The power of parallel computation

D. J. Bernstein

Thanks to:
University of Illinois at Chicago
NSF CCR–9983950
Alfred P. Sloan Foundation

How fast is sorting?

Input: array of n numbers.
Each number in $\{1, 2, \ldots, n^2\}$,
represented in binary.

Output: array of n numbers,
in increasing order,
represented in binary;
same multiset as input.

A machine is given the input
and computes the output.
How much time does it use?

on

is at Chicago

0

undation

How fast is sorting?

Input: array of n numbers.
Each number in $\{1, 2, \ldots, n^2\}$,
represented in binary.

Output: array of n numbers,
in increasing order,
represented in binary;
same multiset as input.

A machine is given the input
and computes the output.
How much time does it use?

Summarize scalabi
by reporting expor

$n^{o(1)}$ means $\log n$
$100n^{5/\log\log n} +$ √

$n^{1+o(1)}$ means n o
$n \log n$ or $n(7(\log$

(Definition: $o(1)$ r
function of n that
e.g. $5n = n^{1+(\log 5}$
$(\log 5)/\log n$ conv

At this level of det
how fast is the ma

How fast is sorting?

Input: array of n numbers.
Each number in $\{1, 2, \ldots, n^2\}$,
represented in binary.

Output: array of n numbers,
in increasing order,
represented in binary;
same multiset as input.

A machine is given the input
and computes the output.
How much time does it use?

Summarize scalability
by reporting exponent of n.

$n^{o(1)}$ means $\log n$ or $(\log n)^3$ or
$100n^{5/\log\log n} + \sqrt{1/n}$ or ...

$n^{1+o(1)}$ means n or $5n$ or
$n \log n$ or $n(7(\log n)^3 + 8)$ or ...

(Definition: $o(1)$ means any
function of n that converges to 0.
e.g. $5n = n^{1+(\log 5)/\log n}$;
$(\log 5)/\log n$ converges to 0.)

At this level of detail,
how fast is the machine?

g?

numbers.

$1, 2, \ldots, n^2\}$,

ary.

n numbers,

;

ary;

nput.

n the input

output.

oes it use?

Summarize scalability
by reporting exponent of n.

$n^{o(1)}$ means $\log n$ or $(\log n)^3$ or
$100n^{5/\log\log n} + \sqrt{1/n}$ or ...

$n^{1+o(1)}$ means n or $5n$ or
$n \log n$ or $n(7(\log n)^3 + 8)$ or ...

(Definition: $o(1)$ means any
function of n that converges to 0.
e.g. $5n = n^{1+(\log 5)/\log n}$;
$(\log 5)/\log n$ converges to 0.)

At this level of detail,
how fast is the machine?

The answer depen
how the machine v

Possibility 1: The
“1-tape Turing ma
using selection sor

Specifically: The r
a 1-dimensional ar
containing $n^{1+o(1)}$
Each cell stores n^{o}

Input and output a
stored in these cel

Summarize scalability
by reporting exponent of n.

$n^{o(1)}$ means $\log n$ or $(\log n)^3$ or $100n^{5/\log\log n} + \sqrt{1/n}$ or ...

$n^{1+o(1)}$ means n or $5n$ or $n \log n$ or $n(7(\log n)^3 + 8)$ or ...

(Definition: $o(1)$ means any function of n that converges to 0. e.g. $5n = n^{1+(\log 5)/\log n}$; $(\log 5)/\log n$ converges to 0.)

At this level of detail, how fast is the machine?

The answer depends on how the machine works.

Possibility 1: The machine is a "1-tape Turing machine using selection sort."

Specifically: The machine has a 1-dimensional array containing $n^{1+o(1)}$ "cells." Each cell stores $n^{o(1)}$ bits.

Input and output are stored in these cells.

lity

nent of n.

or $(\log n)^3$ or

$\sqrt{1/n}$ or ...

or $5n$ or

$n)^3 + 8)$ or ...

means any

converges to 0.

$5)/\log n$;

erges to 0.)

tail,

chine?

The answer depends on how the machine works.

Possibility 1: The machine is a "1-tape Turing machine using selection sort."

Specifically: The machine has a 1-dimensional array containing $n^{1+o(1)}$ "cells." Each cell stores $n^{o(1)}$ bits.

Input and output are stored in these cells.

The machine also

"head" moving thr

Head contains $n^{o(}$

Head can see the

its current array p

perform arithmeti

move to adjacent

Selection sort: He

looks at each arra

picks up the larges

moves it to the en

picks up the secon

etc.

The answer depends on
how the machine works.

Possibility 1: The machine is a
"1-tape Turing machine
using selection sort."

Specifically: The machine has
a 1-dimensional array
containing $n^{1+o(1)}$ "cells."
Each cell stores $n^{o(1)}$ bits.

Input and output are
stored in these cells.

The machine also has a
"head" moving through array.
Head contains $n^{o(1)}$ cells.

Head can see the cell at
its current array position;
perform arithmetic etc.;
move to adjacent array position.

Selection sort: Head
looks at each array position,
picks up the largest number,
moves it to the end of the array,
picks up the second largest,
etc.

ds on
works.

machine is a
achine
t."

machine has
rray
) "cells."
$o(1)$ bits.

are
ls.

The machine also has a
"head" moving through array.
Head contains $n^{o(1)}$ cells.

Head can see the cell at
its current array position;
perform arithmetic etc.;
move to adjacent array position.

Selection sort: Head
looks at each array position,
picks up the largest number,
moves it to the end of the array,
picks up the second largest,
etc.

Moving to adjacen
takes $n^{o(1)}$ second

Moving a number
takes $n^{1+o(1)}$ seco
Same for comparis

Total sorting time
$n^{2+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ dollars
for $n^{1+o(1)}$ cells.
Negligible extra co

The machine also has a "head" moving through array. Head contains $n^{o(1)}$ cells.

Head can see the cell at its current array position; perform arithmetic etc.; move to adjacent array position.

Selection sort: Head looks at each array position, picks up the largest number, moves it to the end of the array, picks up the second largest, etc.

Moving to adjacent array position takes $n^{o(1)}$ seconds.

Moving a number to end of array takes $n^{1+o(1)}$ seconds. Same for comparisons etc.

Total sorting time: $n^{2+o(1)}$ seconds.

