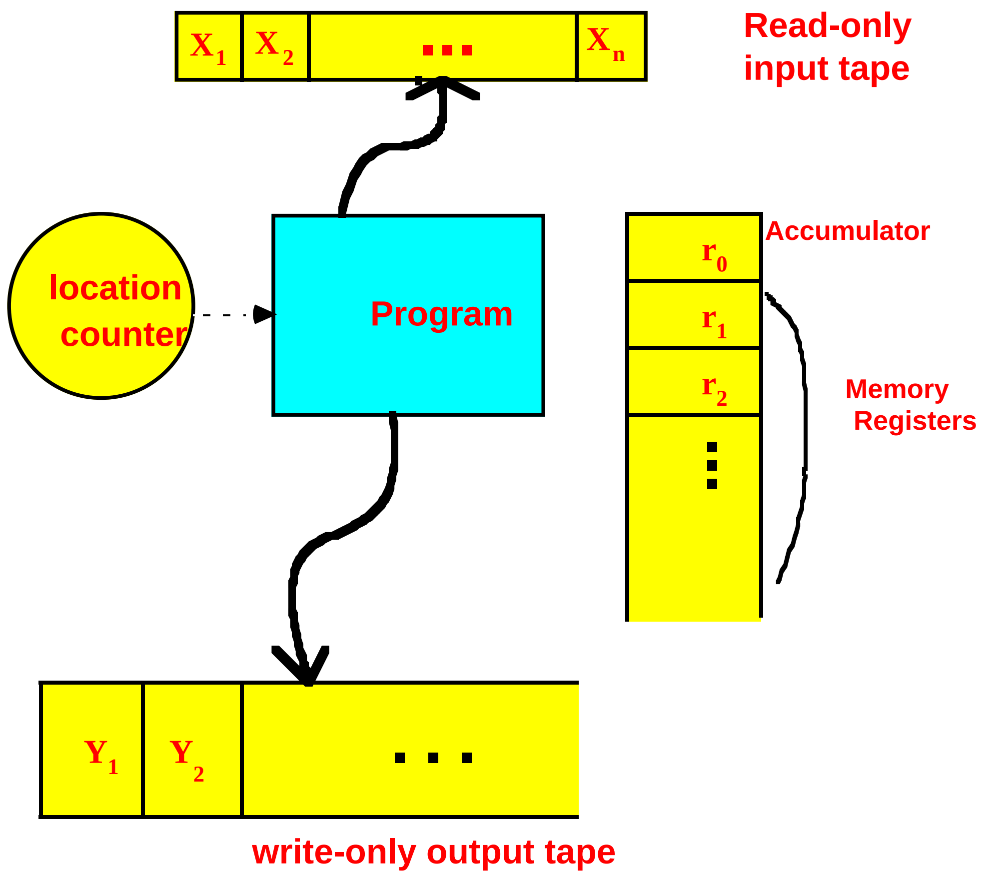
Algorithms
Professor John Reif

ALG 1.1

Models of Computation:

(a) Random Access Machines (RAMs)
(b) Straight Line Programs and Circuits
(c) Decision Trees
(d) Machines That Make Random Choices

Main Reading Selection:
CLR, Chapter 1
Auxillary Reading Selections:
AHU-Design, Chapter 1
BB, Sections 1.1-1.5, 1.8
AHU-Data, Chapter 1



RANDOM ACCESS MACHINE

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 statements (goto, if-then, etc.)

Written in "Pidgin Algol"

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)

input X
size n = |X| $\Rightarrow$ Algorithm A $\Rightarrow$ output

Complexity Measures of Algorithms

$\text{Time}_A(X)$ = time cost of Algorithm A, input X

$\text{Space}_A(X)$ = space " " "

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

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

Note: "time" and "space" depend on machine

UNIFORM COST CRITERIA

$$\begin{cases} \text{time} = \#\text{RAM instructions} \\ \text{space} = \#\text{RAM memory registers} \end{cases}$$

LOGORITHMIC COST CRITERIA

$$\begin{cases} \text{time} = L(i) \text{ units per RAM instruction on integer size } i \\ \text{space} = L(i) \text{ units per RAM register size } i \end{cases}$$

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

example

$Z \leftarrow 2$

for $k = 1$ *to* n *do* $Z \leftarrow Z \cdot Z$

output $Z = 2^{2^n}$

$$\begin{cases} \textit{uniform} & \text{time cost} = n \\ \textit{logarithmic} & \text{time cost} > 2^n \end{cases}$$

