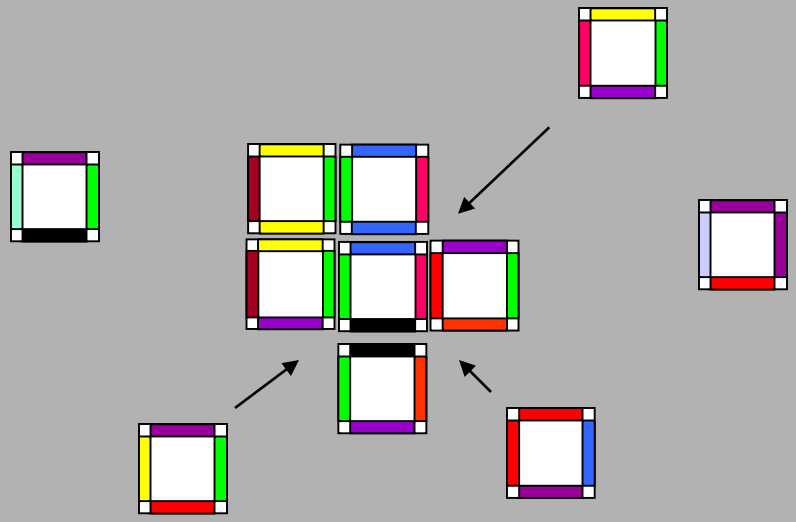
Fast Arithmetic in Algorithmic Self-Assembly

Unconventional Computation & Natural Computation

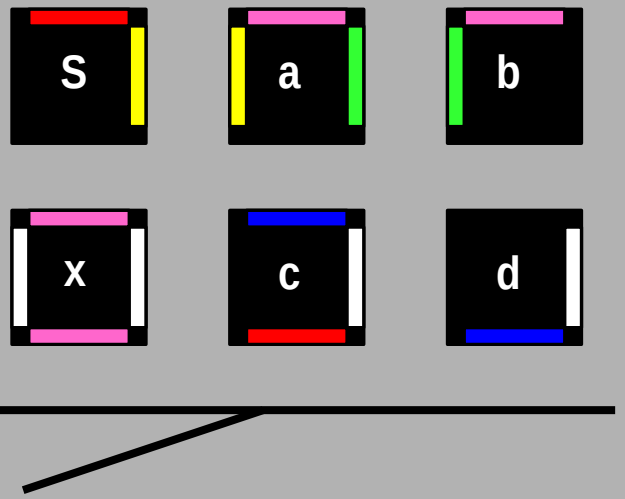
July 14, 2014



Alexandra Keenan — Wyle Science Technology and Engineering Group
Robert Schweller — University of Texas - Pan American
Michael Sherman — Amazon
Xingsi Zhong — Clemson University

How a tile system self assembles

T =



Tile Set:

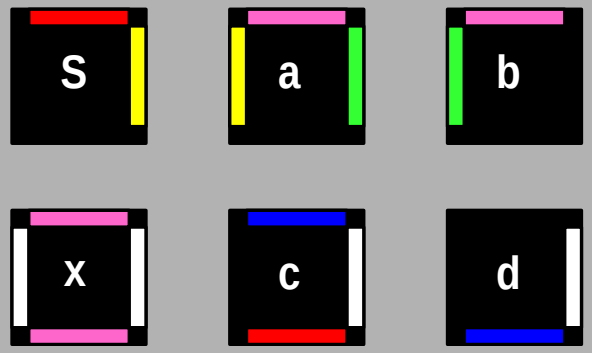
Glue Function:

G(y)=2
G(g)=2
G(r)=2
G(b)=2
G(p)=1
G(w)=1

Temperature: t = 2

Seed:

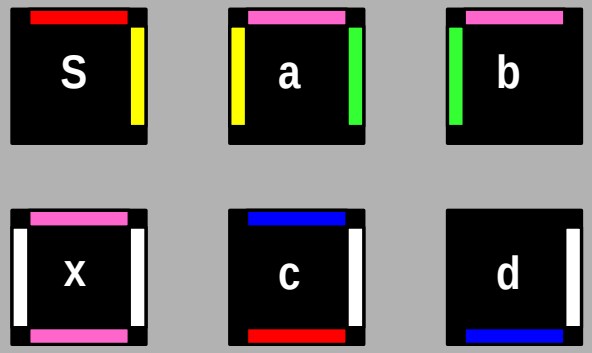
How a tile system self assembles

T = 

G(y)=2
G(g)=2
G(r)=2
G(b)=2
G(p)=1
G(w)=1

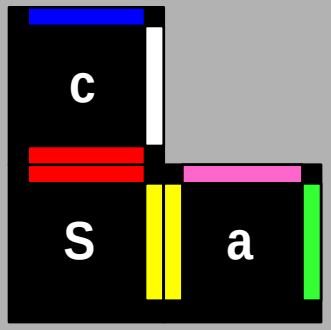
t = 2

How a tile system self assembles

T = 

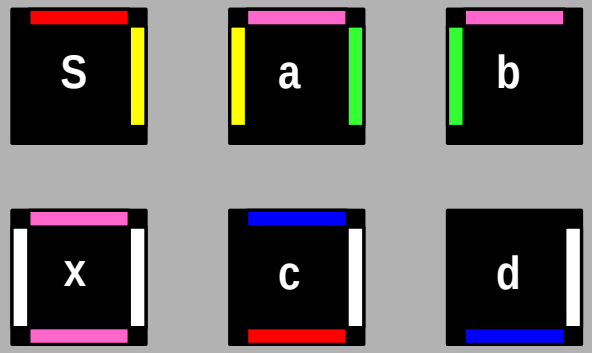
G(y)=2
G(g)=2
G(r)=2
G(b)=2
G(p)=1
G(w)=1

t = 2



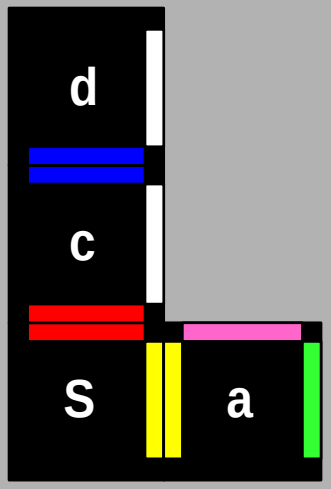
How a tile system self assembles

T =



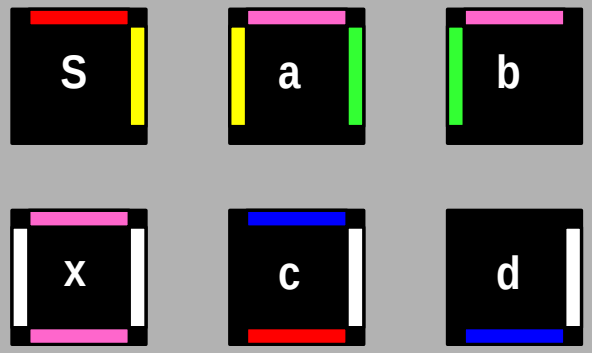
G(y)=2
G(g)=2
G(r)=2
G(b)=2
G(p)=1
G(w)=1

t = 2



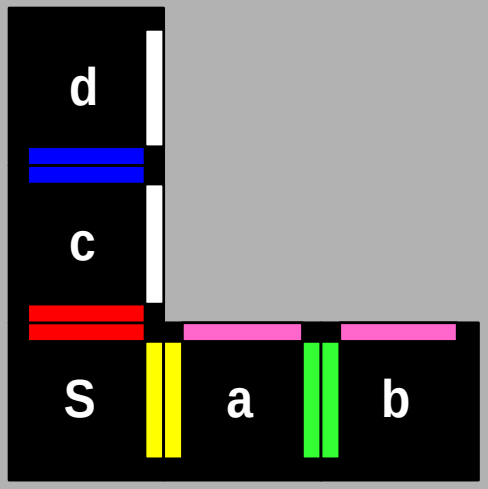
How a tile system self assembles

T =



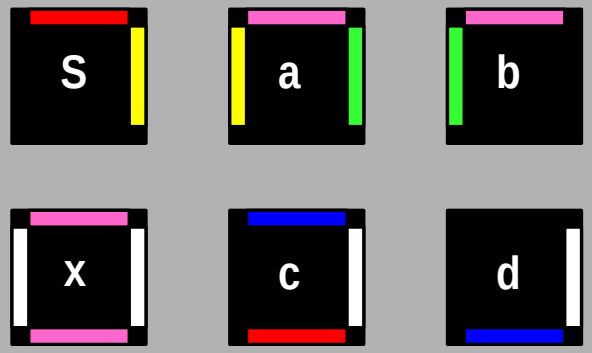
G(y)=2
G(g)=2
G(r)=2
G(b)=2
G(p)=1
G(w)=1

t = 2



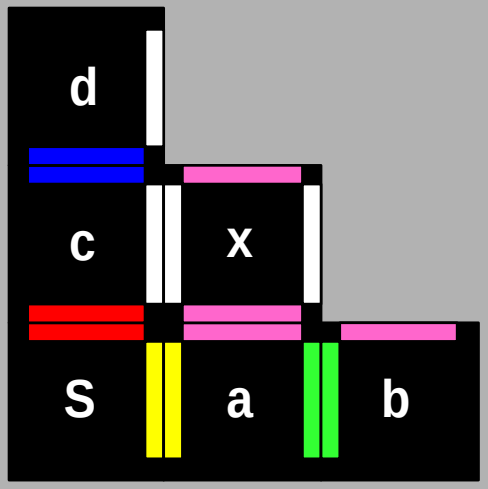
How a tile system self assembles

T =



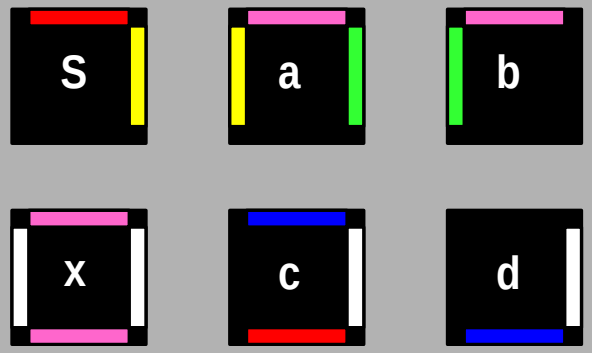
G(y)=2
G(g)=2
G(r)=2
G(b)=2
G(p)=1
G(w)=1

t = 2



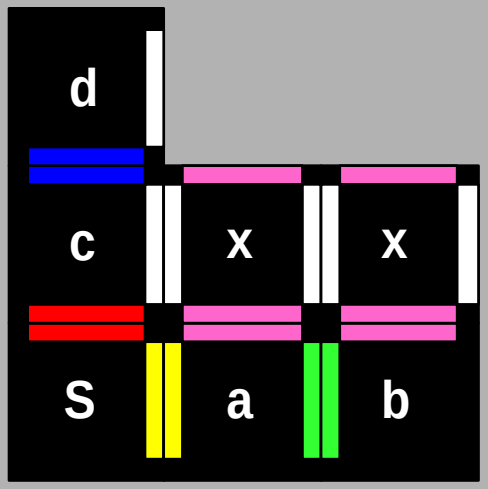
How a tile system self assembles

T =



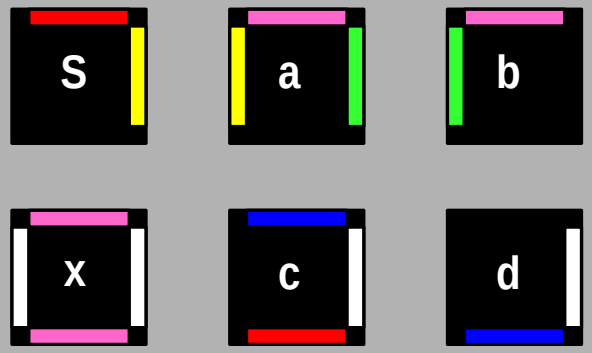
G(y)=2
G(g)=2
G(r)=2
G(b)=2
G(p)=1
G(w)=1

t = 2



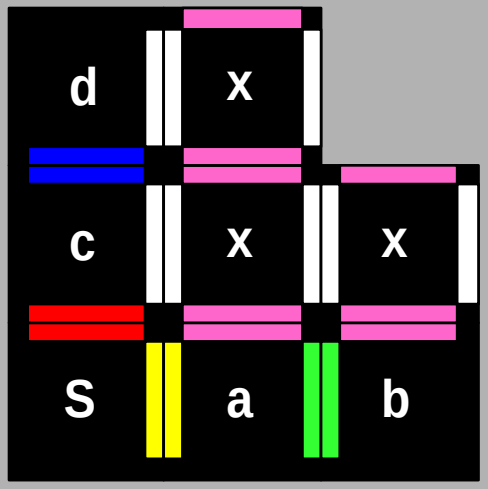
How a tile system self assembles

T =



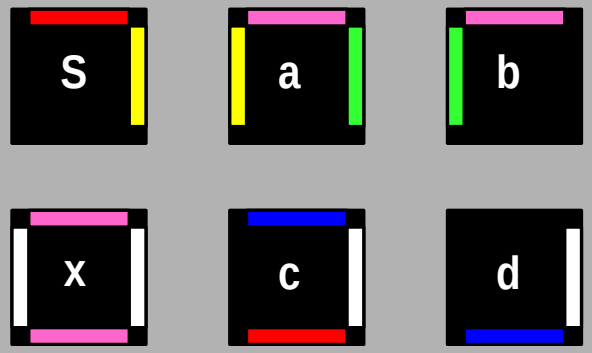
G(y)=2
G(g)=2
G(r)=2
G(b)=2
G(p)=1
G(w)=1

t = 2

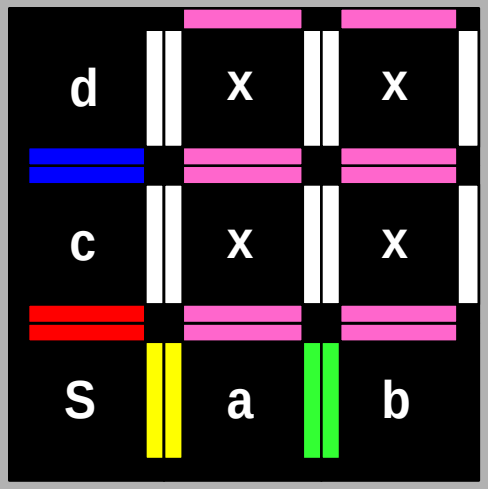


How a tile system self assembles

T =



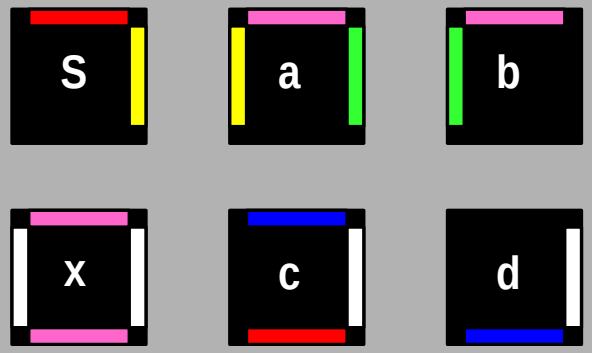
G(y)=2
G(g)=2
G(r)=2
G(b)=2
G(p)=1
G(w)=1



t = 2

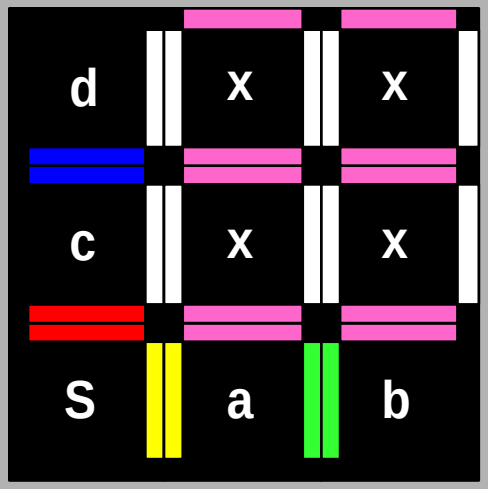
How a tile system self assembles

T =



G(y)=2
G(g)=2
G(r)=2
G(b)=2
G(p)=1
G(w)=1

t = 2



What is this model capable above?