Cost of machine: $n^{1+o(1)}$ dollars for $n^{1+o(1)}$ cells. Negligible extra cost for head.

has a
rough array.
1) cells.

cell at
osition;
c etc.;
array position.

ad
y position,
st number,
d of the array,
nd largest,

Moving to adjacent array position takes $n^{o(1)}$ seconds.

Moving a number to end of array takes $n^{1+o(1)}$ seconds.
Same for comparisons etc.

Total sorting time:
$n^{2+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ dollars
for $n^{1+o(1)}$ cells.
Negligible extra cost for head.

Possibility 2: The
"2-dimensional RA
using merge sort."

Machine has n^{1+o}
in a 2-dimensional
$n^{0.5+o(1)}$ rows, n^{0}
Machine also has a

Merge sort: Head
sorts first $\lfloor n/2 \rfloor$ n
sorts last $\lceil n/2 \rceil$ nu
merges the sorted

Moving to adjacent array position takes $n^{o(1)}$ seconds.

Moving a number to end of array takes $n^{1+o(1)}$ seconds.
Same for comparisons etc.

Total sorting time:
$n^{2+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ dollars
for $n^{1+o(1)}$ cells.
Negligible extra cost for head.

Possibility 2: The machine is a "2-dimensional RAM
using merge sort."

Machine has $n^{1+o(1)}$ cells
in a 2-dimensional array:
$n^{0.5+o(1)}$ rows, $n^{0.5+o(1)}$ columns.
Machine also has a head.

Merge sort: Head recursively
sorts first $\lfloor n/2 \rfloor$ numbers;
sorts last $\lceil n/2 \rceil$ numbers;
merges the sorted lists.

nt array position
ls.

to end of array
onds.
sons etc.

:

ost for head.

Possibility 2: The machine is a "2-dimensional RAM using merge sort."

Machine has $n^{1+o(1)}$ cells in a 2-dimensional array: $n^{0.5+o(1)}$ rows, $n^{0.5+o(1)}$ columns. Machine also has a head.

Merge sort: Head recursively sorts first $\lfloor n/2 \rfloor$ numbers; sorts last $\lceil n/2 \rceil$ numbers; merges the sorted lists.

Merging requires n
to "random" array

Average jump: n^{0}
to adjacent array p

Each move takes n

Total sorting time:
$n^{1.5+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ dollars.

Possibility 2: The machine is a "2-dimensional RAM using merge sort."

Machine has $n^{1+o(1)}$ cells in a 2-dimensional array: $n^{0.5+o(1)}$ rows, $n^{0.5+o(1)}$ columns. Machine also has a head.

Merge sort: Head recursively sorts first $\lfloor n/2 \rfloor$ numbers; sorts last $\lceil n/2 \rceil$ numbers; merges the sorted lists.

Merging requires $n^{1+o(1)}$ jumps to "random" array positions.

Average jump: $n^{0.5+o(1)}$ moves to adjacent array positions.

Each move takes $n^{o(1)}$ seconds.

Total sorting time: $n^{1.5+o(1)}$ seconds.

Cost of machine: once again $n^{1+o(1)}$ dollars.

machine is a
AM

(1) cells
array:
.5+o(1) columns.
a head.

recursively
umbers;
umbers;
lists.

Merging requires $n^{1+o(1)}$ jumps to "random" array positions.

Average jump: $n^{0.5+o(1)}$ moves to adjacent array positions.

Each move takes $n^{o(1)}$ seconds.

Total sorting time: $n^{1.5+o(1)}$ seconds.

Cost of machine: once again $n^{1+o(1)}$ dollars.

Possibility 3: The
"pipelined 2-dimen
using radix-2 sort."

Machine has n^{1+o}
in a 2-dimensional
Each cell in the ar
network links to th
cells in the same c
Each cell in the to
network links to th
cells in the top row

Merging requires $n^{1+o(1)}$ jumps to "random" array positions.

Average jump: $n^{0.5+o(1)}$ moves to adjacent array positions.

Each move takes $n^{o(1)}$ seconds.

Total sorting time:
$n^{1.5+o(1)}$ seconds.

Cost of machine: once again $n^{1+o(1)}$ dollars.

Possibility 3: The machine is a "pipelined 2-dimensional RAM using radix-2 sort."

Machine has $n^{1+o(1)}$ cells in a 2-dimensional array.
Each cell in the array has network links to the 2 adjacent cells in the same column.
Each cell in the top row has network links to the 2 adjacent cells in the top row.

$n^{1+o(1)}$ jumps

y positions.

$.5+o(1)$ moves

positions.

$n^{o(1)}$ seconds.

f

once again

Possibility 3: The machine is a "pipelined 2-dimensional RAM using radix-2 sort."

Machine has $n^{1+o(1)}$ cells in a 2-dimensional array. Each cell in the array has network links to the 2 adjacent cells in the same column. Each cell in the top row has network links to the 2 adjacent cells in the top row.

Machine also has a attached to top-le

CPU can read/wri sending request th Does not need to before sending ne

CPU can read an of $n^{0.5+o(1)}$ cells in $n^{0.5+o(1)}$ secon Sends all requests, then receives resp

Possibility 3: The machine is a "pipelined 2-dimensional RAM using radix-2 sort."

Machine has $n^{1+o(1)}$ cells in a 2-dimensional array. Each cell in the array has network links to the 2 adjacent cells in the same column. Each cell in the top row has network links to the 2 adjacent cells in the top row.

Machine also has a CPU attached to top-left cell.

CPU can read/write any cell by sending request through network. Does not need to wait for response before sending next request.

CPU can read an entire row of $n^{0.5+o(1)}$ cells in $n^{0.5+o(1)}$ seconds. Sends all requests, then receives responses.

machine is a
nsional RAM
"

$^{(1)}$ cells
array.
ray has
ne 2 adjacent
column.
p row has
ne 2 adjacent
v.

Machine also has a CPU
attached to top-left cell.

CPU can read/write any cell by
sending request through network.
Does not need to wait for response
before sending next request.

CPU can read an entire row
of $n^{0.5+o(1)}$ cells
in $n^{0.5+o(1)}$ seconds.
Sends all requests,
then receives responses.

Radix-2 sort: CPU
shuffles array using
even numbers befo
3 1 4 1 5 9 2 6 $\mapsto$
4 2 6 3 1 1 5 9.