Varieties of Computing Machine Models

$$\begin{cases} \text{RAMs} \\ \text{straight line programs} \\ \text{circuits} \\ \text{bit vectors} \\ \text{lisp machines} \end{cases}$$

⋮

Turing Machines

Straight Line Programs

idea fix $n =$ input size

unroll each iteration loop
until result is
loop-free program Π_n

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

$\Rightarrow$ for each $n > 0$,
get a distinct program Π_n

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

input X
$Y \leftarrow a_n$
for $i = n-1$ *by* -1 *to* 0 *do*
$Y \leftarrow (Y \cdot X) + a_i$
output Y

n=1 Π_1 :	n=2 Π_2 :	n=3 Π_3 :
$Y \leftarrow a_1 \cdot X$	$Y \leftarrow a_2 \cdot X$	$Y \leftarrow a_3 \cdot X$
$Y \leftarrow Y + a_0$	$Y \leftarrow Y + a_1$	$Y \leftarrow Y + a_2$
	$Y \leftarrow Y \cdot X$	$Y \leftarrow Y \cdot X$
	$Y \leftarrow Y + a_0$	$Y \leftarrow Y + a_1$
		$Y \leftarrow Y \cdot X$
		$Y \leftarrow Y + a_0$

Straight Line Programs $\Longleftrightarrow$ *Circuits*

1-1 correspondence

(DAG) graph model for straight line programs

Π_3

$Y \leftarrow a_3 \cdot X$

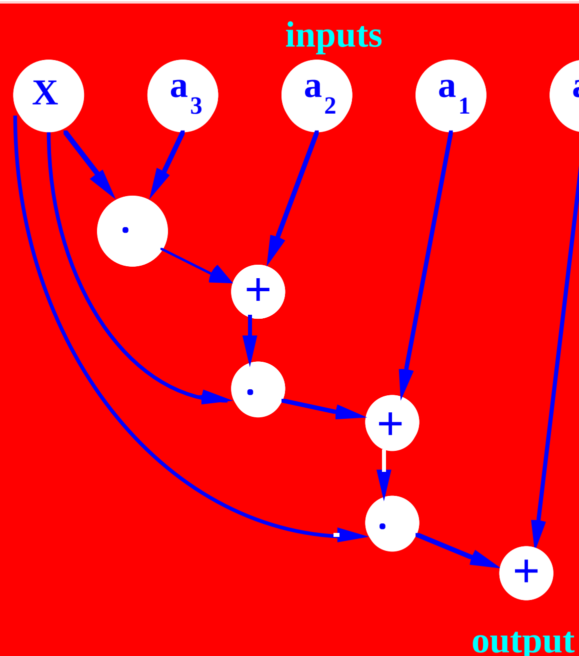
$Y \leftarrow Y + a_2$

$Y \leftarrow Y \cdot X$

$Y \leftarrow Y + a_1$

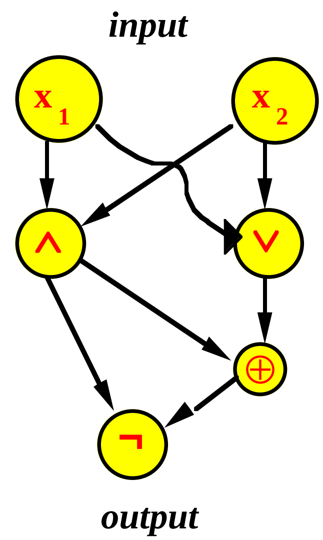
$Y \leftarrow Y \cdot X$

$Y \leftarrow Y + a_0$



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

(modeling CONVEX computer)