-efficient assembly of shapes/patterns

-shape and pattern replication

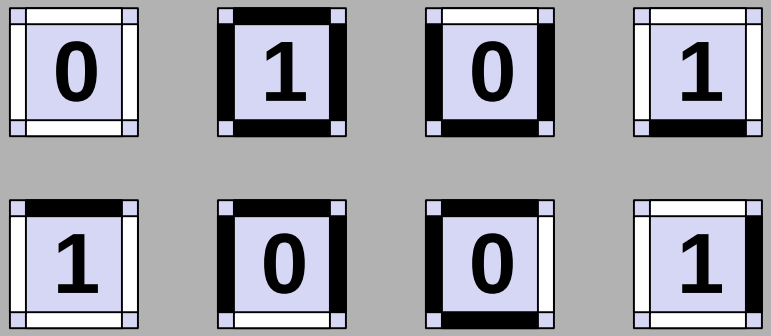
-computation

-fast arithmetic

Binary Addition

(Yuriy Brun, TCS 2007)

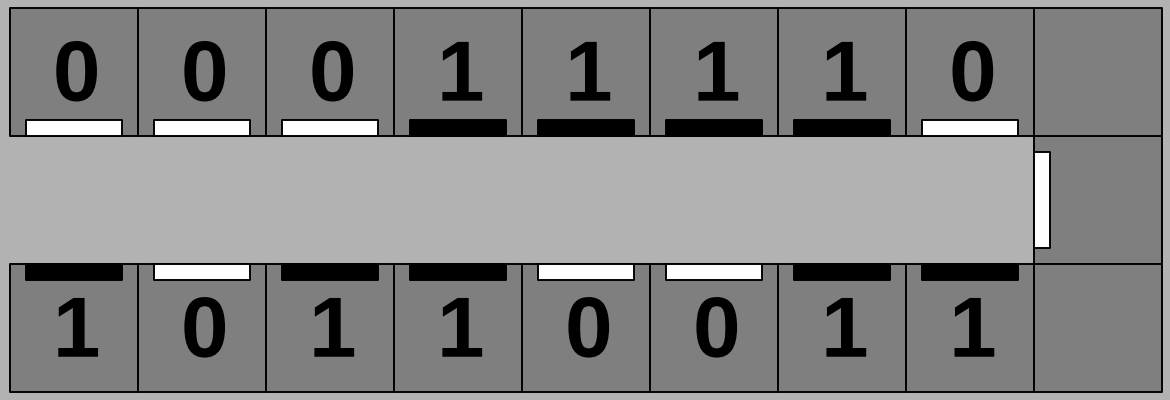
Tile set :



Compute:

```
  00011110
+ 10110011
----------
```

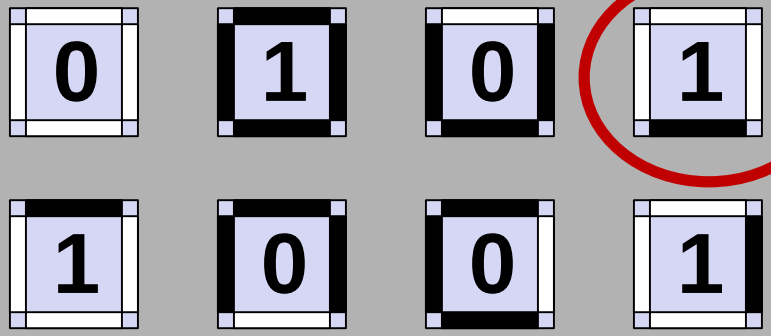
Seed :



Binary Addition

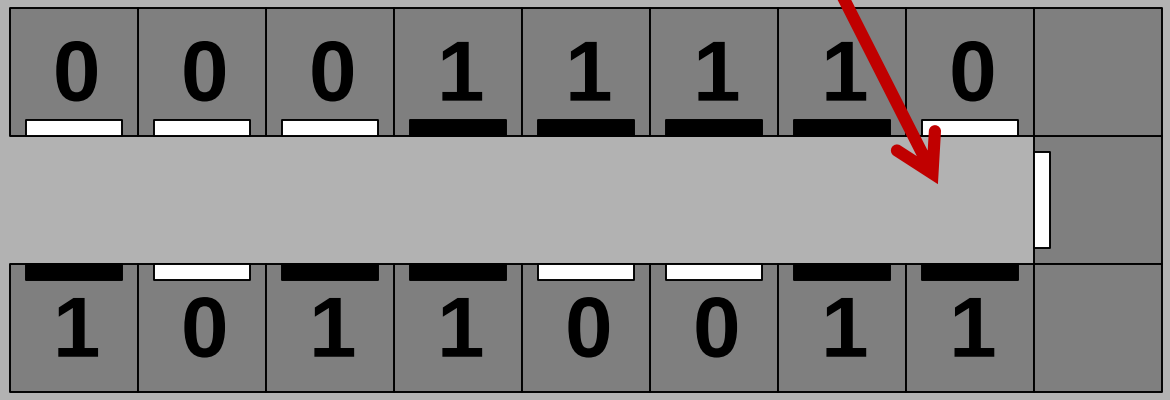
(Yuriy Brun, TCS 2007)

Tile set :




```
  00011110
+ 10110011
----------
```

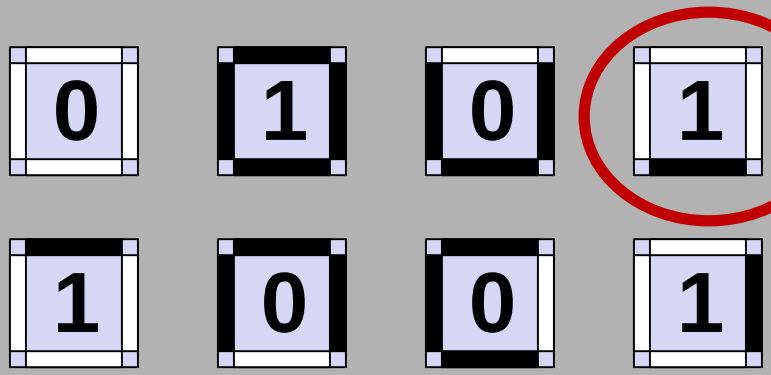
Seed :



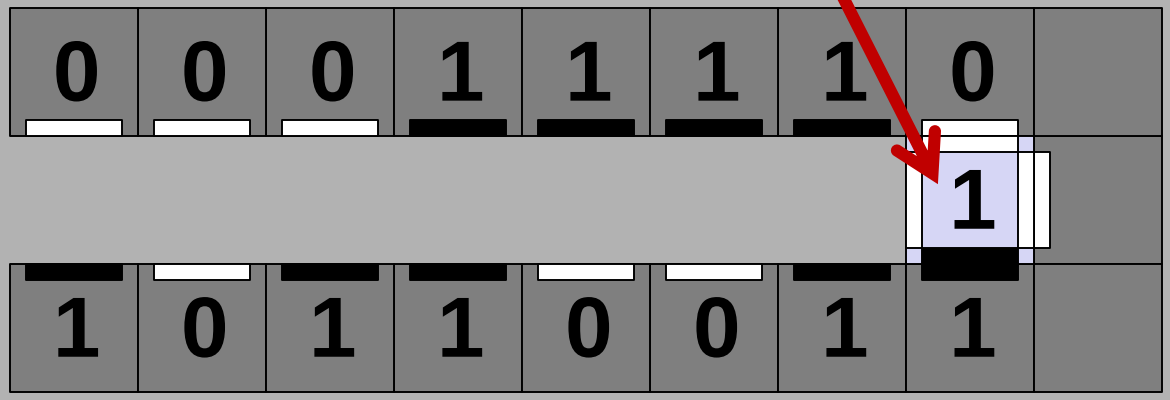
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :



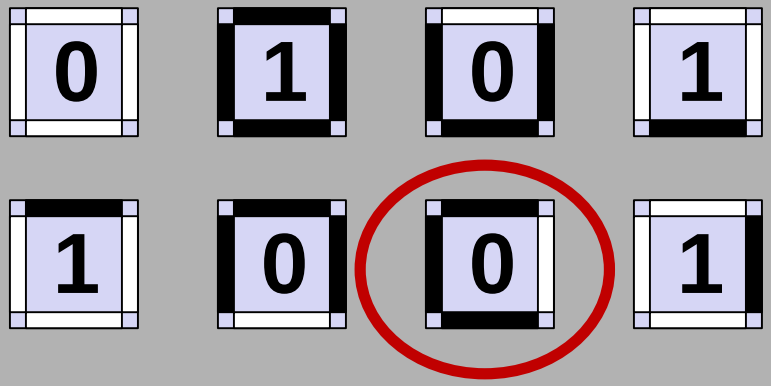

```
  00011110
+ 10110011
----------
         1
```



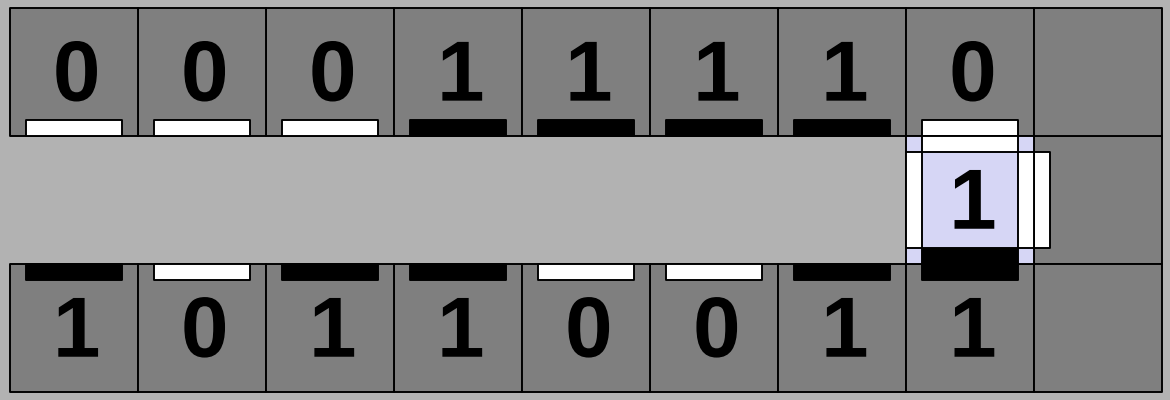
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :



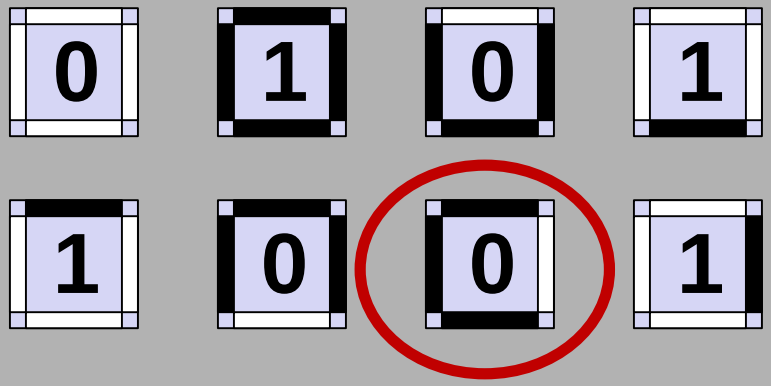

```
  00011110
+ 10110011
----------
         1
```



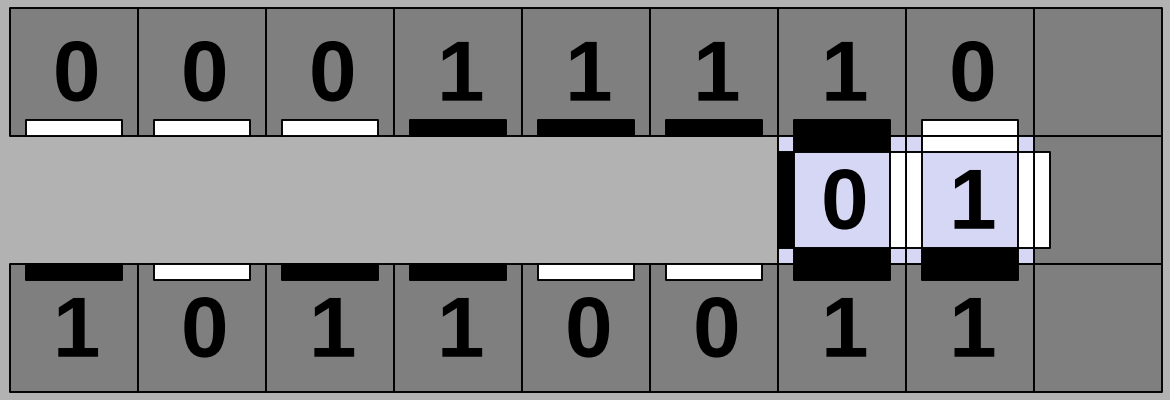
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :



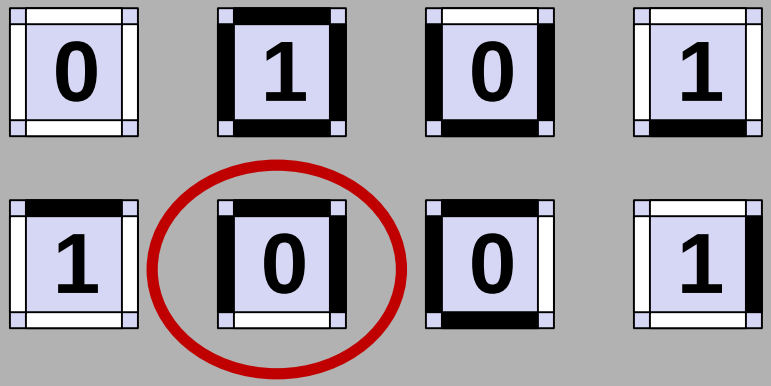

```
      1
  00011110
+ 10110011
----------
        01
```



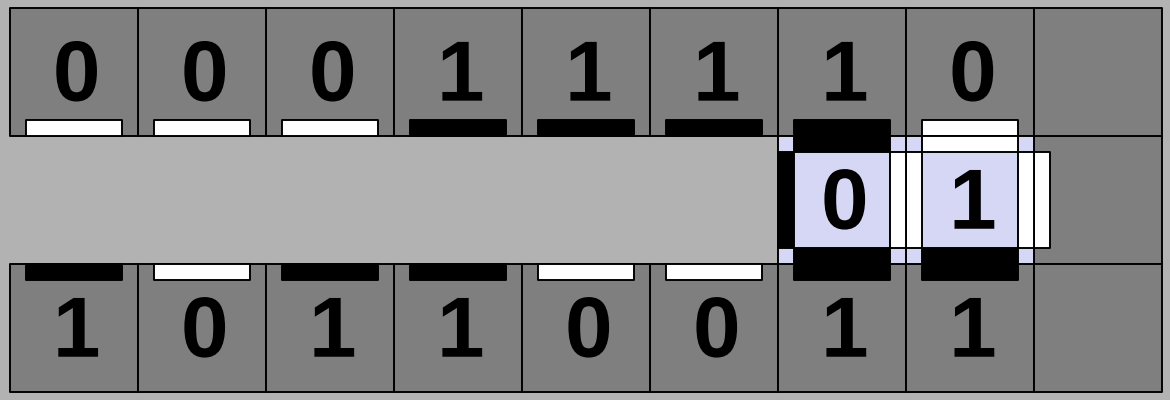
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :



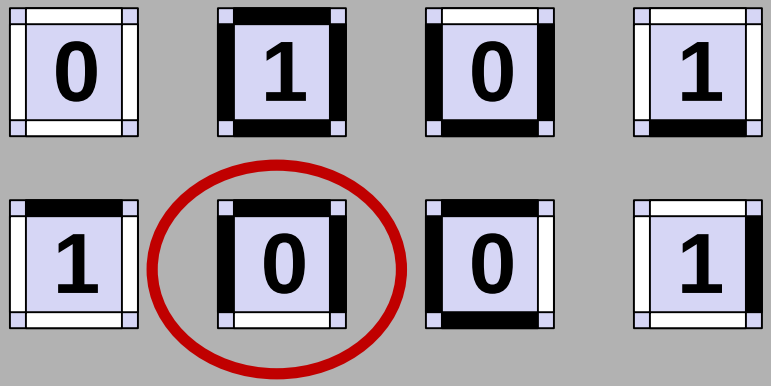

```
      1
  00011110
+ 10110011
----------
        01
```



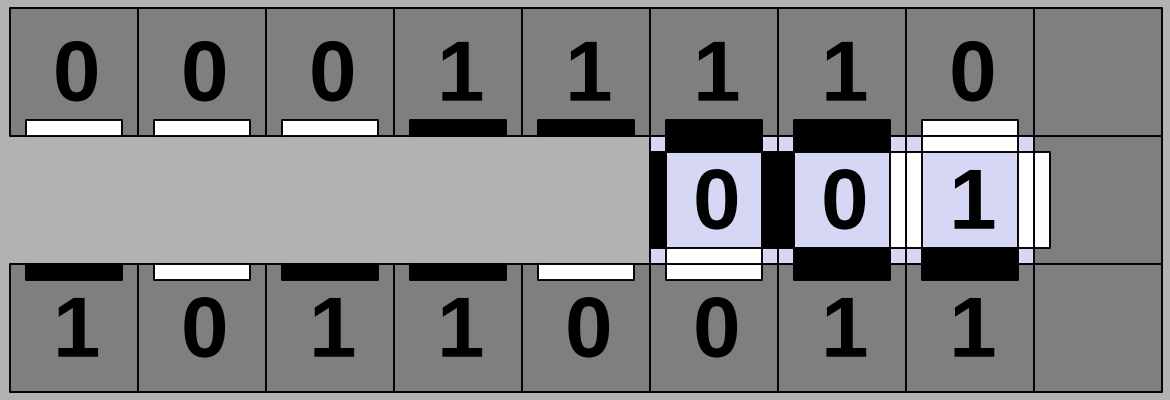
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :



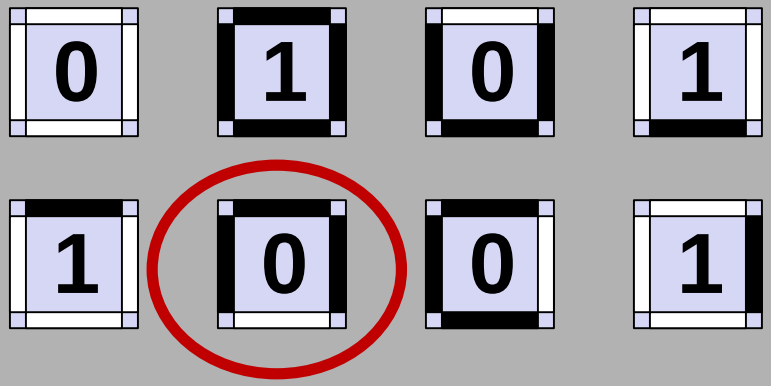

```
      11
  00011110
+ 10110011
----------
      001
```



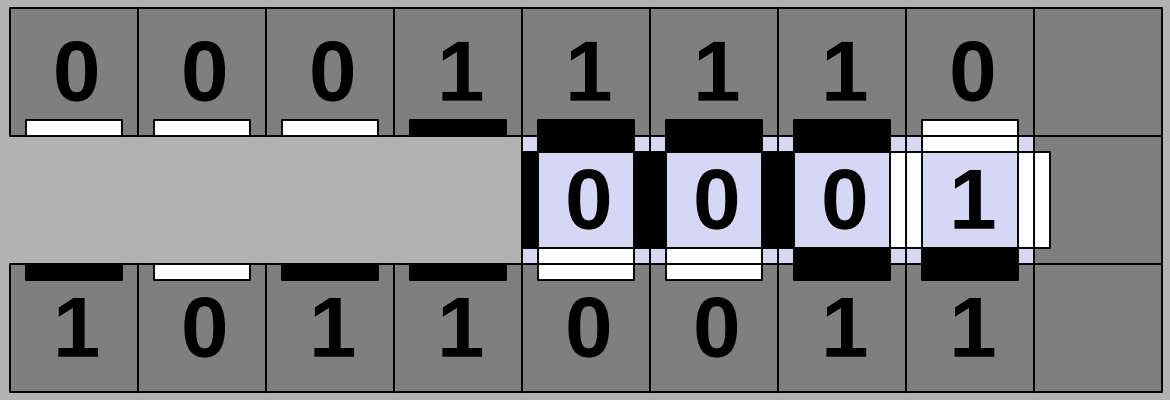
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :



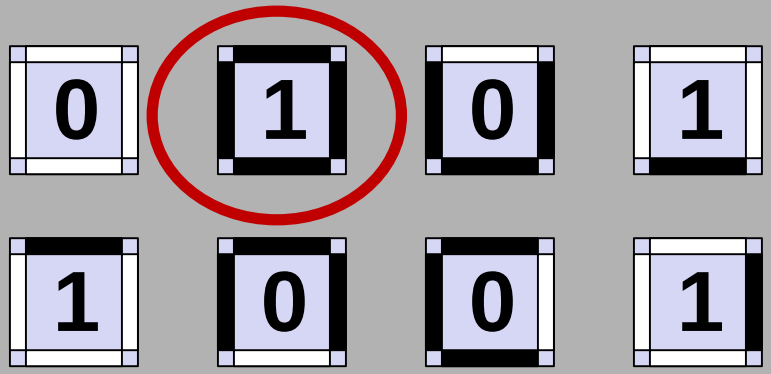

```
   111
  00011110
+ 10110011
----------
      0001
```



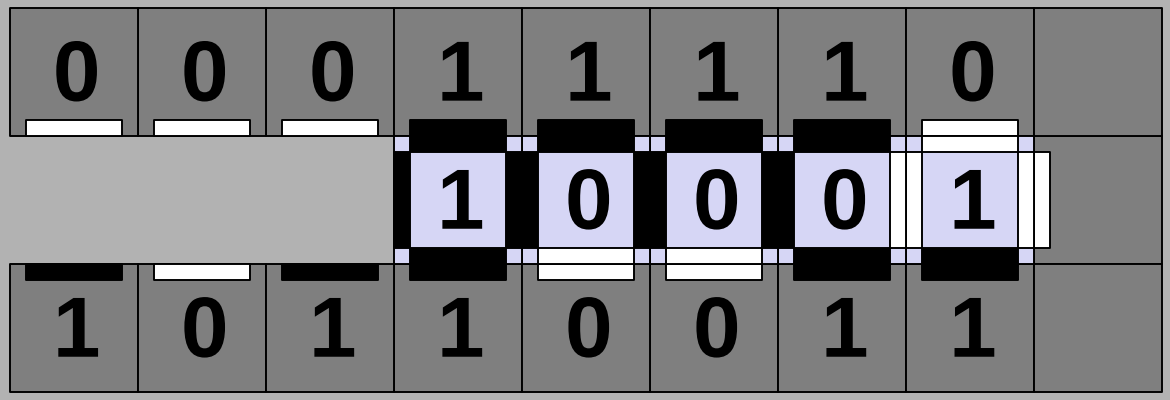
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :



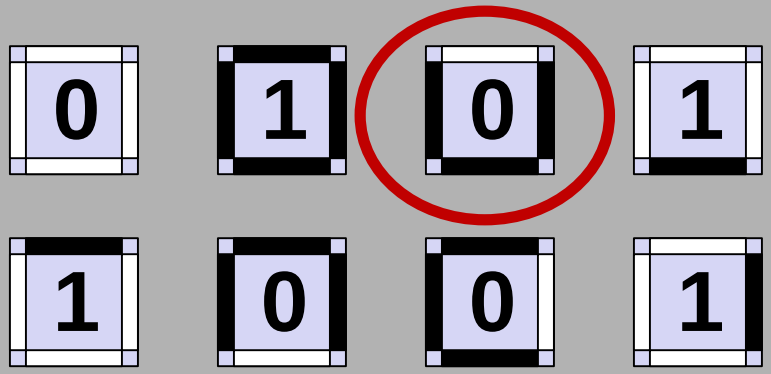

```
   1111
  00011110
+ 10110011
----------
  10001
```



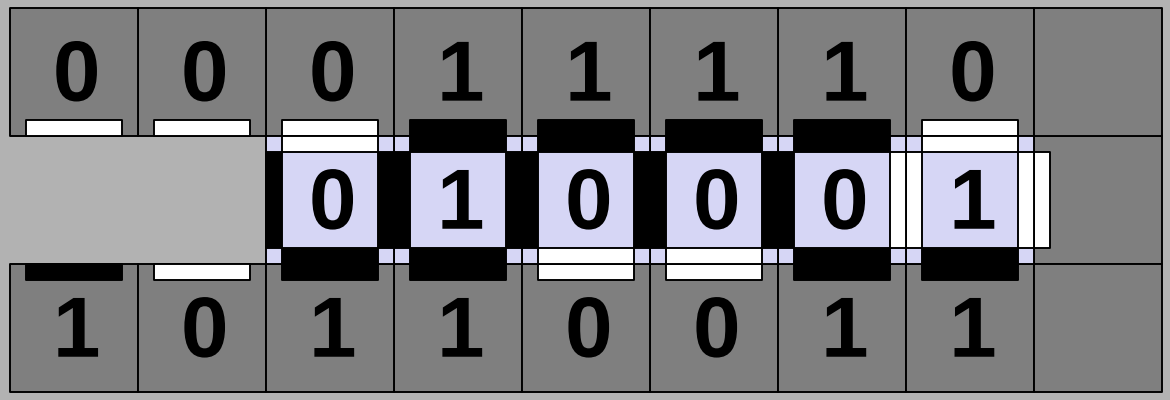
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :



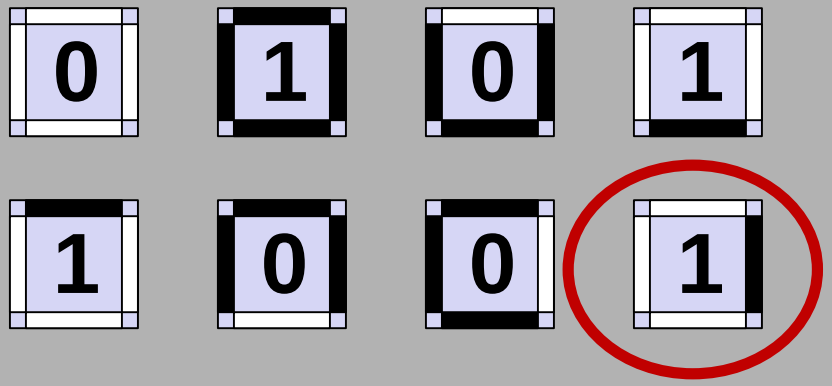

```
  11111
  00011110
+ 10110011
-----------
    010001
```



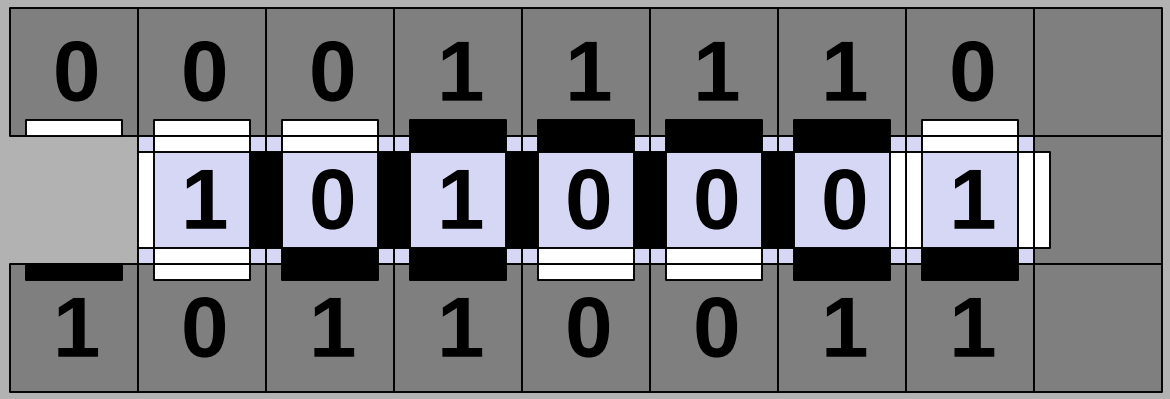
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :



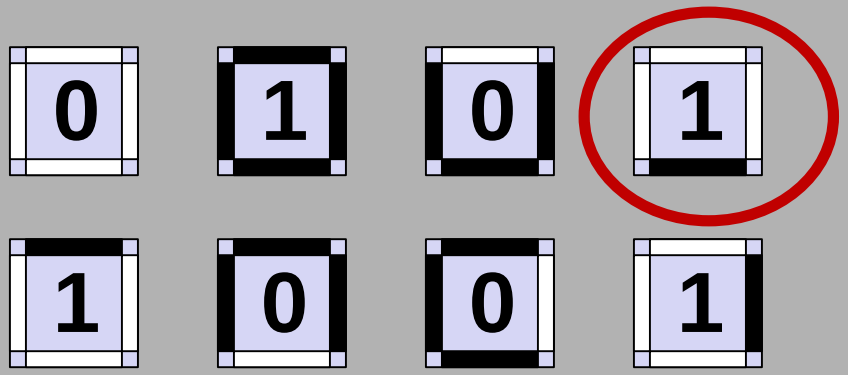

```
  11111
  00011110
+ 10110011
----------
  1010001
```



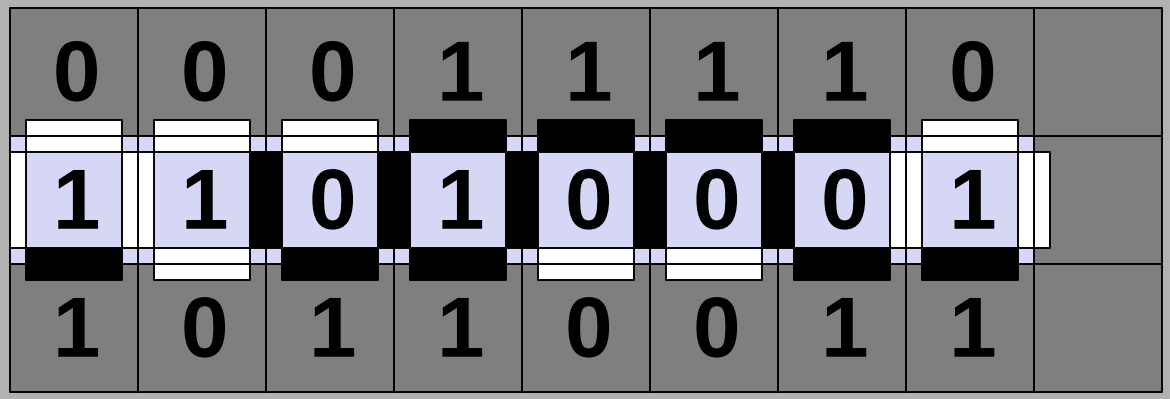
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :



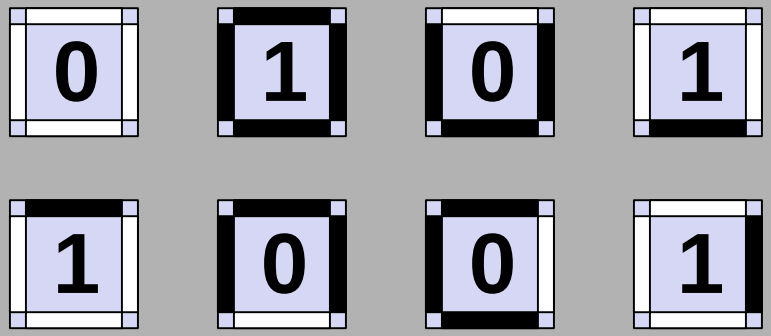

```
  11111
  00011110
+ 10110011
----------
  11010001
```



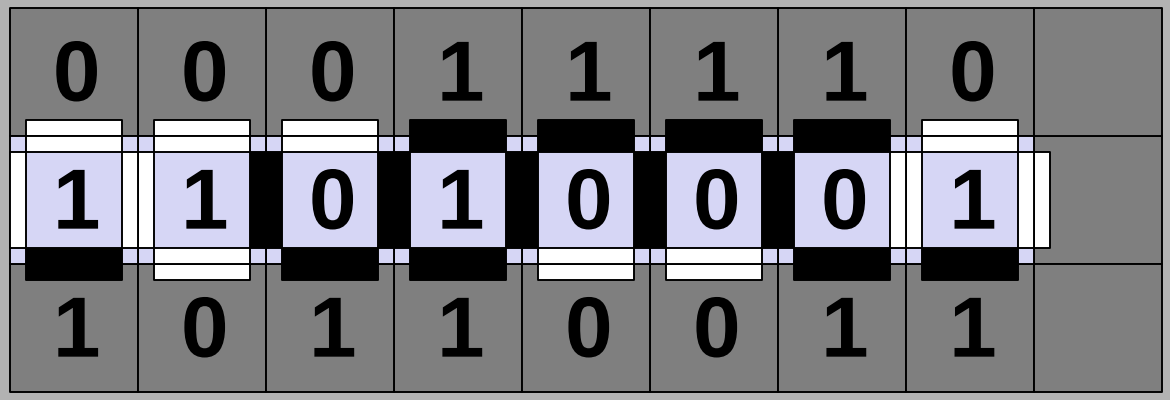
Binary Addition

(Yuriy Brun, TCS 2007)

Tile set :




```
  11111
  00011110
+ 10110011
----------
  11010001
```



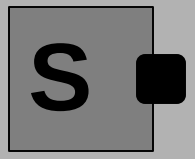
Run time: O(n)

¿Are there faster algorithms?