Then using bit 1:
4 1 1 5 9 2 6 3.

Then using bit 2:
1 1 9 2 3 4 5 6.

Then using bit 3:
1 1 2 3 4 5 6 9.

etc.

Machine also has a CPU attached to top-left cell.

CPU can read/write any cell by sending request through network. Does not need to wait for response before sending next request.

CPU can read an entire row of $n^{0.5+o(1)}$ cells in $n^{0.5+o(1)}$ seconds. Sends all requests, then receives responses.

Radix-2 sort: CPU shuffles array using bit 0, even numbers before odd. 3 1 4 1 5 9 2 6 $\mapsto$ 4 2 6 3 1 1 5 9.

Then using bit 1: 4 1 1 5 9 2 6 3.

Then using bit 2: 1 1 9 2 3 4 5 6.

Then using bit 3: 1 1 2 3 4 5 6 9.

etc.

a CPU
ft cell.

te any cell by
rough network.
wait for response
xt request.

entire row

ds.

onses.

Radix-2 sort: CPU
shuffles array using bit 0,
even numbers before odd.
3 1 4 1 5 9 2 6 $\mapsto$
4 2 6 3 1 1 5 9.

Then using bit 1:
4 1 1 5 9 2 6 3.

Then using bit 2:
1 1 9 2 3 4 5 6.

Then using bit 3:
1 1 2 3 4 5 6 9.

etc.

Each shuffle takes
$n^{1+o(1)}$ seconds.

$n^{o(1)}$ shuffles.

Total sorting time
$n^{1+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ dollars.

Radix-2 sort: CPU shuffles array using bit 0, even numbers before odd.

3 1 4 1 5 9 2 6 $\mapsto$
4 2 6 3 1 1 5 9.

Then using bit 1:
4 1 1 5 9 2 6 3.

Then using bit 2:
1 1 9 2 3 4 5 6.

Then using bit 3:
1 1 2 3 4 5 6 9.

etc.

Each shuffle takes $n^{1+o(1)}$ seconds.

$n^{o(1)}$ shuffles.

Total sorting time: $n^{1+o(1)}$ seconds.

Cost of machine: once again $n^{1+o(1)}$ dollars.

U

g bit 0,

ore odd.

Each shuffle takes $n^{1+o(1)}$ seconds.

$n^{o(1)}$ shuffles.

Total sorting time: $n^{1+o(1)}$ seconds.

Cost of machine: once again $n^{1+o(1)}$ dollars.

Possibility 4: The
"2-dimensional me
using Schimmler s

Machine has n^{1+o}
in a 2-dimensional
Each cell has netw
to the 4 adjacent

Machine also has
attached to top-le
CPU broadcasts in
to all of the cells,
cells do most of th

Each shuffle takes
$n^{1+o(1)}$ seconds.

$n^{o(1)}$ shuffles.

Total sorting time:
$n^{1+o(1)}$ seconds.

Cost of machine: once again
$n^{1+o(1)}$ dollars.

Possibility 4: The machine is a "2-dimensional mesh using Schimmler sort."

Machine has $n^{1+o(1)}$ cells in a 2-dimensional array. Each cell has network links to the 4 adjacent cells.

Machine also has a CPU attached to top-left cell. CPU broadcasts instructions to all of the cells, but cells do most of the processing.

:

once again

Possibility 4: The machine is a "2-dimensional mesh using Schimmler sort."

Machine has $n^{1+o(1)}$ cells in a 2-dimensional array. Each cell has network links to the 4 adjacent cells.

Machine also has a CPU attached to top-left cell. CPU broadcasts instructions to all of the cells, but cells do most of the processing.

Sort row of $n^{0.5+o}$
in $n^{0.5+o(1)}$ secon

Sort each pair in p
3 1 4 1 5 9 2 6 $\mapsto$
1 3 1 4 5 9 2 6

Sort alternate pair
1 3 1 4 5 9 2 6 $\mapsto$
1 1 3 4 5 2 9 6

Repeat until numb
equals row length.

Sort *each* row, in
in $n^{0.5+o(1)}$ secon

Possibility 4: The machine is a "2-dimensional mesh using Schimmler sort."

Machine has $n^{1+o(1)}$ cells in a 2-dimensional array. Each cell has network links to the 4 adjacent cells.

Machine also has a CPU attached to top-left cell. CPU broadcasts instructions to all of the cells, but cells do most of the processing.

Sort row of $n^{0.5+o(1)}$ cells in $n^{0.5+o(1)}$ seconds:

Sort each pair in parallel.

$\underline{3\ 1}\ \underline{4\ 1}\ \underline{5\ 9}\ \underline{2\ 6} \mapsto$

1 3 1 4 5 9 2 6

Sort alternate pairs in parallel.

$1\ \underline{3\ 1}\ \underline{4\ 5}\ \underline{9\ 2}\ 6 \mapsto$

1 1 3 4 5 2 9 6

Repeat until number of steps equals row length.

Sort *each* row, in parallel, in $n^{0.5+o(1)}$ seconds.

machine is a
esh
ort."

(1) cells
array.
ork links
cells.

a CPU
ft cell.
nstructions
but
ne processing.

Sort row of $n^{0.5+o(1)}$ cells in $n^{0.5+o(1)}$ seconds:

Sort each pair in parallel.

3 1 4 1 5 9 2 6 $\mapsto$

1 3 1 4 5 9 2 6

Sort alternate pairs in parallel.

1 3 1 4 5 9 2 6 $\mapsto$

1 1 3 4 5 2 9 6

Repeat until number of steps equals row length.

Sort *each* row, in parallel, in $n^{0.5+o(1)}$ seconds.

Schimmler sort:
Recursively sort qu
in parallel. Then f
Sort each column
Sort each row in p
Sort each column
Sort each row in p

With proper choic
left-to-right/right-
for each row, can
that this sorts who

Sort row of $n^{0.5+o(1)}$ cells
in $n^{0.5+o(1)}$ seconds:

Sort each pair in parallel.
$\underline{3\ 1}\ \underline{4\ 1}\ \underline{5\ 9}\ \underline{2\ 6} \mapsto$
1 3 1 4 5 9 2 6

Sort alternate pairs in parallel.
$1\ \underline{3\ 1}\ \underline{4\ 5}\ \underline{9\ 2}\ 6 \mapsto$
1 1 3 4 5 2 9 6

Repeat until number of steps
equals row length.