logical operations $\vee, \wedge, \oplus, \neg$

applied to *vector* elements

memory locations hold *boolean vectors*

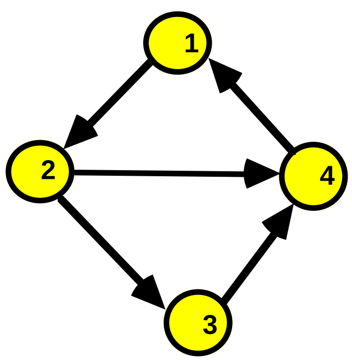
0	1	0	1	...	0

may also allow vector shift operations

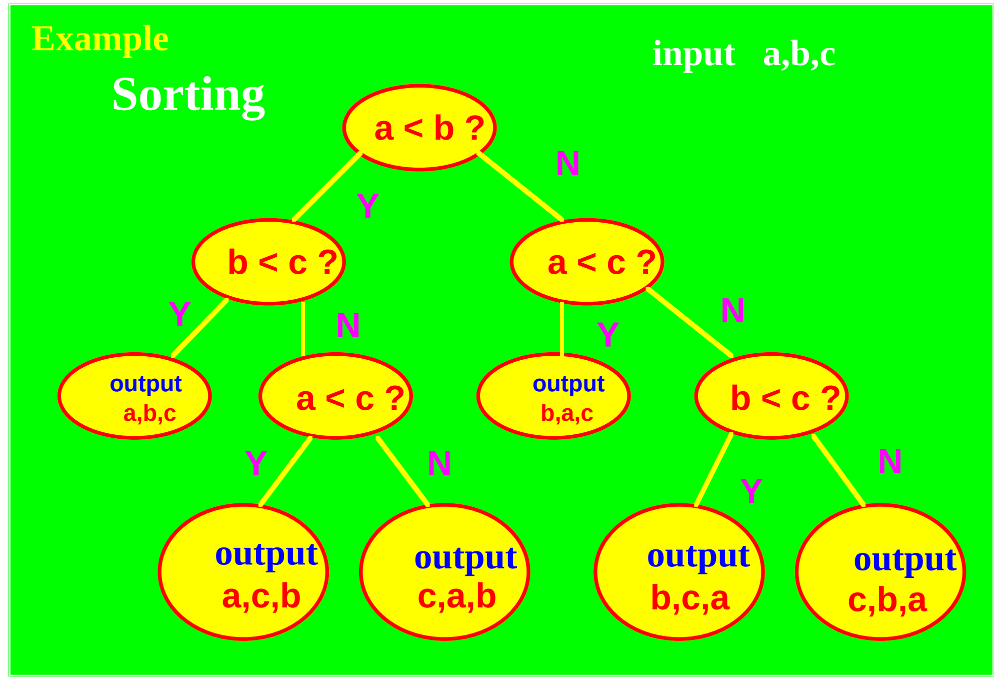
Example

	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



To sort *n keys*

any decision tree must have

n! output leaves

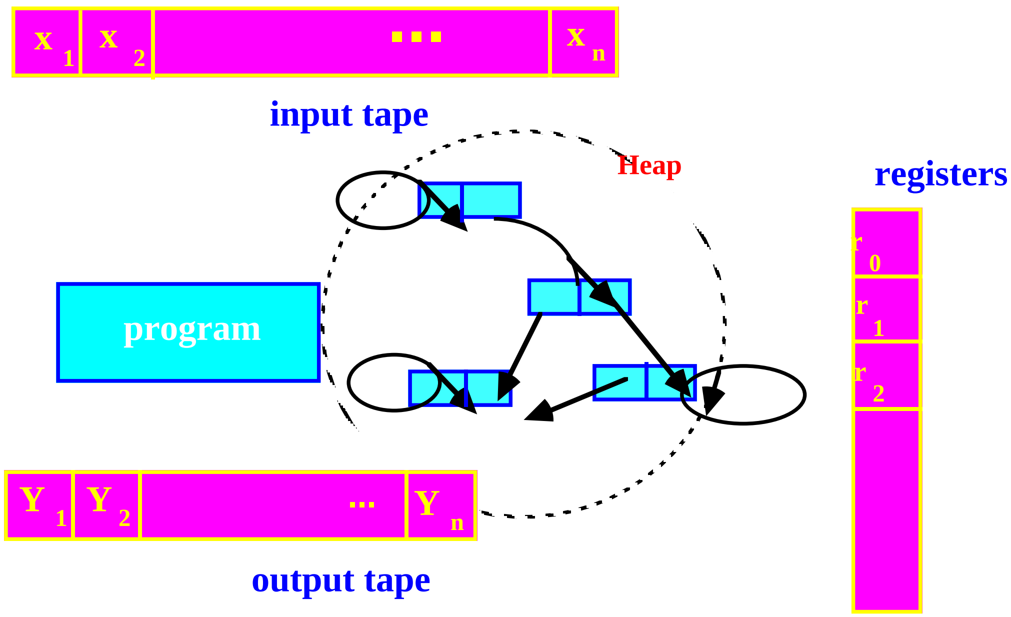
(n! = # permutations of n keys)

hence height of tree is

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

Lisp Machines

Complexity theory for AI



operations

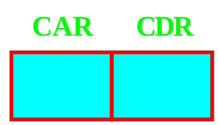
CDR (r)