Run Time Model

Build a 3x3 square

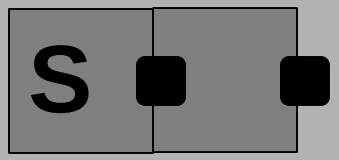
Time steps: 0



Run Time Model

Build a 3x3 square

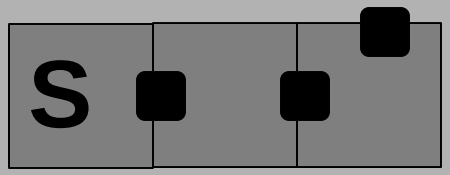
Time steps: 1



Run Time Model

Build a 3x3 square

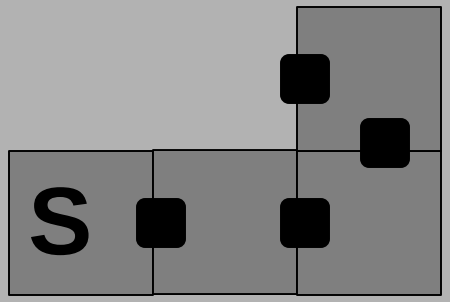
Time steps: 2



Run Time Model

Build a 3x3 square

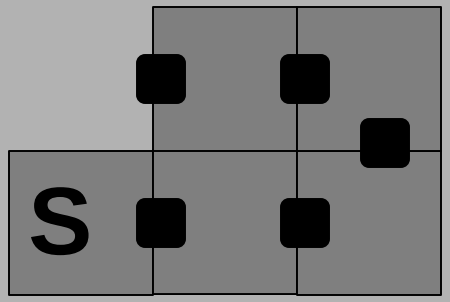
Time steps: 3



Run Time Model

Build a 3x3 square

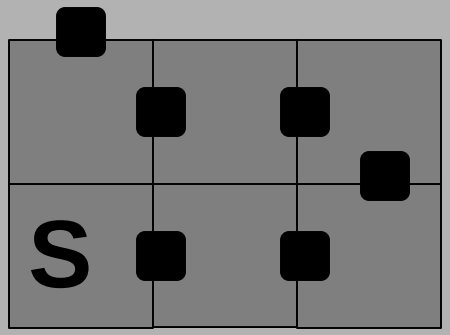
Time steps: 4



Run Time Model

Build a 3x3 square

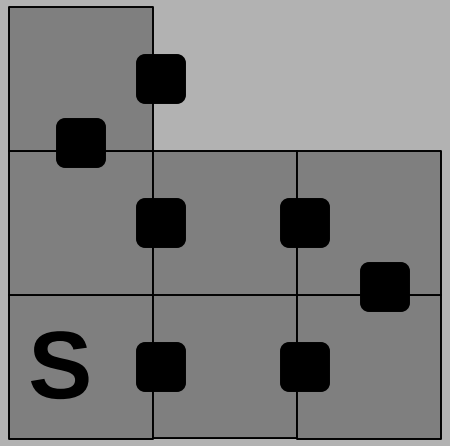
Time steps: 5



Run Time Model

Build a 3x3 square

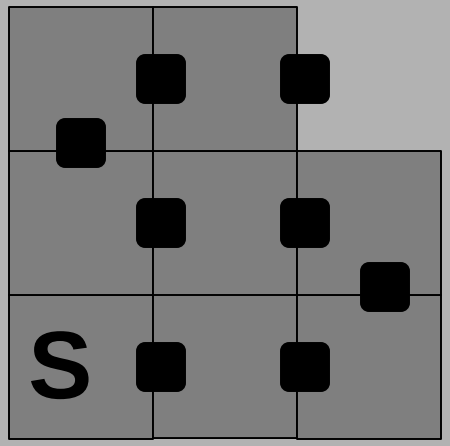
Time steps: 6



Run Time Model

Build a 3x3 square

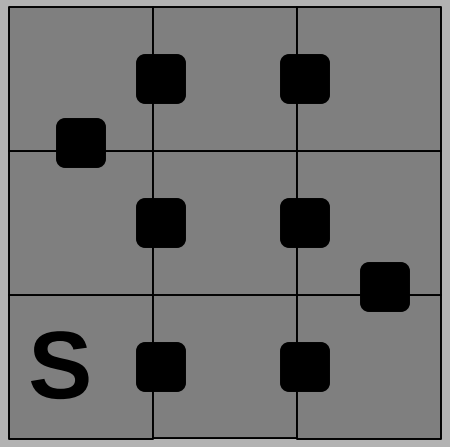
Time steps: 7



Run Time Model

Build a 3x3 square

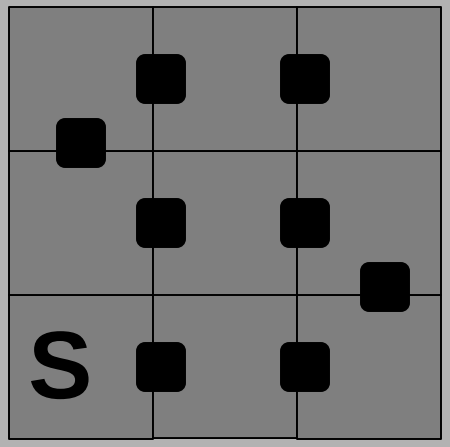
Time steps: 8



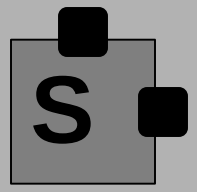
Run Time Model

Build a 3x3 square

Time steps: 8



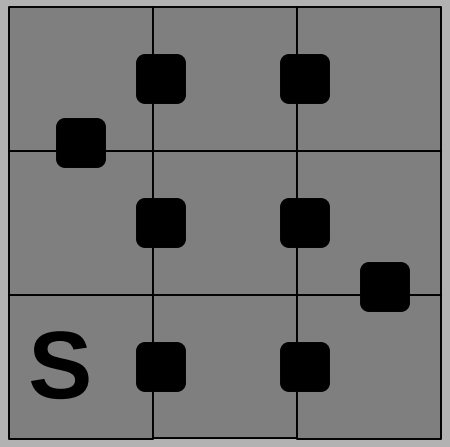
Time steps: 0



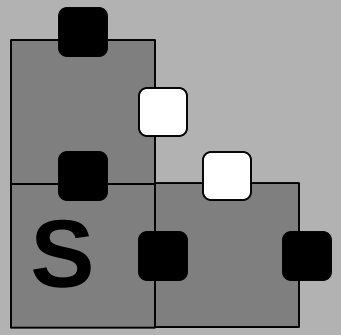
Run Time Model

Build a 3x3 square

Time steps: 8



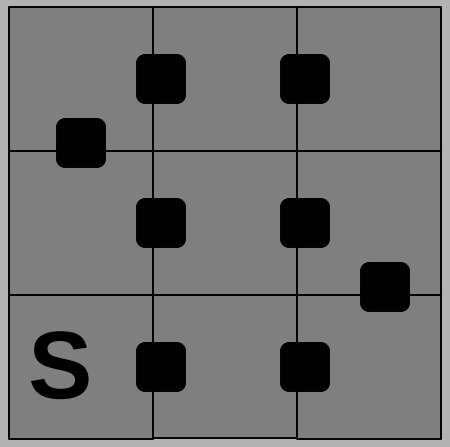
Time steps: 1



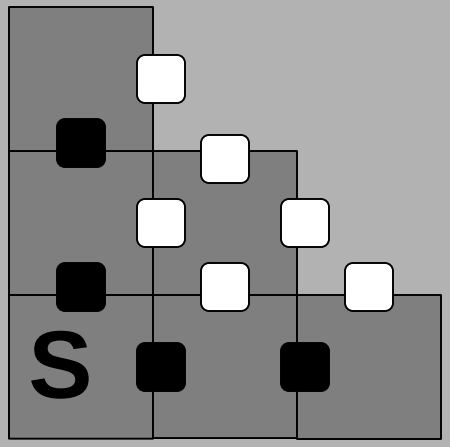
Run Time Model

Build a 3x3 square

Time steps: 8



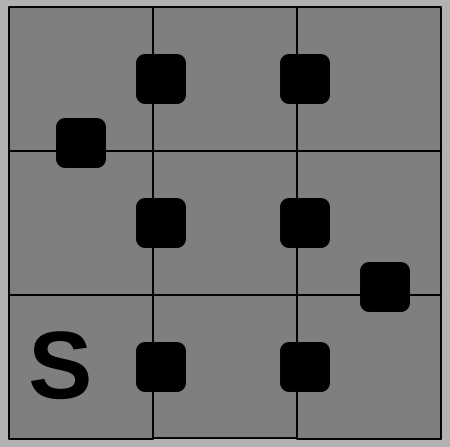
Time steps: 2



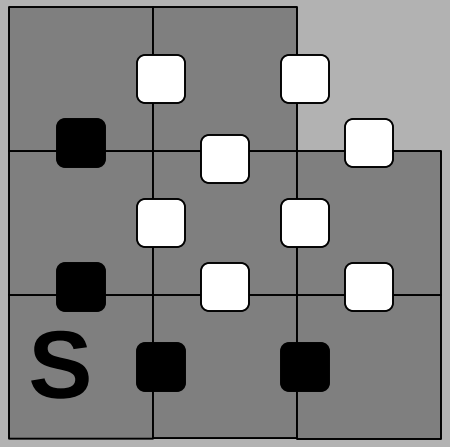
Run Time Model

Build a 3x3 square

Time steps: 8



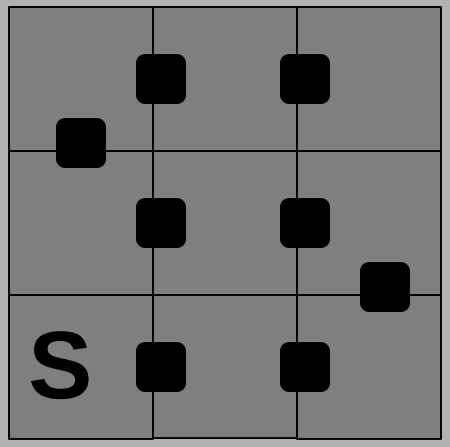
Time steps: 3



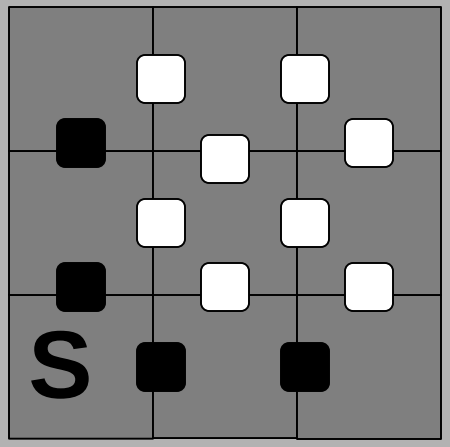
Run Time Model

Build a 3x3 square

Time steps: 8

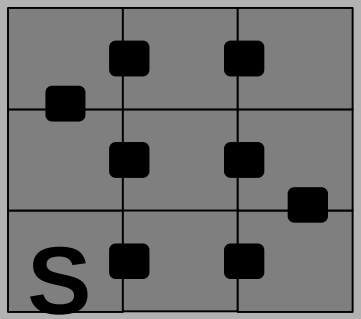


Time steps: 4

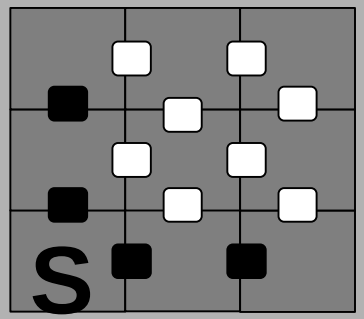


Run Time Model

Time steps: 8



Time steps: 4



Parallel timing model: [Becker et. al., 2006]
All attachable tiles attach in 1 time step.

- Simple

Continuous timing model: [Adleman et. al., 2001]
Assembly is modelled as a continuous time Markov process.

- More realistic

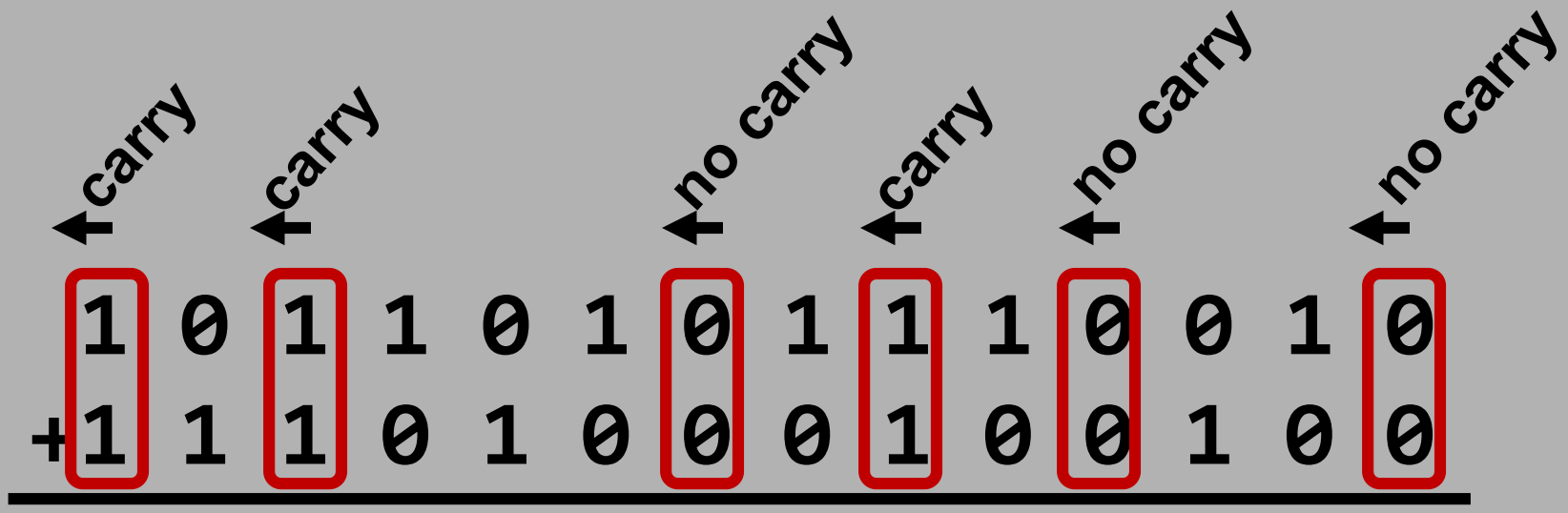
Our upper bounds hold within both models

	Addition Algorithms	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm		
Worst-case algorithm		
Combo algorithm		
3D		

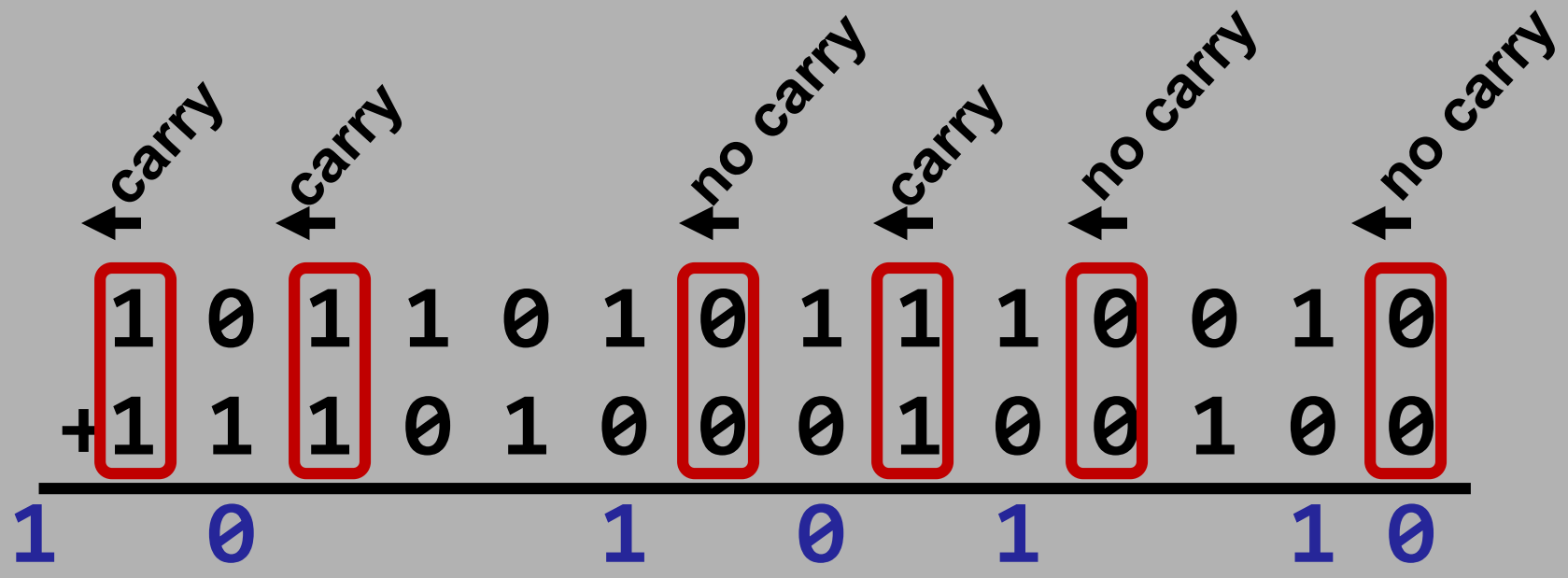
Addition Parallelization Scheme: average case

```
  1 0 1 1 0 1 0 1 1 1 0 0 1 0
+1 1 1 0 1 0 0 0 1 0 0 1 0 0
```

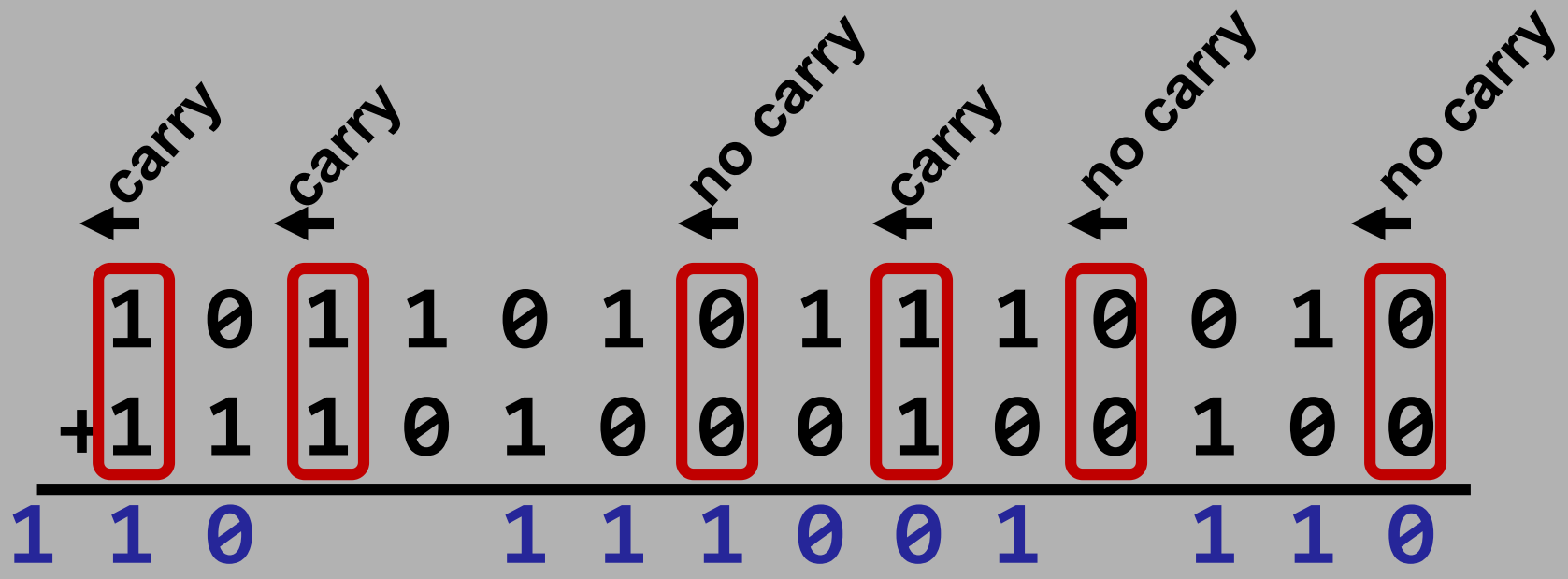
Addition Parallelization Scheme: average case



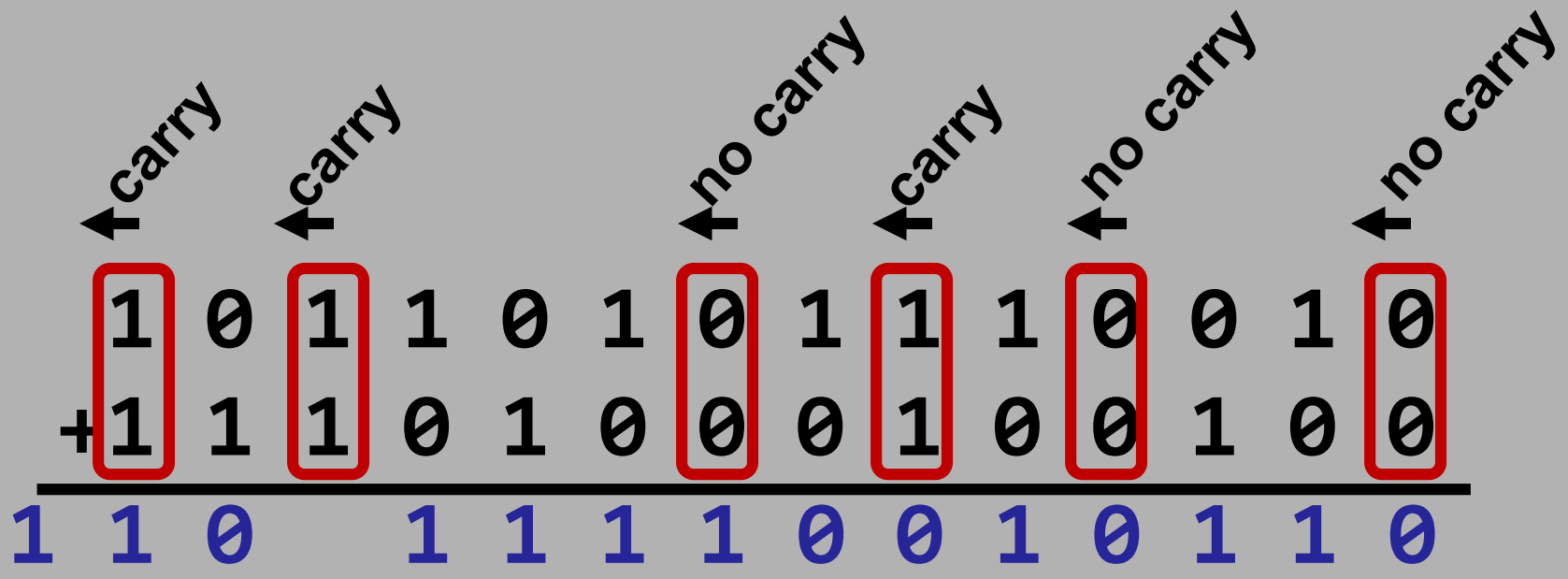
Addition Parallelization Scheme: average case