Sort *each* row, in parallel,
in $n^{0.5+o(1)}$ seconds.

Schimmler sort:
Recursively sort quadrants
in parallel. Then four steps:
Sort each column in parallel.
Sort each row in parallel.
Sort each column in parallel.
Sort each row in parallel.

With proper choice of
left-to-right/right-to-left
for each row, can prove
that this sorts whole array.

(1) cells
ds:

arallel.

s in parallel.

er of steps

parallel,
ds.

Schimmler sort:
Recursively sort quadrants
in parallel. Then four steps:
Sort each column in parallel.
Sort each row in parallel.
Sort each column in parallel.
Sort each row in parallel.

With proper choice of
left-to-right/right-to-left
for each row, can prove
that this sorts whole array.

For example, assu
this 8×8 array is

3	1	4	1	5	9
5	3	5	8	9	7
2	3	8	4	6	2
3	3	8	3	2	7
0	2	8	8	4	1
1	6	9	3	9	9
5	1	0	5	8	2
7	4	9	4	4	5

Schimmler sort:
Recursively sort quadrants in parallel. Then four steps:
Sort each column in parallel.
Sort each row in parallel.
Sort each column in parallel.
Sort each row in parallel.

With proper choice of left-to-right/right-to-left for each row, can prove that this sorts whole array.

For example, assume that this 8×8 array is in cells:

3	1	4	1	5	9	2	6
5	3	5	8	9	7	9	3
2	3	8	4	6	2	6	4
3	3	8	3	2	7	9	5
0	2	8	8	4	1	9	7
1	6	9	3	9	9	3	7
5	1	0	5	8	2	0	9
7	4	9	4	4	5	9	2

uadrants
our steps:
in parallel.
arallel.
in parallel.
arallel.

e of
to-left
prove
le array.

For example, assume that this 8×8 array is in cells:

3	1	4	1	5	9	2	6
5	3	5	8	9	7	9	3
2	3	8	4	6	2	6	4
3	3	8	3	2	7	9	5
0	2	8	8	4	1	9	7
1	6	9	3	9	9	3	7
5	1	0	5	8	2	0	9
7	4	9	4	4	5	9	2

Recursively sort qu
top $\rightarrow$, bottom $\leftarrow$

1	1	2	3	2	2
3	3	3	3	4	5
3	4	4	5	6	6
5	8	8	8	9	9
1	1	0	0	2	2
4	4	3	2	5	4
7	6	5	5	9	8
9	9	8	8	9	9

For example, assume that this 8×8 array is in cells:

3	1	4	1	5	9	2	6
5	3	5	8	9	7	9	3
2	3	8	4	6	2	6	4
3	3	8	3	2	7	9	5
0	2	8	8	4	1	9	7
1	6	9	3	9	9	3	7
5	1	0	5	8	2	0	9
7	4	9	4	4	5	9	2

Recursively sort quadrants, top $\rightarrow$, bottom $\leftarrow$:

1	1	2	3	2	2	2	3
3	3	3	3	4	5	5	6
3	4	4	5	6	6	7	7
5	8	8	8	9	9	9	9
1	1	0	0	2	2	1	0
4	4	3	2	5	4	4	3
7	6	5	5	9	8	7	7
9	9	8	8	9	9	9	9

me that
in cells:

2	6
9	3
6	4
9	5
9	7
3	7
0	9
9	2

Recursively sort quadrants,
top $\rightarrow$, bottom $\leftarrow$:

1	1	2	3	2	2	2	3
3	3	3	3	4	5	5	6
3	4	4	5	6	6	7	7
5	8	8	8	9	9	9	9
1	1	0	0	2	2	1	0
4	4	3	2	5	4	4	3
7	6	5	5	9	8	7	7
9	9	8	8	9	9	9	9

Sort each column
in parallel:

1	1	0	0	2	2
1	1	2	2	2	2
3	3	3	3	4	4
3	4	3	3	5	5
4	4	4	5	6	6
5	6	5	5	9	8
7	8	8	8	9	9
9	9	8	8	9	9

Recursively sort quadrants, top $\rightarrow$, bottom $\leftarrow$:

1	1	2	3	2	2	2	3
3	3	3	3	4	5	5	6
3	4	4	5	6	6	7	7
5	8	8	8	9	9	9	9
1	1	0	0	2	2	1	0
4	4	3	2	5	4	4	3
7	6	5	5	9	8	7	7
9	9	8	8	9	9	9	9

Sort each column in parallel:

1	1	0	0	2	2	1	0
1	1	2	2	2	2	2	3
3	3	3	3	4	4	4	3
3	4	3	3	5	5	5	6
4	4	4	5	6	6	7	7
5	6	5	5	9	8	7	7
7	8	8	8	9	9	9	9
9	9	8	8	9	9	9	9

uadrants,

:

2	3
5	6
7	7
9	9
1	0
4	3
7	7
9	9

Sort each column in parallel:

1	1	0	0	2	2	1	0
1	1	2	2	2	2	2	3
3	3	3	3	4	4	4	3
3	4	3	3	5	5	5	6
4	4	4	5	6	6	7	7
5	6	5	5	9	8	7	7
7	8	8	8	9	9	9	9
9	9	8	8	9	9	9	9

Sort each row in p

alternately ←, →:

0	0	0	1	1	1
3	2	2	2	2	2
3	3	3	3	3	4
6	5	5	5	4	3
4	4	4	5	6	6
9	8	7	7	6	5
7	8	8	8	9	9
9	9	9	9	9	9

Sort each column in parallel:

1	1	0	0	2	2	1	0
1	1	2	2	2	2	2	3
3	3	3	3	4	4	4	3
3	4	3	3	5	5	5	6
4	4	4	5	6	6	7	7
5	6	5	5	9	8	7	7
7	8	8	8	9	9	9	9
9	9	8	8	9	9	9	9

Sort each row in parallel, alternately $\leftarrow$, $\rightarrow$:

0	0	0	1	1	1	2	2
3	2	2	2	2	2	1	1
3	3	3	3	3	4	4	4
6	5	5	5	4	3	3	3
4	4	4	5	6	6	7	7
9	8	7	7	6	5	5	5
7	8	8	8	9	9	9	9
9	9	9	9	9	9	8	8