$r_i \leftarrow \text{CAR}(r_i)$

$r_i \leftarrow \text{CDR}(r_j)$

r $\quad \text{CAR}(r_i) \leftarrow j$

r $\quad \text{CDR}(r_i) \leftarrow j$

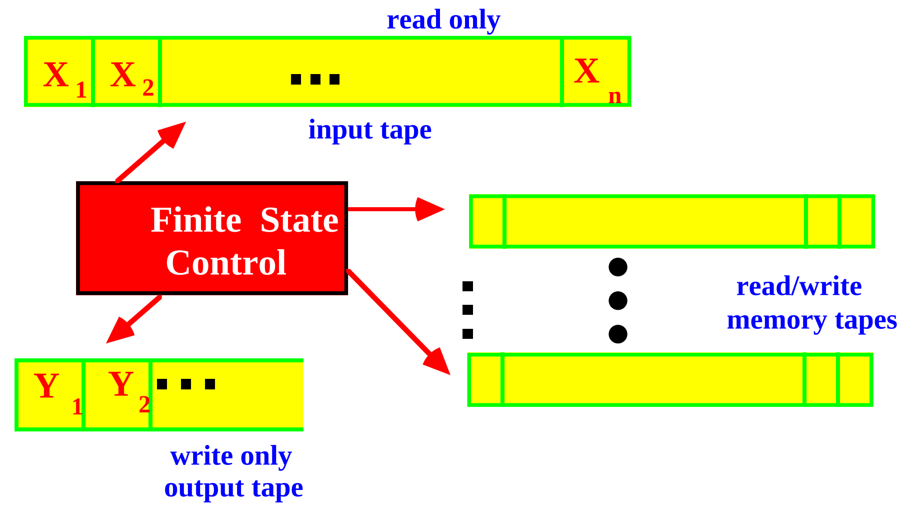


CELL

Some problems are faster on LISP machine

"The VW of Machines"

"The Ultimate Program Language of Theory"
Turing Machine (TM)



Invented by Turing (a Cambridge logician)

Built by British for WWII crytography!

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 logarithmic cost

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

proof idea

registers

r_0		r_1		r_2			r_k
#	|||	#	|||	# |||	#	...	|||

read write memory

do arithmetic by Grammer 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

Proof idea

Let RAM use memory registers to store modifiable program of RASP

(due to Von Neumann)

Randomized Machines

Extend RAM instructions to include

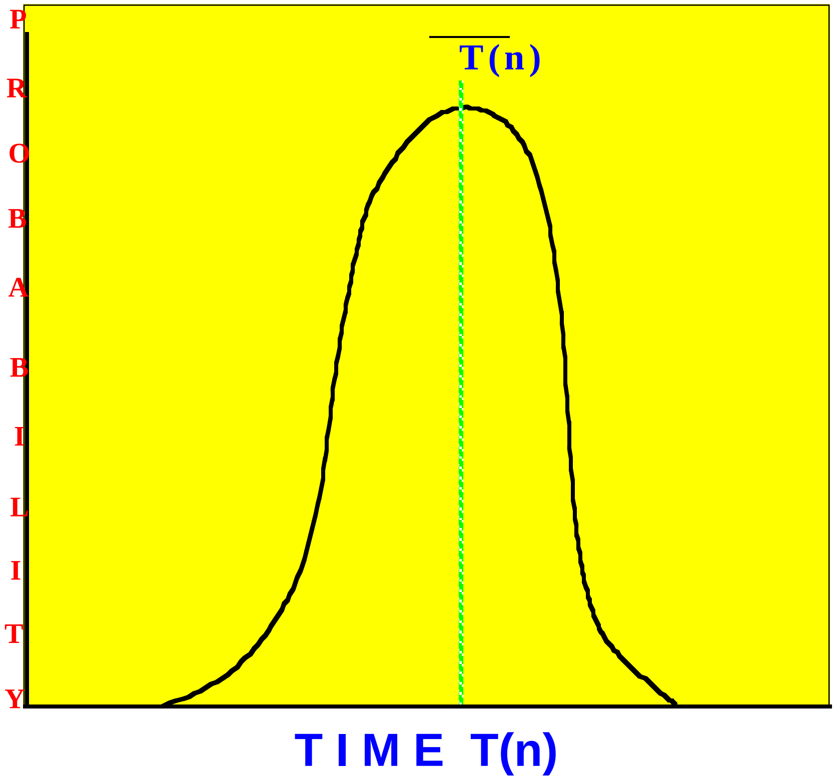
$r \leftarrow \text{RANDOM}(k)$ gives a random k bit number