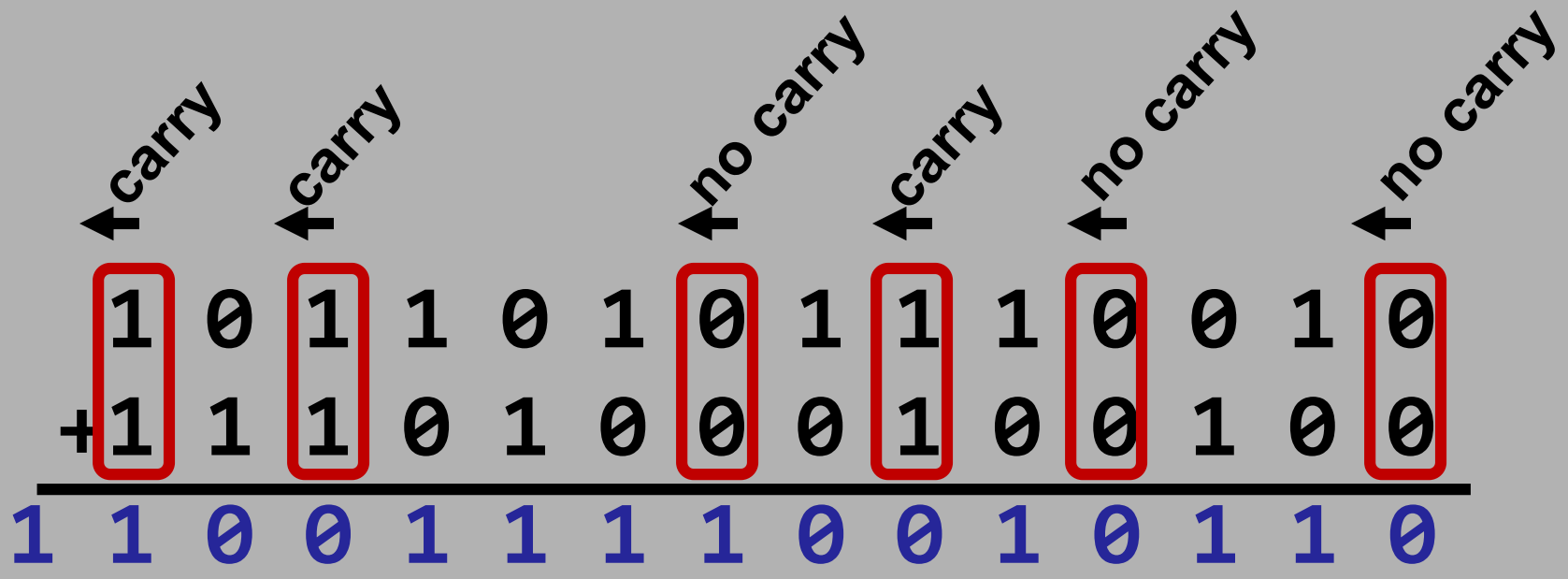
Addition Parallelization Scheme: average case



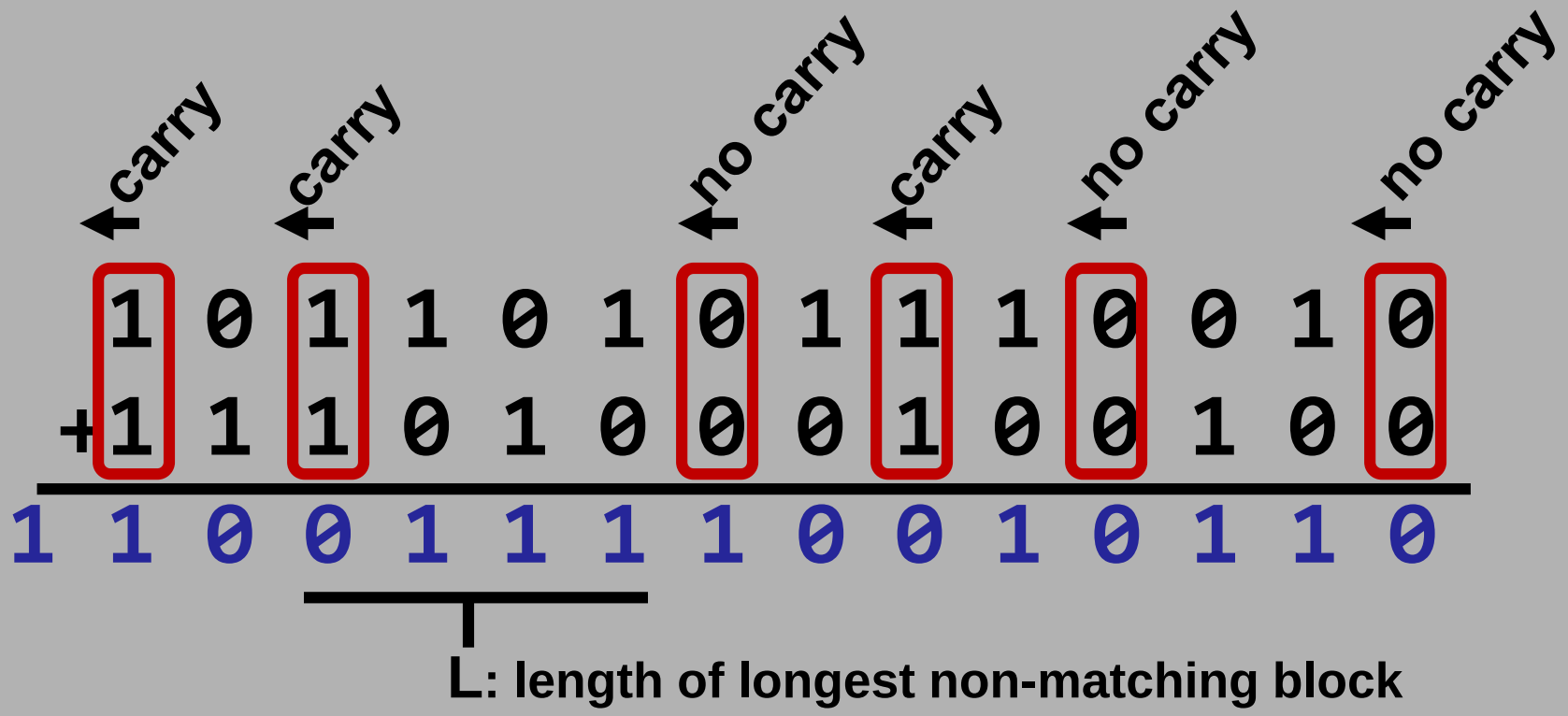
Addition Parallelization Scheme: average case



Addition Parallelization Scheme: average case



Addition Parallelization Scheme: average case



Run time: O(L)

Average run time: O(log n)

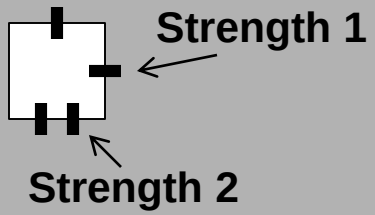
Challenge:
Can we implement this parallelization scheme within a tile system?

Addition

Faster average case

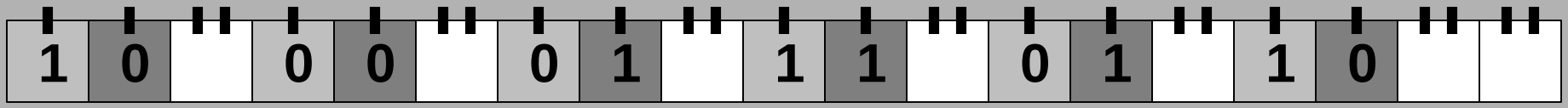
Temperature: 2

Notation:




```
  100101
+ 001110
--------
```

Seed input:

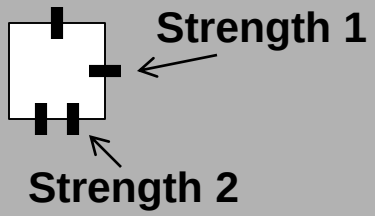


Addition

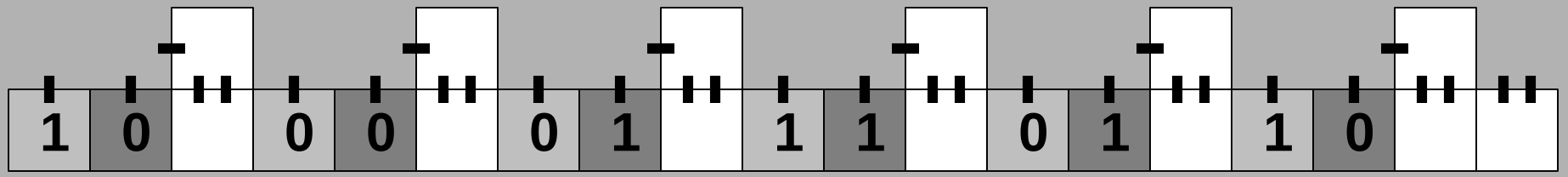
Faster average case

Temperature: 2

Notation:




```
  100101
+ 001110
--------
```

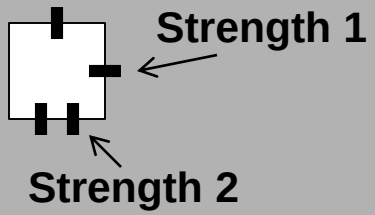


Addition

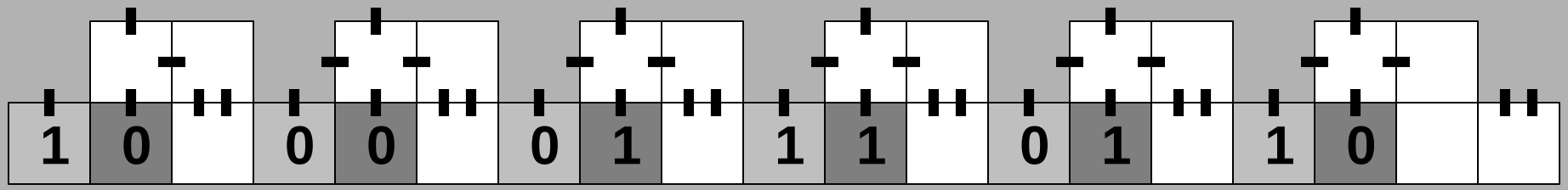
Faster average case

Temperature: 2

Notation:




```
  100101
+ 001110
--------
```

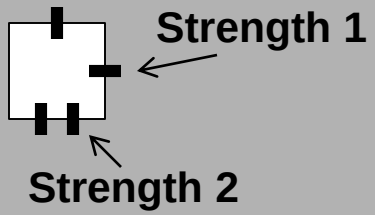


Addition

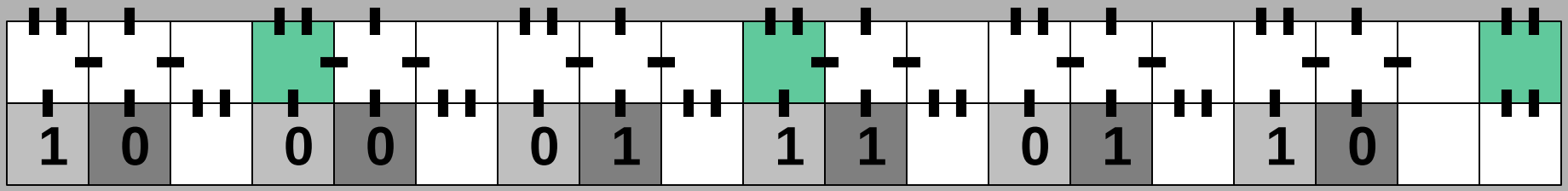
Faster average case

Temperature: 2

Notation:




```
  100101
+ 001110
--------
```

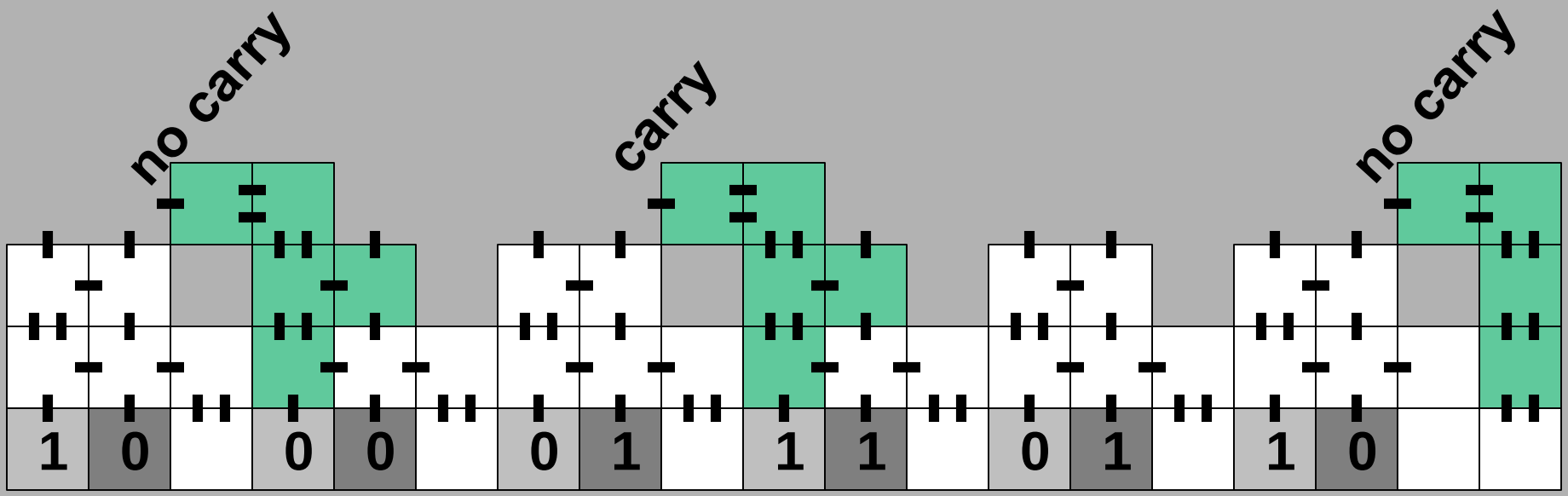


Addition

Faster average case

Temperature: 2

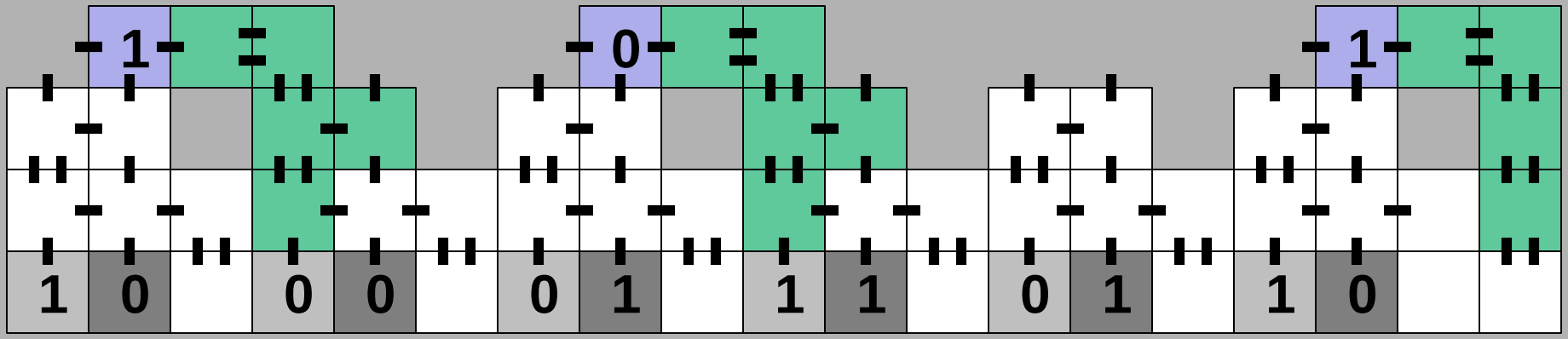
```
  100101
+ 001110
--------
```



Addition

Faster average case

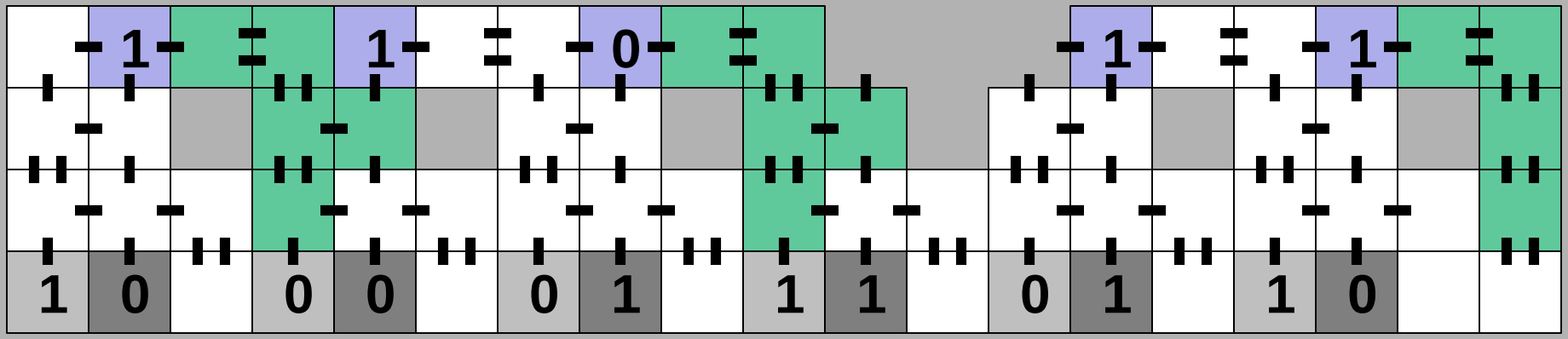
$$\begin{array}{r} 100101 \\ +\ 001110 \\ \hline 1\ 0\ \ 1 \end{array}$$