1	0
2	3
4	3
5	6
7	7
7	7
9	9
9	9

Sort each row in parallel, alternately ←, →:

0	0	0	1	1	1	2	2
3	2	2	2	2	2	1	1
3	3	3	3	3	4	4	4
6	5	5	5	4	3	3	3
4	4	4	5	6	6	7	7
9	8	7	7	6	5	5	5
7	8	8	8	9	9	9	9
9	9	9	9	9	9	8	8

Sort each column in parallel:

0	0	0	1	1	1
3	2	2	2	2	2
3	3	3	3	3	3
4	4	4	5	4	4
6	5	5	5	6	5
7	8	7	7	6	6
9	8	8	8	9	9
9	9	9	9	9	9

Sort each row in parallel, alternately ←, →:

0	0	0	1	1	1	2	2
3	2	2	2	2	2	1	1
3	3	3	3	3	4	4	4
6	5	5	5	4	3	3	3
4	4	4	5	6	6	7	7
9	8	7	7	6	5	5	5
7	8	8	8	9	9	9	9
9	9	9	9	9	9	8	8

Sort each column in parallel:

0	0	0	1	1	1	1	1
3	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	5	4	4	4	4
6	5	5	5	6	5	5	5
7	8	7	7	6	6	7	7
9	8	8	8	9	9	8	8
9	9	9	9	9	9	9	9

arallel,

2	2
1	1
4	4
3	3
7	7
5	5
9	9
8	8

Sort each column in parallel:

0	0	0	1	1	1	1	1
3	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	5	4	4	4	4
6	5	5	5	6	5	5	5
7	8	7	7	6	6	7	7
9	8	8	8	9	9	8	8
9	9	9	9	9	9	9	9

Sort each row in p

$\leftarrow$ or $\rightarrow$ as desirec

0	0	0	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3
4	4	4	4	4	4
5	5	5	5	5	5
6	6	7	7	7	7
8	8	8	8	8	9
9	9	9	9	9	9

Sort each column in parallel:

0	0	0	1	1	1	1	1
3	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	5	4	4	4	4
6	5	5	5	6	5	5	5
7	8	7	7	6	6	7	7
9	8	8	8	9	9	8	8
9	9	9	9	9	9	9	9

Sort each row in parallel, $\leftarrow$ or $\rightarrow$ as desired:

0	0	0	1	1	1	1	1
2	2	2	2	2	2	2	3
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	5
5	5	5	5	5	5	6	6
6	6	7	7	7	7	7	8
8	8	8	8	8	9	9	9
9	9	9	9	9	9	9	9

1	1
2	2
3	3
4	4
5	5
7	7
8	8
9	9

Sort each row in parallel,
$\leftarrow$ or $\rightarrow$ as desired:

0	0	0	1	1	1	1	1
2	2	2	2	2	2	2	3
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	5
5	5	5	5	5	5	6	6
6	6	7	7	7	7	7	8
8	8	8	8	8	9	9	9
9	9	9	9	9	9	9	9

Sort one row
in $n^{0.5+o(1)}$ secon

All rows in parallel
$n^{0.5+o(1)}$ seconds.

Total sorting time
$n^{0.5+o(1)}$ seconds.

Cost of machine:
$n^{1+o(1)}$ dollars.

Sort each row in parallel,
$\leftarrow$ or $\rightarrow$ as desired:

0	0	0	1	1	1	1	1
2	2	2	2	2	2	2	3
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	5
5	5	5	5	5	5	6	6
6	6	7	7	7	7	7	8
8	8	8	8	8	9	9	9
9	9	9	9	9	9	9	9

Sort one row
in $n^{0.5+o(1)}$ seconds.

All rows in parallel:
$n^{0.5+o(1)}$ seconds.

Total sorting time:
$n^{0.5+o(1)}$ seconds.

Cost of machine: once again
$n^{1+o(1)}$ dollars.

arallel,

d:

1	1
2	3
3	3
4	5
6	6
7	8
9	9
9	9

Sort one row
in $n^{0.5+o(1)}$ seconds.

All rows in parallel:
$n^{0.5+o(1)}$ seconds.

Total sorting time:
$n^{0.5+o(1)}$ seconds.

Cost of machine: once again
$n^{1+o(1)}$ dollars.

Some philosophica

1-tape Turing mac
RAMs, 2-dimensio
compute the same

Prove this by prov
each machine can
computations on t

(We believe that e
reasonable model
can be simulated b
1-tape Turing mac
"Church-Turing th

Sort one row
in $n^{0.5+o(1)}$ seconds.

All rows in parallel:
$n^{0.5+o(1)}$ seconds.

Total sorting time:
$n^{0.5+o(1)}$ seconds.

Cost of machine: once again
$n^{1+o(1)}$ dollars.

Some philosophical notes

1-tape Turing machines,
RAMs, 2-dimensional meshes
compute the same functions.

Prove this by proving that
each machine can simulate
computations on the others.

(We believe that *every*
reasonable model of computation
can be simulated by a
1-tape Turing machine.
"Church-Turing thesis.")

ds.

l:

:

once again

Some philosophical notes

1-tape Turing machines, RAMs, 2-dimensional meshes compute the same functions.

Prove this by proving that each machine can simulate computations on the others.

(We believe that *every* reasonable model of computation can be simulated by a 1-tape Turing machine. "Church-Turing thesis.")

1-tape Turing mac
RAMs, 2-dimensio
compute the same
in polynomial time
at polynomial cost

Prove this by prov
simulations are po

(Is this true for ev
reasonable model
Quantum compute
factor in polynomi
Can Turing machi
Can quantum com

Some philosophical notes

1-tape Turing machines,
RAMs, 2-dimensional meshes
compute the same functions.

Prove this by proving that
each machine can simulate
computations on the others.

(We believe that *every*
reasonable model of computation
can be simulated by a
1-tape Turing machine.
"Church-Turing thesis.")

1-tape Turing machines,
RAMs, 2-dimensional meshes
compute the same functions
in polynomial time
at polynomial cost.

Prove this by proving that
simulations are polynomial.

(Is this true for every
reasonable model of computation?
Quantum computers can
factor in polynomial time.
Can Turing machines do that?
Can quantum computers be built?)

al notes

chines,
nal meshes
e functions.

ing that
simulate
he others.

every
of computation
by a
chine.
esis.")