Let $A_R(x)$ denote randomized algorithm with input x, random choices R

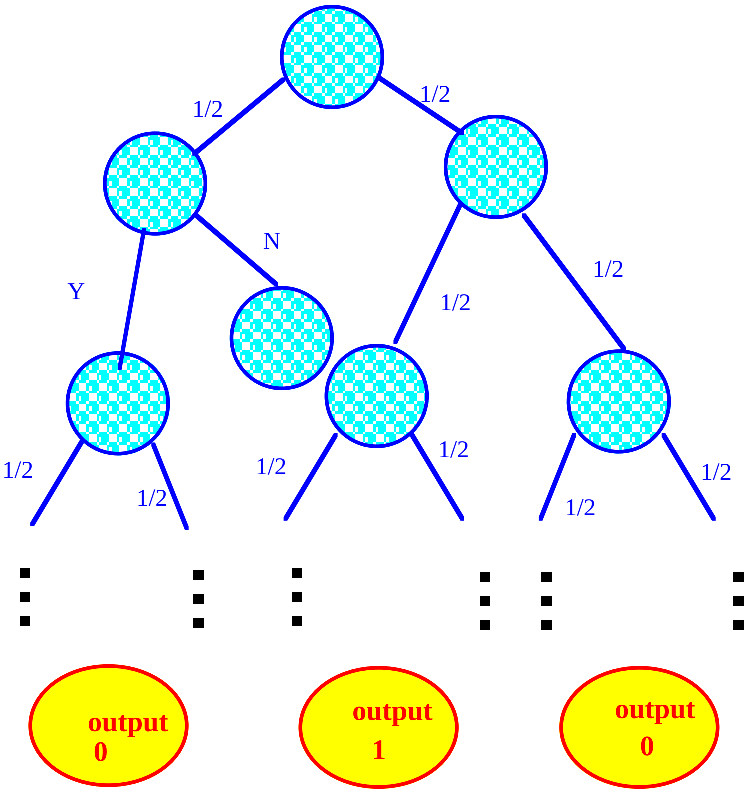
$$\overline{\text{Expected Time}}\ \text{input } X \quad \text{Time } (X) = \sum_{\forall R} \text{Time } (X) \quad \text{Prob } (R)$$

Expected Time Complexity

$$\overline{T(n)} = \max_{\{x \mid n = |x|\}} \overline{\text{Time } (x)}$$



A Randomized Computation



(1) If machine *outputs value v with prob >* $\frac{1}{2}$
then v is considered its output.

(2) machine *accepts input X* if outputs 1 with prob> $\frac{1}{2}$

(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

1	0	1	1	...	0

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,

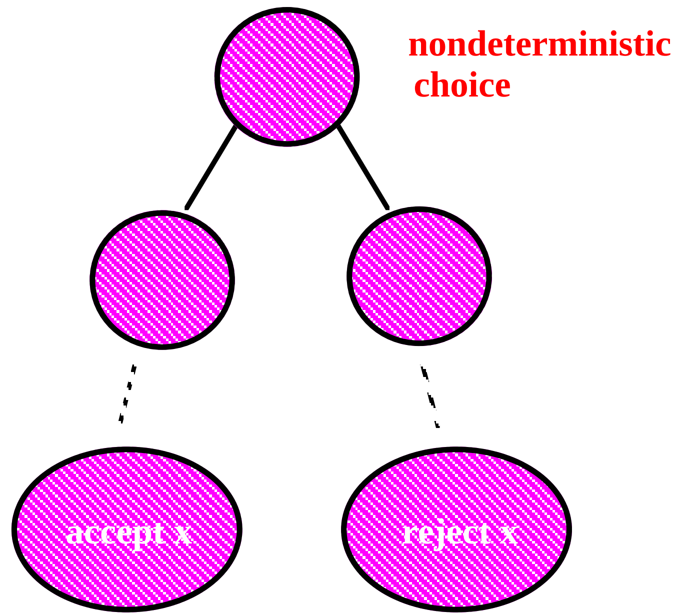
$\exists$ a non-uniform *deterministic* algorithm with polynomial time and polynomial advice!

(tricky proof)

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 = 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 TM machines

not known
probably
no

P = NP ?

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 NP 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!