Addition

Faster average case

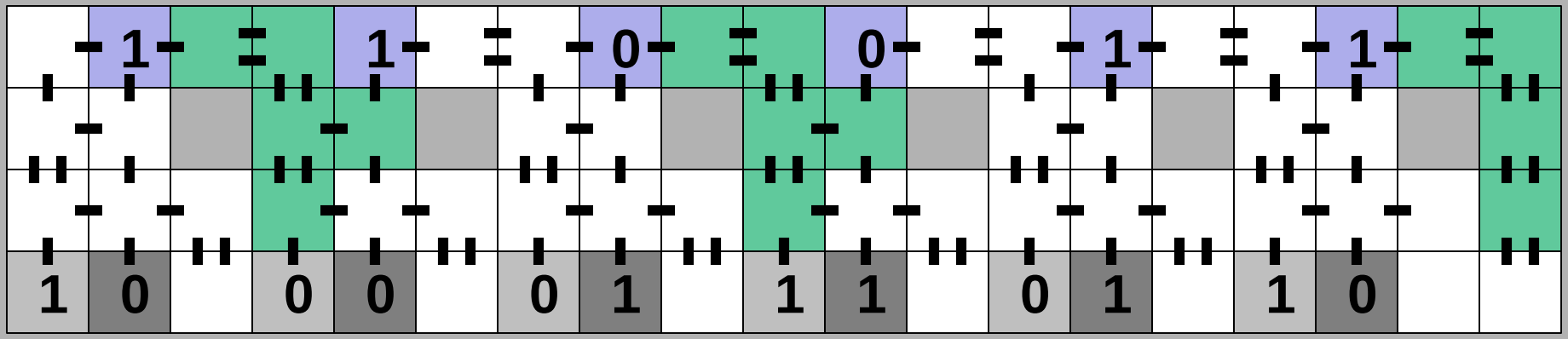
```
  100101
+ 001110
--------
  110 11
```



Addition

Faster average case

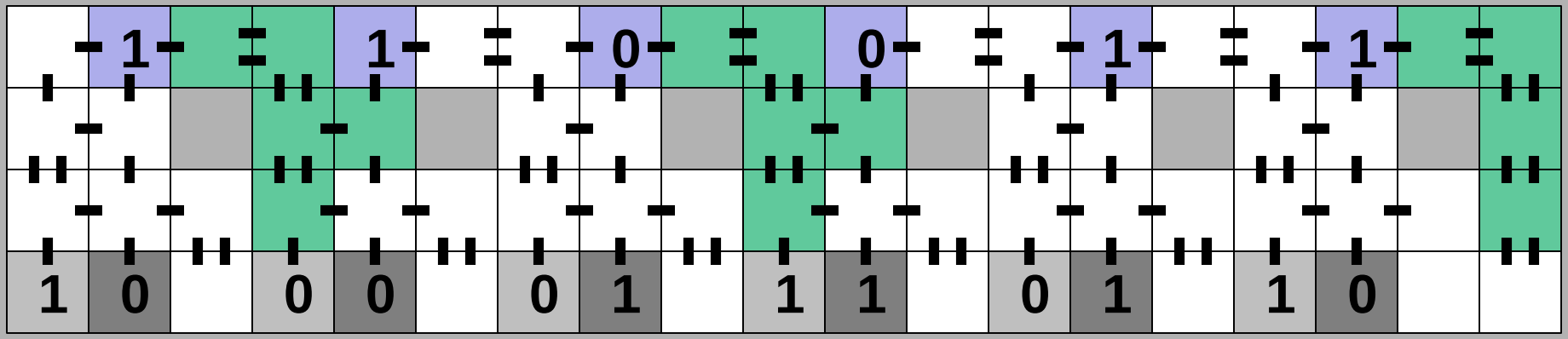
$$\begin{array}{r} 100101 \\ +\ 001110 \\ \hline 110011 \end{array}$$



Addition

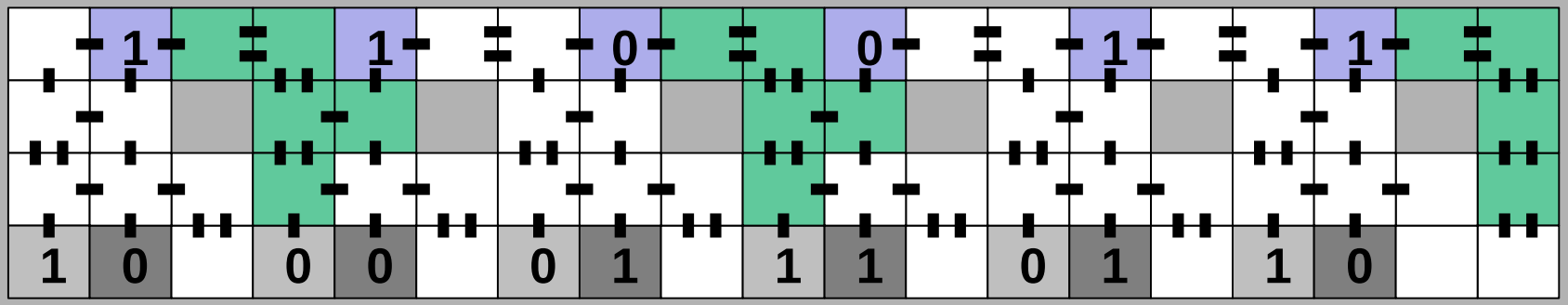
Faster average case

```
  100101
+ 001110
--------
  110011
```

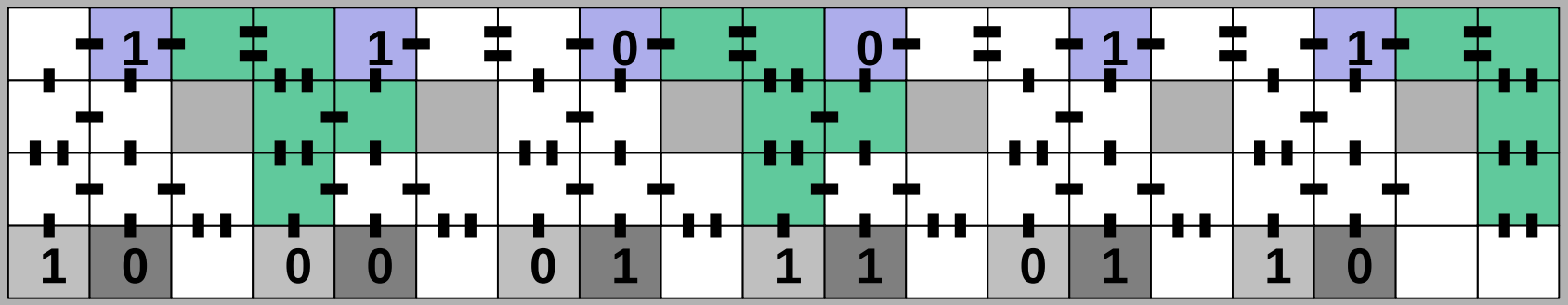


Average case run time: O(log n)

	Addition Complexity	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm	O(n)	O(log n)
Worst-case algorithm		
Combo algorithm		
3D		



	Addition Complexity	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm	O(n)	O(log n)
Worst-case algorithm	?	
Combo algorithm		
3D		

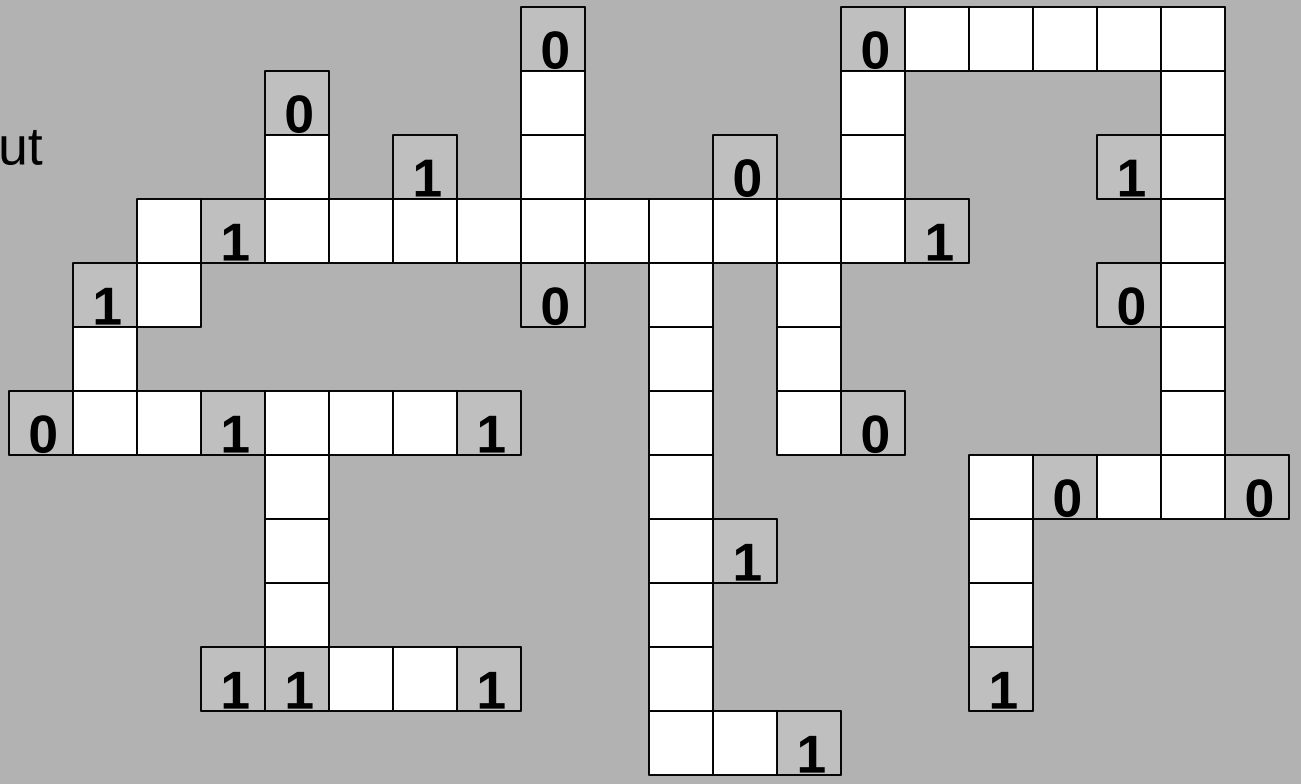


Addition

Worst Case Lower Bound

Some input seed:

n-bits of input

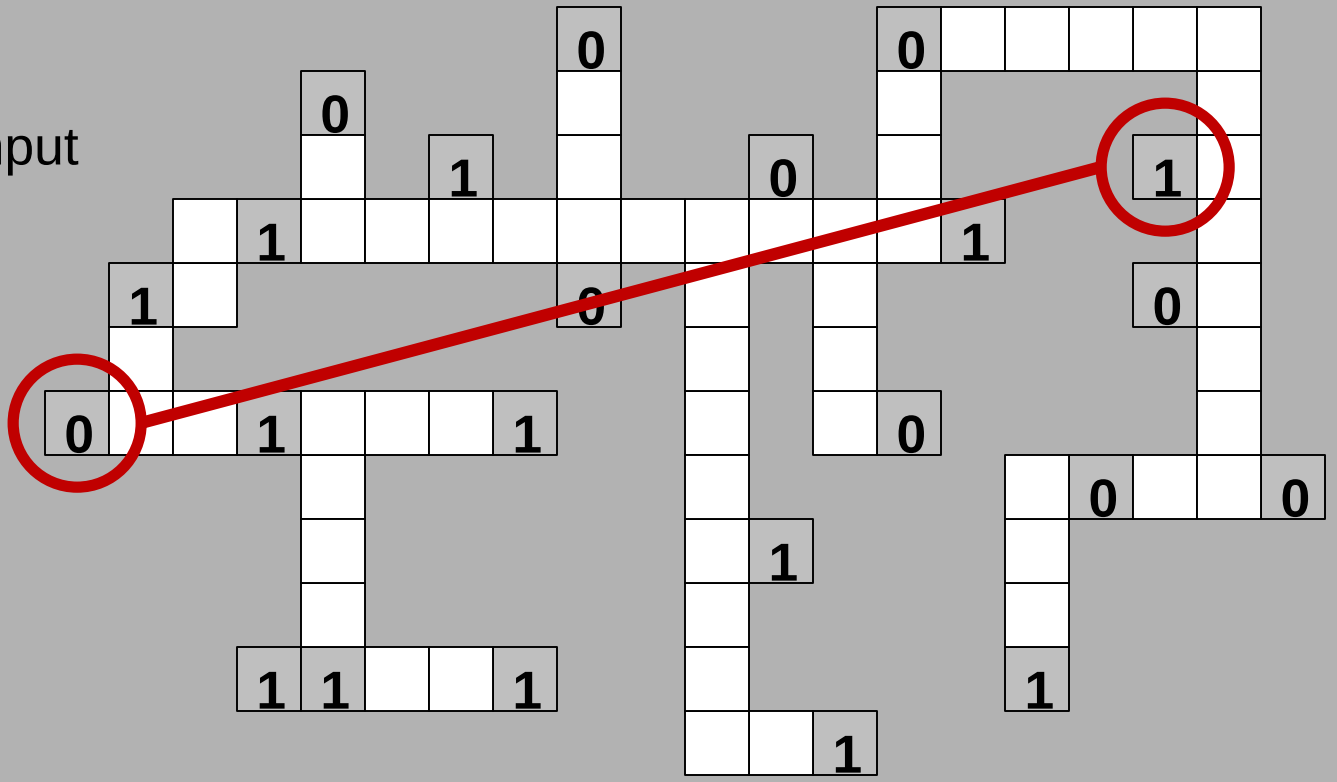


Addition

Worst Case Lower Bound

Some input seed:

n-bits of input



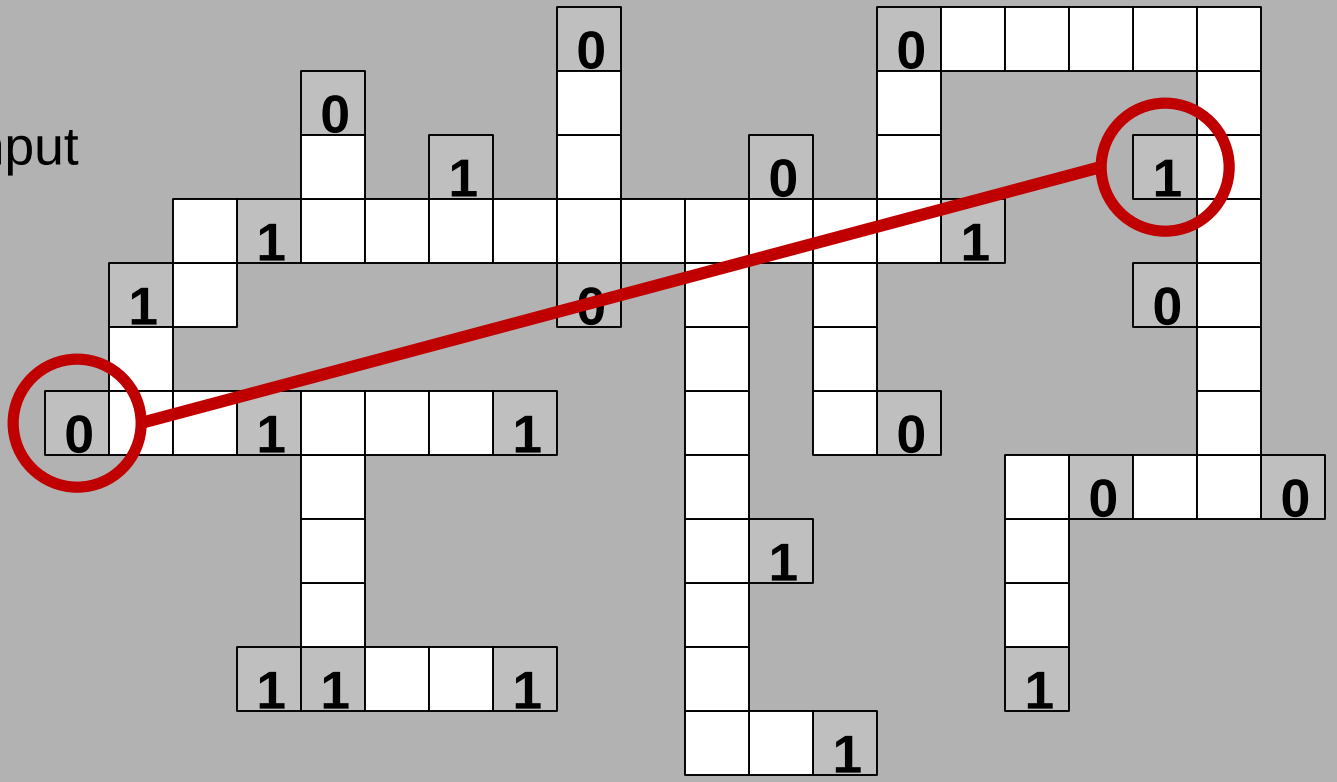
-Must be some bit-pair separated by √**n** distance (for a 2D seed).

Addition

Worst Case Lower Bound

Some input seed:

n-bits of input



-Must be some bit-pair separated by **√n** distance (for a 2D seed).