1-tape Turing machines,
RAMs, 2-dimensional meshes
compute the same functions
in polynomial time
at polynomial cost.

Prove this by proving that
simulations are polynomial.

(Is this true for every
reasonable model of computation?
Quantum computers can
factor in polynomial time.
Can Turing machines do that?
Can quantum computers be built?)

1-tape Turing mac
RAMs, 2-dimensio
do not compute
the same function
within, e.g., time n
and cost $n^{1+o(1)}$.

Example: 1-tape T
cannot sort in n^{1+}
Too local.

Example: 2-dimen
cannot sort in $n^{0.5}$
Too sequential.

1-tape Turing machines,
RAMs, 2-dimensional meshes
compute the same functions
in polynomial time
at polynomial cost.

Prove this by proving that
simulations are polynomial.

(Is this true for every
reasonable model of computation?
Quantum computers can
factor in polynomial time.
Can Turing machines do that?
Can quantum computers be built?)

1-tape Turing machines,
RAMs, 2-dimensional meshes
do not compute
the same functions
within, e.g., time $n^{1+o(1)}$
and cost $n^{1+o(1)}$.

Example: 1-tape Turing machine
cannot sort in $n^{1+o(1)}$ seconds.
Too local.

Example: 2-dimensional RAM
cannot sort in $n^{0.5+o(1)}$ seconds.
Too sequential.

chines,
nal meshes
functions
e
t.

ing that
lynomial.

ery
of computation?
ers can
al time.
nes do that?
puters be built?)

1-tape Turing machines,
RAMs, 2-dimensional meshes
do not compute
the same functions
within, e.g., time $n^{1+o(1)}$
and cost $n^{1+o(1)}$.

Example: 1-tape Turing machine
cannot sort in $n^{1+o(1)}$ seconds.
Too local.

Example: 2-dimensional RAM
cannot sort in $n^{0.5+o(1)}$ seconds.
Too sequential.

Review of sorting
measured in secon
machine costing n

$n^{2.0+o(1)}$: 1-tape
$n^{1.5+o(1)}$: 2-dimen
$n^{1.0+o(1)}$: pipeline
$n^{0.5+o(1)}$: 2-dimen

Why does anyone
sorting time is n^{1-}
Why choose third
Silly! Fourth mach

1-tape Turing machines,
RAMs, 2-dimensional meshes
do not compute
the same functions
within, e.g., time $n^{1+o(1)}$
and cost $n^{1+o(1)}$.

Example: 1-tape Turing machine
cannot sort in $n^{1+o(1)}$ seconds.
Too local.

Example: 2-dimensional RAM
cannot sort in $n^{0.5+o(1)}$ seconds.
Too sequential.

Review of sorting times,
measured in seconds, for
machine costing $n^{1+o(1)}$ dollars:

$n^{2.0+o(1)}$: 1-tape Turing machine.
$n^{1.5+o(1)}$: 2-dimensional RAM.
$n^{1.0+o(1)}$: pipelined RAM.
$n^{0.5+o(1)}$: 2-dimensional mesh.

Why does anyone say that
sorting time is $n^{1+o(1)}$?
Why choose third machine?
Silly! Fourth machine is better!

chines,
nal meshes

s
$n^{1+o(1)}$

Turing machine
$o(1)$ seconds.

sional RAM
$5+o(1)$ seconds.

Review of sorting times,
measured in seconds, for
machine costing $n^{1+o(1)}$ dollars:

$n^{2.0+o(1)}$: 1-tape Turing machine.
$n^{1.5+o(1)}$: 2-dimensional RAM.
$n^{1.0+o(1)}$: pipelined RAM.
$n^{0.5+o(1)}$: 2-dimensional mesh.

Why does anyone say that
sorting time is $n^{1+o(1)}$?
Why choose third machine?
Silly! Fourth machine is better!

Warning: $o(1)$ is a
Speedup factor su
might not be a sp
for small values of

When n is small,
RAM might seem
sensible machine c

But, once n is larg
having a huge me
waiting for a singl
is a silly machine c

Review of sorting times,
measured in seconds, for
machine costing $n^{1+o(1)}$ dollars:

$n^{2.0+o(1)}$: 1-tape Turing machine.
$n^{1.5+o(1)}$: 2-dimensional RAM.
$n^{1.0+o(1)}$: pipelined RAM.
$n^{0.5+o(1)}$: 2-dimensional mesh.

Why does anyone say that
sorting time is $n^{1+o(1)}$?
Why choose third machine?
Silly! Fourth machine is better!

Warning: $o(1)$ is asymptotic.
Speedup factor such as $n^{0.5+o(1)}$
might not be a speedup
for small values of n.

When n is small,
RAM might seem to be a
sensible machine design.

But, once n is large enough,
having a huge memory
waiting for a single CPU
is a silly machine design.

times,
ds, for
$^{1+o(1)}$ dollars:

Turing machine.
nsional RAM.
ed RAM.
nsional mesh.

say that
$+o(1)$?
machine?
hine is better!

Warning: $o(1)$ is asymptotic.
Speedup factor such as $n^{0.5+o(1)}$
might not be a speedup
for small values of n.

When n is small,
RAM might seem to be a
sensible machine design.

But, once n is large enough,
having a huge memory
waiting for a single CPU
is a silly machine design.

Myth:
Parallel computati
improve price-perf
p parallel compute
may reduce time b
but increase cost b

Reality: Can often
a *large* serial comp
into p *small* parall
so cost does *not*
increase by factor

Warning: $o(1)$ is asymptotic.
Speedup factor such as $n^{0.5+o(1)}$
might not be a speedup
for small values of n.

When n is small,
RAM might seem to be a
sensible machine design.

But, once n is large enough,
having a huge memory
waiting for a single CPU
is a silly machine design.

Myth:
Parallel computation cannot
improve price-performance ratio;
p parallel computers
may reduce time by factor p
but increase cost by factor p.

Reality: Can often convert
a *large* serial computer
into p *small* parallel cells,
so cost does *not*
increase by factor p.

asymptotic.
ch as $n^{0.5+o(1)}$
eedup
n.

to be a
design.

ge enough,
mory
e CPU
design.

Myth:
Parallel computation cannot improve price-performance ratio; p parallel computers may reduce time by factor p but increase cost by factor p.

Reality: Can often convert a *large* serial computer into p *small* parallel cells, so cost does *not* increase by factor p.