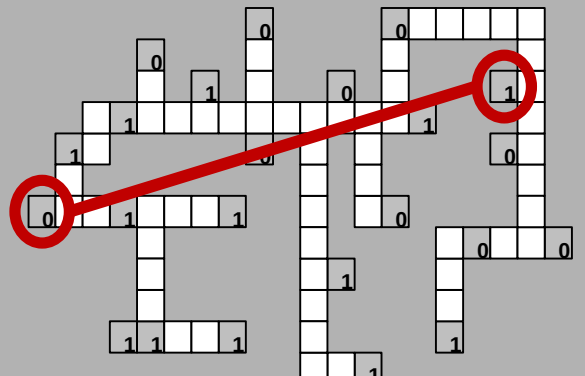
-Any function depending on the AND of these two bits requires:

Ω(√n) time in the worst case.

-includes addition, multiplication, and many other functions.

	Addition Complexity		
	Worst case		Average case
	upper	lower	
[Brun 2007]	O(n)		O(n)
Average-case algorithm	O(n)		O(log n)
Worst-case algorithm	?	Ω(√n)	
Combo algorithm			
3D			

Is $O(\sqrt{n})$ achievable?



Addition Parallelization Scheme: worst case

Break into $\sqrt{n}$ subproblems

```
  101 100 101
+001 010 110
-----------
```

Run time:

Addition Parallelization Scheme: worst case

Break into $\sqrt{n}$ subproblems

```
  101      100      101
+001     +010     +110
-----    -----    -----
```

Run time:

Addition Parallelization Scheme: worst case

Break into $\sqrt{n}$ subproblems

Compute subsums: $O(\sqrt{n})$

```
  101      100      101
+001     +010     +110
-----    -----    -----
  110      110     1011
```

Run time:

Addition Parallelization Scheme: worst case

Break into $\sqrt{n}$ subproblems

Compute subsums: $O(\sqrt{n})$

Increment: $O(\sqrt{n})$

Run time:

```
 101     100     101
+001    +010    +110
----    ----    ----
 110     110    1011
 111     111    1100
```

Addition Parallelization Scheme: worst case

Break into √n subproblems

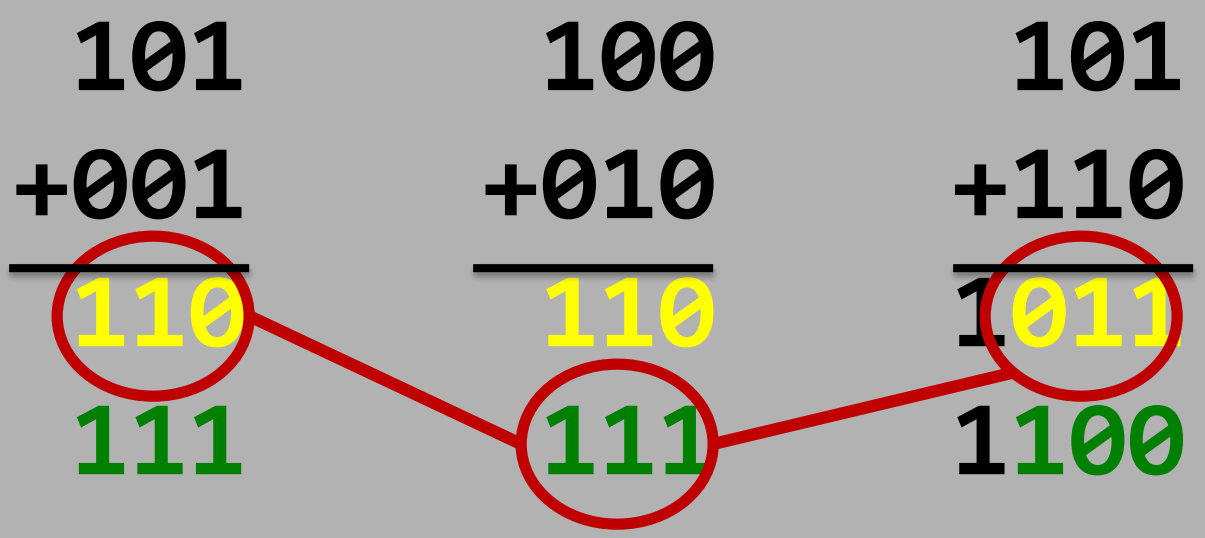
Compute subsums: O(√n)

Increment: O(√n)

Select correct Strings: O(√n)

Run time:

O(√n)



Output:

110 111 011

Addition Parallelization Scheme: worst case

Break into √n subproblems

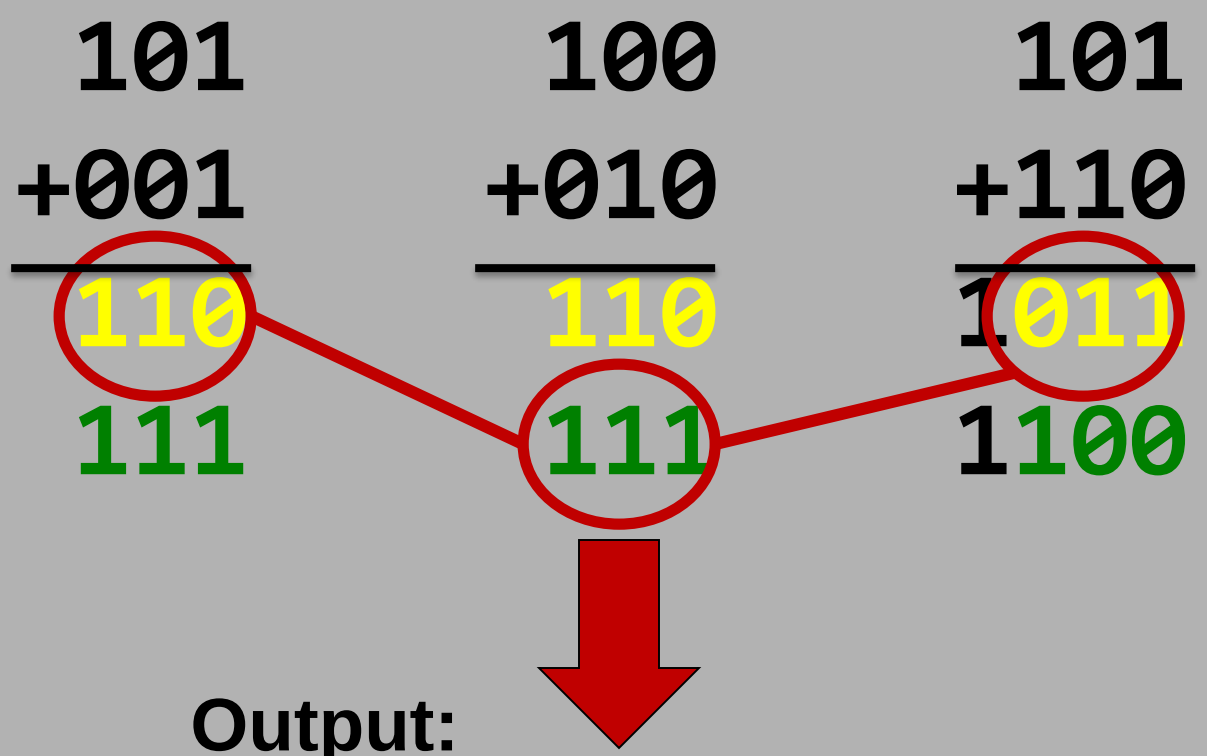
Compute subsums: O(√n)

Increment: O(√n)

Select correct Strings: O(√n)

Run time:

O(√n)



Output:

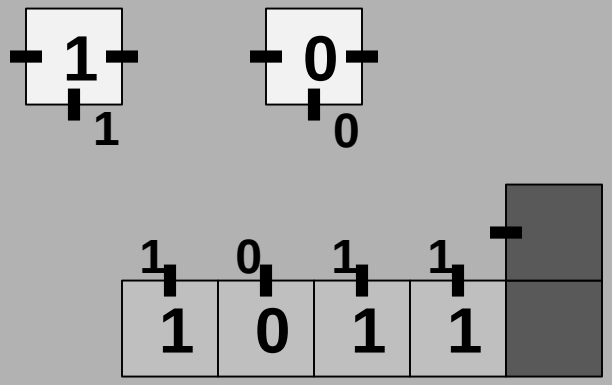
110 111 011

Challenge:

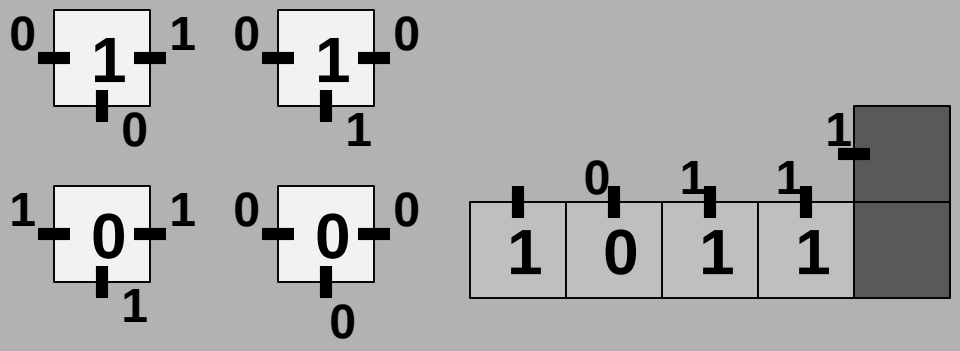
Can we implement this parallelization scheme within a tile system?

Yes. First, some primitives.

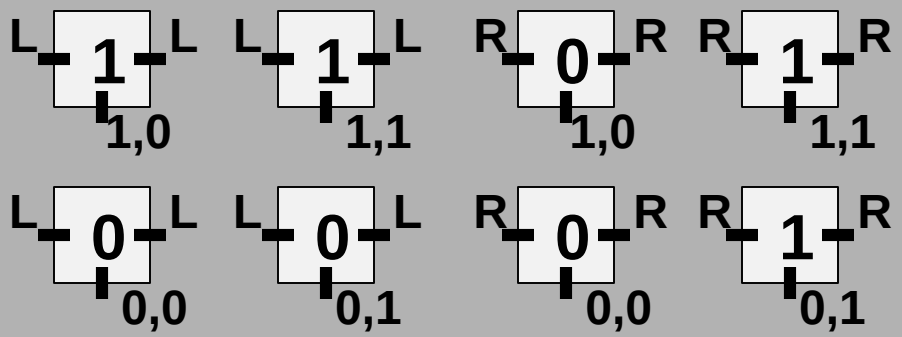
Copy:

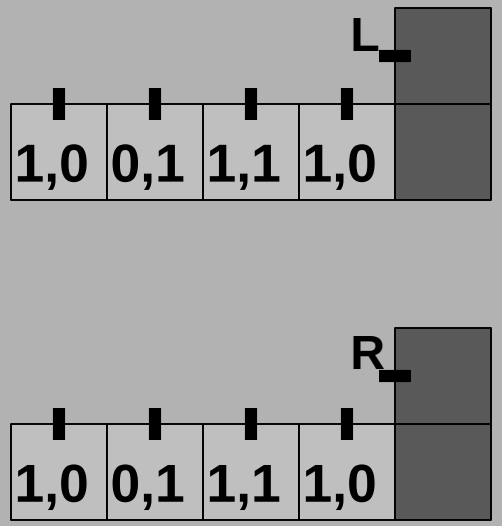


Increment:



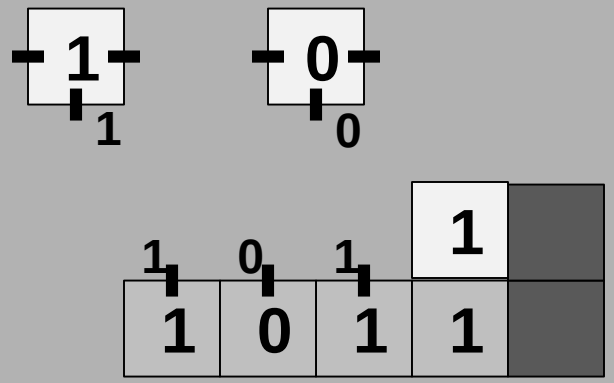
Select:



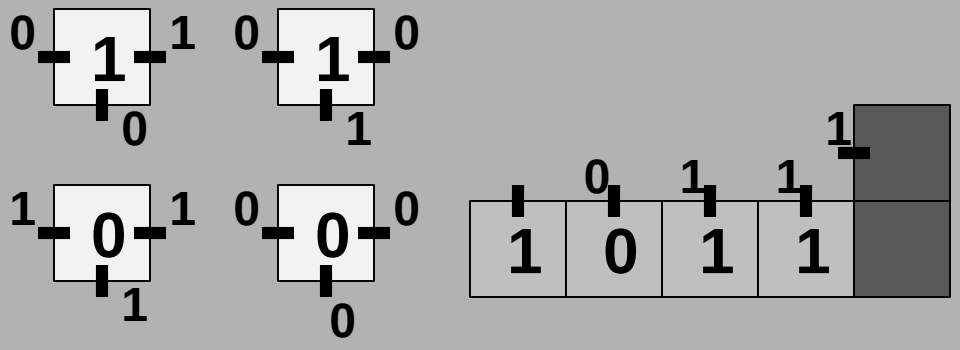


Yes. First, some primitives.

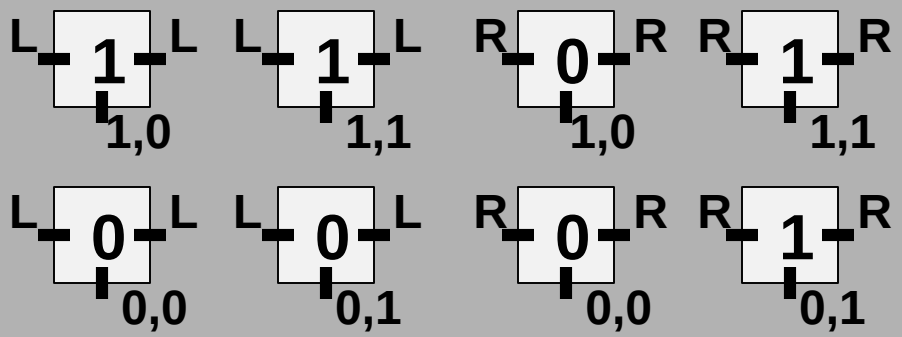
Copy:

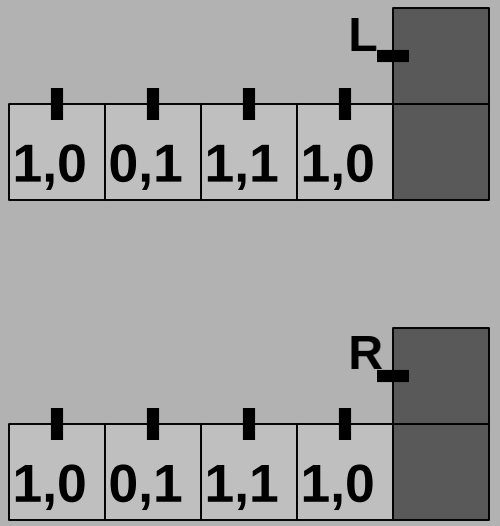


Increment:



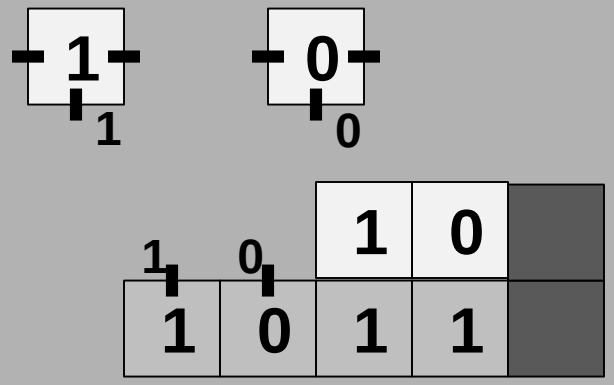
Select:



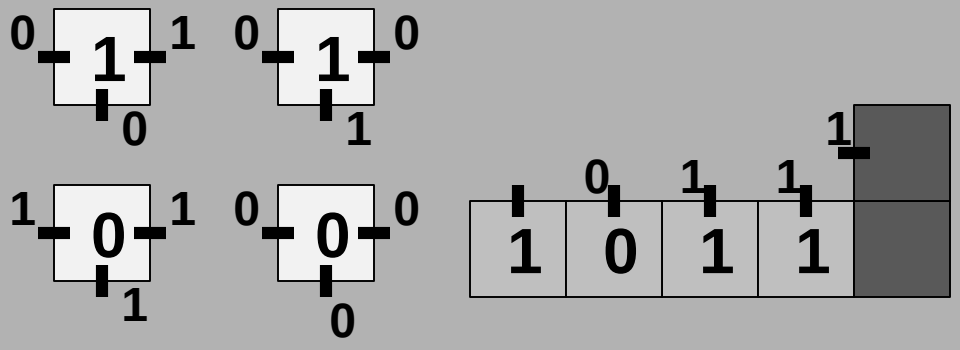


Yes. First, some primitives.