Myth: Designing a
cannot produce m
small constant-fac
compared to, e.g.,
What matters is s
streamlining, such
instruction-decodi

Reality: In 1997, D
was 1000 times fa
set of Pentiums a
What matters is p

Myth:
Parallel computation cannot improve price-performance ratio; p parallel computers may reduce time by factor p but increase cost by factor p.

Reality: Can often convert a *large* serial computer into p *small* parallel cells, so cost does *not* increase by factor p.

Myth: Designing a new machine cannot produce more than a small constant-factor improvement compared to, e.g., a Pentium. What matters is special-purpose streamlining, such as reducing instruction-decoding costs.

Reality: In 1997, DES Cracker was 1000 times faster than a set of Pentiums at the same price. What matters is parallelism.

on cannot
ormance ratio;
ers
y factor p
y factor p.

convert
puter
el cells,

p.

Myth: Designing a new machine cannot produce more than a small constant-factor improvement compared to, e.g., a Pentium. What matters is special-purpose streamlining, such as reducing instruction-decoding costs.

Reality: In 1997, DES Cracker was 1000 times faster than a set of Pentiums at the same price. What matters is parallelism.

Future computers
massively parallel
Look at $o(1)$ detai
we've reached larg

Computer designer
today's RAM-style
just as we laugh a
a 1-tape Turing m

Algorithm experts
today's dominant
algorithm analysis,
count CPU "opera
view memory acce

Myth: Designing a new machine cannot produce more than a small constant-factor improvement compared to, e.g., a Pentium. What matters is special-purpose streamlining, such as reducing instruction-decoding costs.

Reality: In 1997, DES Cracker was 1000 times faster than a set of Pentiums at the same price. What matters is parallelism.

Future computers will be massively parallel meshes. Look at $o(1)$ details to see that we've reached large enough n.

Computer designers will laugh at today's RAM-style machines, just as we laugh at a 1-tape Turing machine.

Algorithm experts will laugh at today's dominant style of algorithm analysis, where we count CPU "operations" and view memory access as free.

a new machine
ore than a
tor improvement
a Pentium.
pecial-purpose
as reducing
ng costs.

DES Cracker
ster than a
t the same price.
arallelism.

Future computers will be massively parallel meshes. Look at $o(1)$ details to see that we've reached large enough n.

Computer designers will laugh at today's RAM-style machines, just as we laugh at a 1-tape Turing machine.

Algorithm experts will laugh at today's dominant style of algorithm analysis, where we count CPU "operations" and view memory access as free.

Collision search

Common cryptana
Find collision in H

Input: Program to
at high speed.
H is a function fro
256-bit strings to
256-bit strings.

Output: 256-bit st
such that $x_1 \neq x_2$
and $H(x_1) = H(x$

Future computers will be massively parallel meshes. Look at $o(1)$ details to see that we've reached large enough n.

Computer designers will laugh at today's RAM-style machines, just as we laugh at a 1-tape Turing machine.

Algorithm experts will laugh at today's dominant style of algorithm analysis, where we count CPU "operations" and view memory access as free.

Collision search

Common cryptanalytic problem: Find collision in H.

Input: Program to compute H at high speed. H is a function from 256-bit strings to 256-bit strings.

Output: 256-bit strings x_1, x_2 such that $x_1 \neq x_2$ and $H(x_1) = H(x_2)$.

will be
meshes.
ils to see that
ge enough n.

rs will laugh at
e machines,
t
achine.

will laugh at
style of
, where we
ations" and
ss as free.

Collision search

Common cryptanalytic problem:
Find collision in H.

Input: Program to compute H
at high speed.
H is a function from
256-bit strings to
256-bit strings.

Output: 256-bit strings x_1, x_2
such that $x_1 \neq x_2$
and $H(x_1) = H(x_2)$.

For any 256-bit r:
Compute $H(r), H($
until finding a stri
that begins with 4
(A "distinguished
Call that string $Z($

Oops, $Z(r)$ might
But usually it does

Computing $Z(r)$ t
involves $\approx 2^{40}$ inp

Collision search

Common cryptanalytic problem:
Find collision in H.

Input: Program to compute H at high speed.
H is a function from 256-bit strings to 256-bit strings.

Output: 256-bit strings x_1, x_2 such that $x_1 \neq x_2$ and $H(x_1) = H(x_2)$.

For any 256-bit r:
Compute $H(r), H(H(r)), \ldots$ until finding a string that begins with 40 zero bits.
(A "distinguished point.")
Call that string $Z(r)$.

Oops, $Z(r)$ might not exist.
But usually it does.

Computing $Z(r)$ typically involves $\approx 2^{40}$ inputs to H.

lytic problem:
/.

o compute H

om

trings x_1, x_2

$_2)$.

For any 256-bit r:
Compute $H(r), H(H(r)), \ldots$
until finding a string
that begins with 40 zero bits.
(A "distinguished point.")
Call that string $Z(r)$.

Oops, $Z(r)$ might not exist.
But usually it does.

Computing $Z(r)$ typically
involves $\approx 2^{40}$ inputs to H.

Choose random r_1
Compute $Z(r_1), Z$

Uses $\approx 2^{40}n$ input
$r_1, H(r_1), H^2(r_1),$
$r_2, H(r_2), H^2(r_2),$
$\vdots$
$r_n, H(r_n), H^2(r_n)$

"Birthday paradox
$\approx 2^{79}n^2$ input pai
chances for a colli

For any 256-bit r:
Compute $H(r), H(H(r)), \ldots$
until finding a string
that begins with 40 zero bits.
(A "distinguished point.")
Call that string $Z(r)$.

Oops, $Z(r)$ might not exist.
But usually it does.

Computing $Z(r)$ typically
involves $\approx 2^{40}$ inputs to H.

Choose random $r_1, r_2, \ldots, r_n$.
Compute $Z(r_1), Z(r_2), \ldots, Z(r_n)$.

Uses $\approx 2^{40}n$ inputs to H:

$$r_1, H(r_1), H^2(r_1), H^3(r_1), \ldots$$
$$r_2, H(r_2), H^2(r_2), H^3(r_2), \ldots$$
$$\vdots$$
$$r_n, H(r_n), H^2(r_n), H^3(r_n), \ldots$$

"Birthday paradox":
$\approx 2^{79}n^2$ input pairs, so $\approx 2^{79}n^2$
chances for a collision in H.

$(H(r)), \ldots$

ng

0 zero bits.

point.")

(r).

not exist.

s.

ypically

uts to H.

Choose random $r_1, r_2, \ldots, r_n$.

Compute $Z(r_1), Z(r_2), \ldots, Z(r_n)$.

Uses $\approx 2^{40} n$ inputs to H:

$r_1, H(r_1), H^2(r_1), H^3(r_1), \ldots$

$r_2, H(r_2), H^2(r_2), H^3(r_2), \ldots$

$\vdots$

$r_n, H(r_n), H^2(r_n), H^3(r_n), \ldots$

"Birthday paradox":

$\approx 2^{79} n^2$ input pairs, so $\approx 2^{79} n^2$

chances for a collision in H.

Say there's a collis

$H^{161}(r_2) = H^{190}($

$Z(r_2)$ is after H^{16}

$Z(r_7)$ is after H^{19}

and $H^{160}(r_2) \neq H$

Then $Z(r_2) = Z($

Recognize this by

$Z(r_1), Z(r_2), \ldots,$

and comparing adj

Backtrack to find

Oops, may have m

backtracking can b

But usually not a

Choose random $r_1, r_2, \ldots, r_n$.
Compute $Z(r_1), Z(r_2), \ldots, Z(r_n)$.

Uses $\approx 2^{40} n$ inputs to H:

$r_1, H(r_1), H^2(r_1), H^3(r_1), \ldots$
$r_2, H(r_2), H^2(r_2), H^3(r_2), \ldots$
$\vdots$
$r_n, H(r_n), H^2(r_n), H^3(r_n), \ldots$

"Birthday paradox": $\approx 2^{79} n^2$ input pairs, so $\approx 2^{79} n^2$ chances for a collision in H.

Say there's a collision: e.g., $H^{161}(r_2) = H^{190}(r_7)$ where $Z(r_2)$ is after $H^{161}(r_2)$, $Z(r_7)$ is after $H^{190}(r_7)$, and $H^{160}(r_2) \neq H^{189}(r_7)$.

Then $Z(r_2) = Z(r_7)$. Recognize this by sorting $Z(r_1), Z(r_2), \ldots, Z(r_n)$ and comparing adjacent outputs. Backtrack to find collision.

Oops, may have multiple collisions; backtracking can be expensive. But usually not a problem.

$, r_2, \ldots, r_n$.

$Z(r_2), \ldots, Z(r_n)$.

ts to H:

$H^3(r_1), \ldots$

$H^3(r_2), \ldots$

$, H^3(r_n), \ldots$

":

rs, so $\approx 2^{79} n^2$

sion in H.

Say there's a collision: e.g.,
$H^{161}(r_2) = H^{190}(r_7)$ where
$Z(r_2)$ is after $H^{161}(r_2)$,
$Z(r_7)$ is after $H^{190}(r_7)$,
and $H^{160}(r_2) \neq H^{189}(r_7)$.

Then $Z(r_2) = Z(r_7)$.
Recognize this by sorting
$Z(r_1), Z(r_2), \ldots, Z(r_n)$
and comparing adjacent outputs.
Backtrack to find collision.

Oops, may have multiple collisions;
backtracking can be expensive.
But usually not a problem.

Serial computer:
$\approx 2^{40} n$ evaluation
$\approx n \log n$ sorting s
$\approx 256n$ bits of RA

2-dimensional mes
with n parallel pro
$\approx 2^{40}$ evaluations
$\approx 8\sqrt{n}$ sorting ste
$\approx n$ small cells.

Mesh computer is
about n times fast
not much more ex

Say there's a collision: e.g., $H^{161}(r_2) = H^{190}(r_7)$ where $Z(r_2)$ is after $H^{161}(r_2)$, $Z(r_7)$ is after $H^{190}(r_7)$, and $H^{160}(r_2) \neq H^{189}(r_7)$.

Then $Z(r_2) = Z(r_7)$. Recognize this by sorting $Z(r_1), Z(r_2), \ldots, Z(r_n)$ and comparing adjacent outputs. Backtrack to find collision.

Oops, may have multiple collisions; backtracking can be expensive. But usually not a problem.

Serial computer:
$\approx 2^{40}n$ evaluations of H;
$\approx n \log n$ sorting steps;
$\approx 256n$ bits of RAM.

2-dimensional mesh computer with n parallel processors:
$\approx 2^{40}$ evaluations of H;
$\approx 8\sqrt{n}$ sorting steps;
$\approx n$ small cells.

Mesh computer is about n times faster, not much more expensive.

sion: e.g.,

$r_7)$ where

$^1(r_2)$,

$^0(r_7)$,

$^{189}(r_7)$.

$r_7)$.

sorting

$Z(r_n)$

jacent outputs.

collision.

nultiple collisions;

be expensive.

problem.

Serial computer:
$\approx 2^{40}n$ evaluations of H;
$\approx n \log n$ sorting steps;
$\approx 256n$ bits of RAM.

2-dimensional mesh computer
with n parallel processors:
$\approx 2^{40}$ evaluations of H;
$\approx 8\sqrt{n}$ sorting steps;
$\approx n$ small cells.

Mesh computer is
about n times faster,
not much more expensive.

Using collision sea
for "discrete logari

Want to figure out
P, kP on an ellipti

Define $H(x, y) =$
Find collision in H
usually reveals k.

Price-performance
$q^{1/2+o(1)}$ dollar-se
if curve has q poin
Fancier methods, s
rho, kangaroo, etc

Serial computer:
$\approx 2^{40} n$ evaluations of H;
$\approx n \log n$ sorting steps;
$\approx 256n$ bits of RAM.

2-dimensional mesh computer with n parallel processors:
$\approx 2^{40}$ evaluations of H;
$\approx 8\sqrt{n}$ sorting steps;
$\approx n$ small cells.

Mesh computer is about n times faster, not much more expensive.

Using collision search for "discrete logarithms":

Want to figure out k given P, kP on an elliptic curve.

Define $H(x, y) = xP + ykP$.
Find collision in H; usually reveals k.

Price-performance ratio:
$q^{1/2+o(1)}$ dollar-seconds if curve has q points.
Fancier methods, same $1/2$: rho, kangaroo, etc.