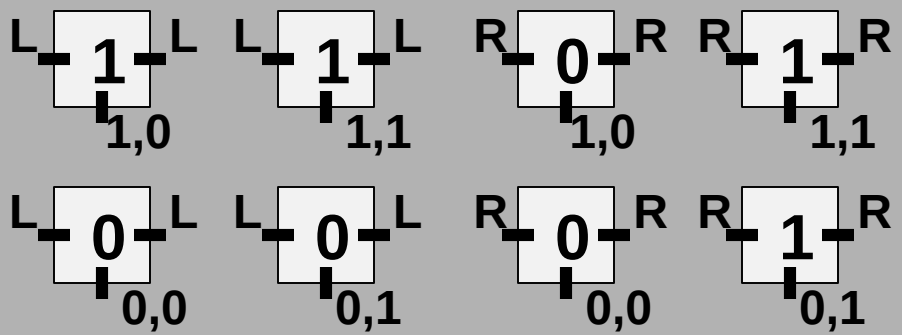
Copy:

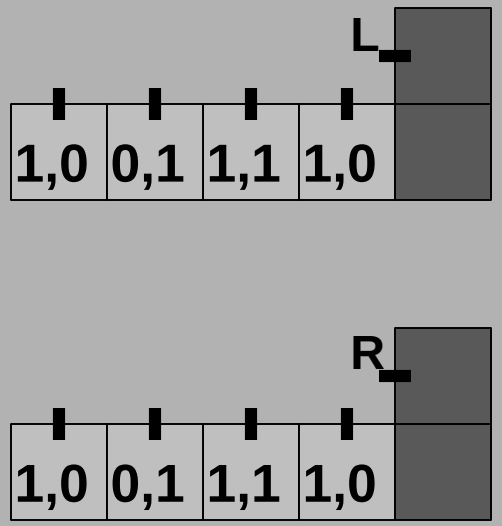


Increment:



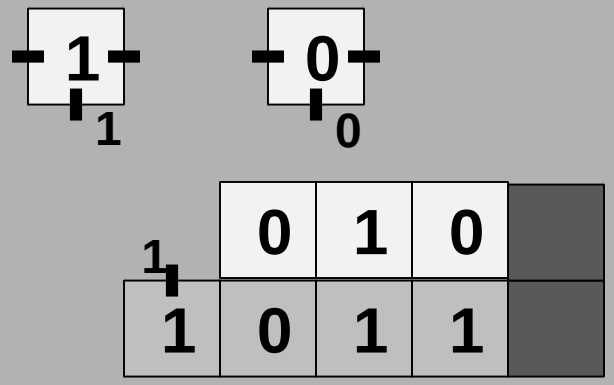
Select:



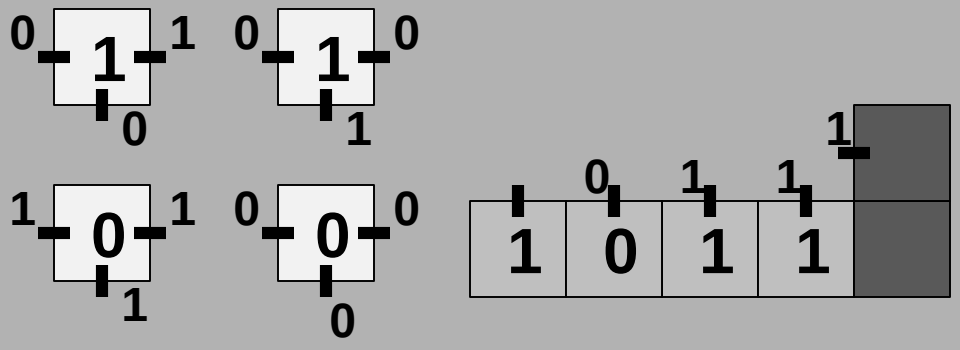


Yes. First, some primitives.

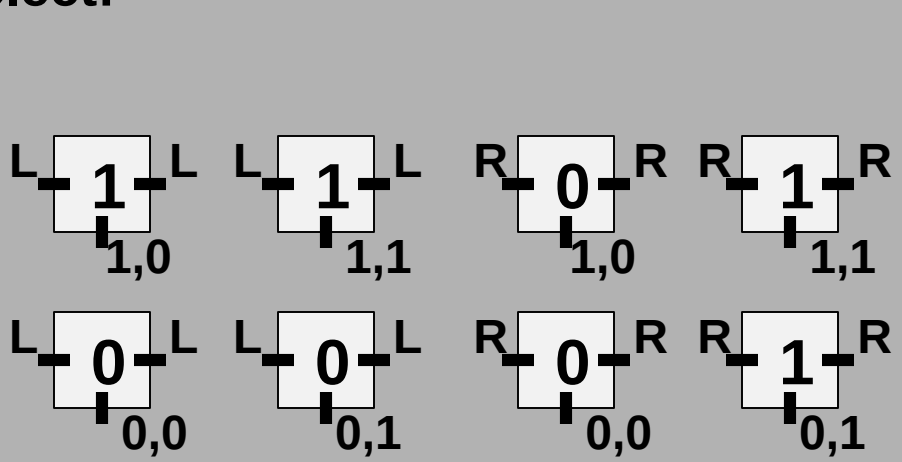
Copy:

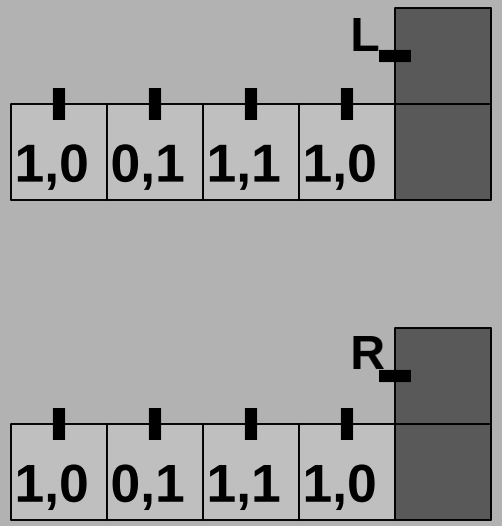


Increment:



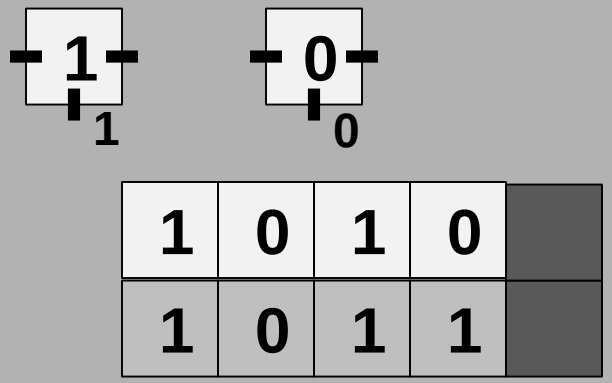
Select:



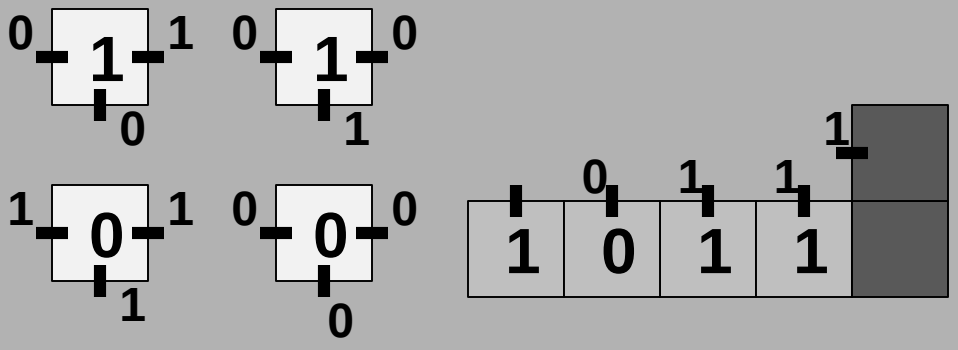


Yes. First, some primitives.

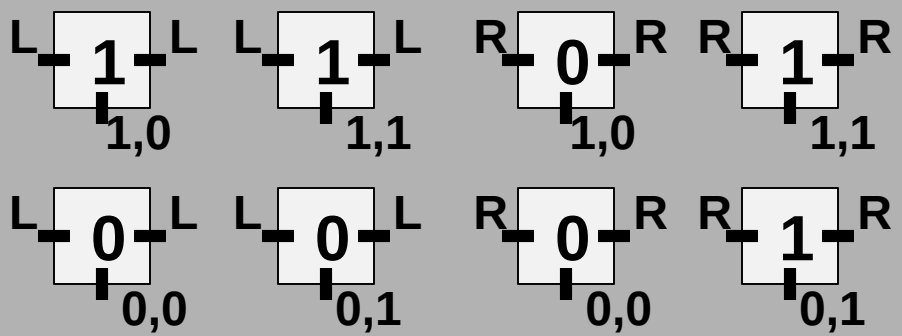
Copy:

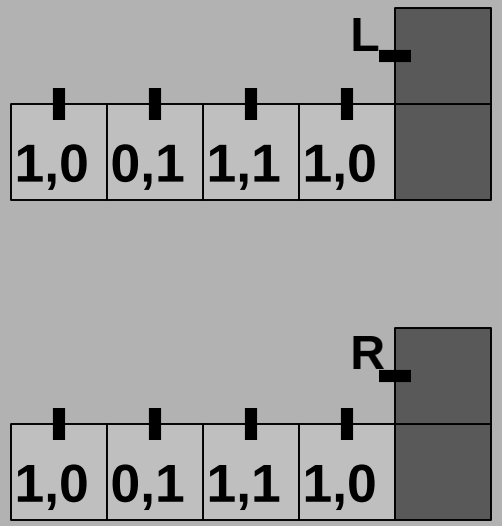


Increment:



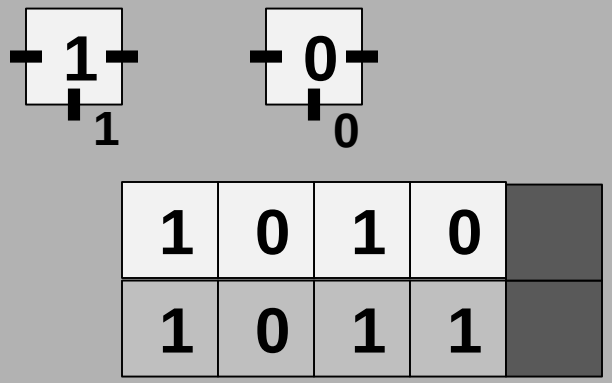
Select:



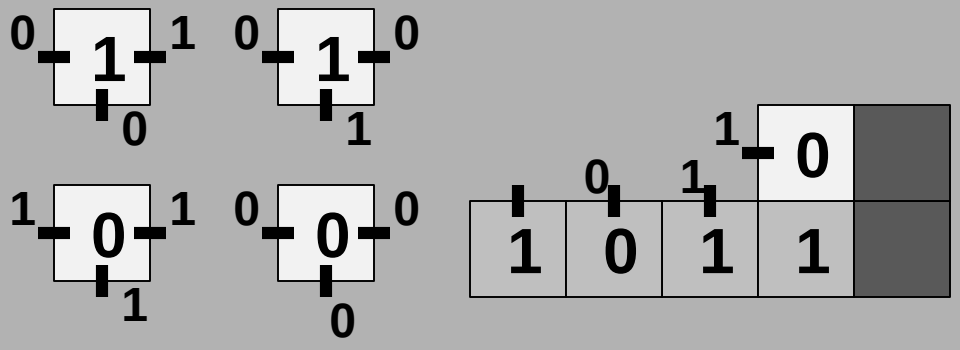


Yes. First, some primitives.

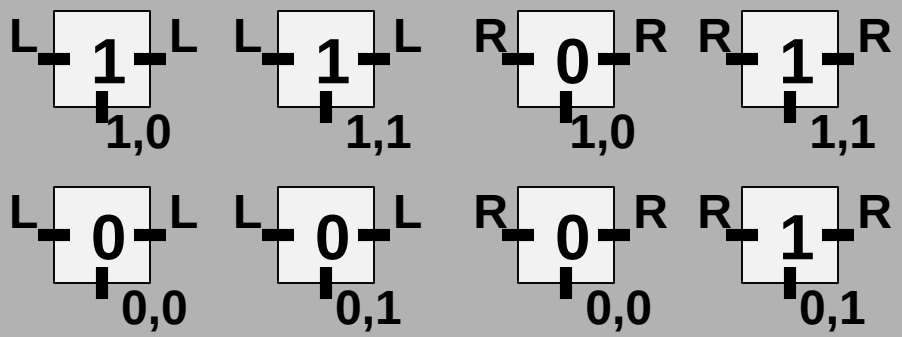
Copy:

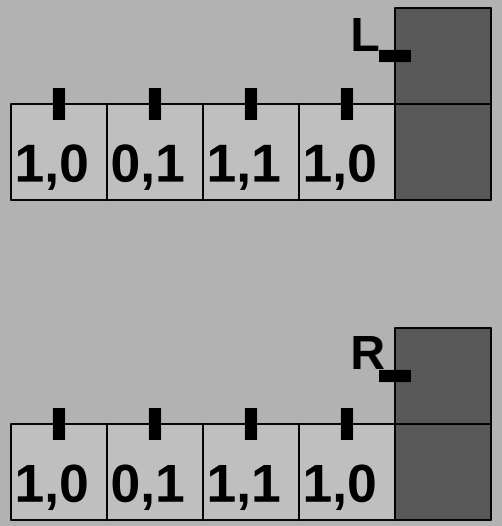


Increment:



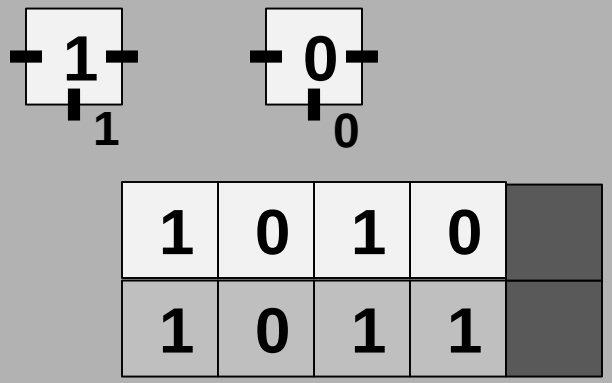
Select:



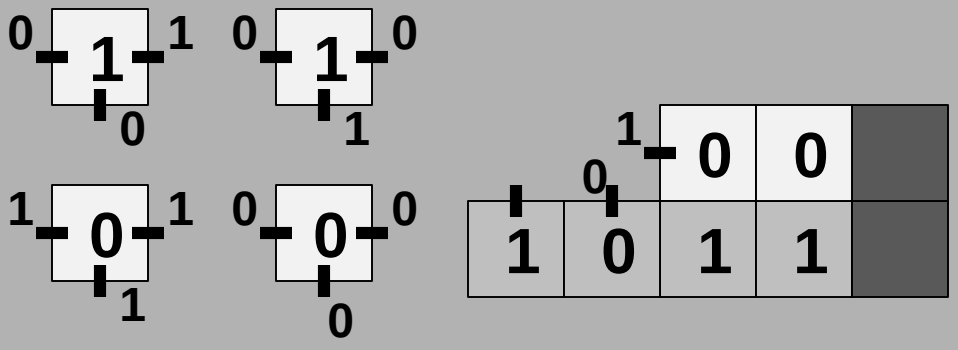


Yes. First, some primitives.

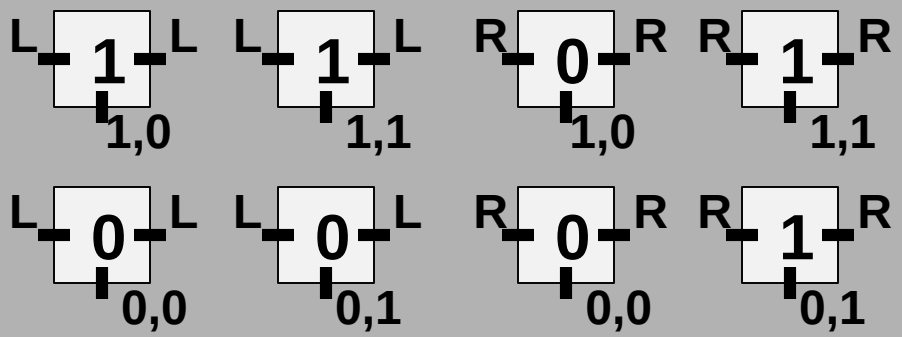
Copy:

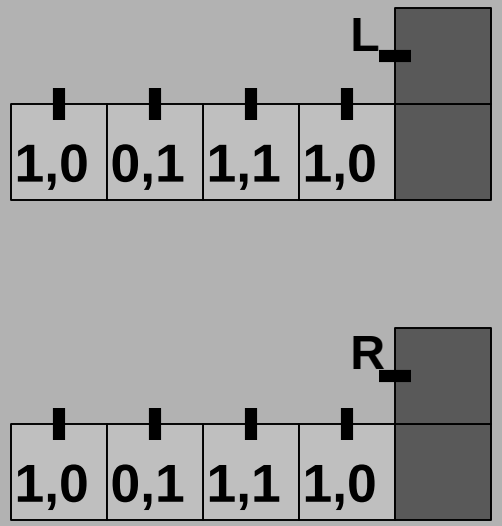


Increment:



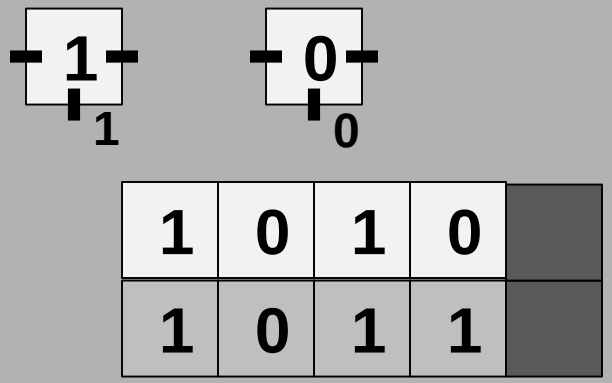
Select:



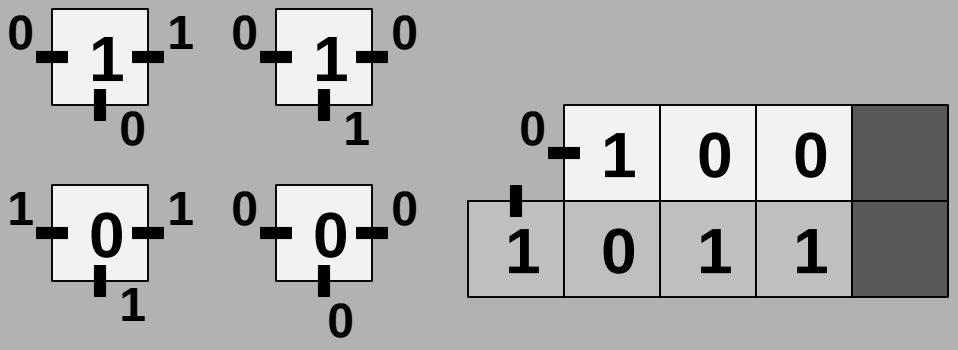


Yes. First, some primitives.

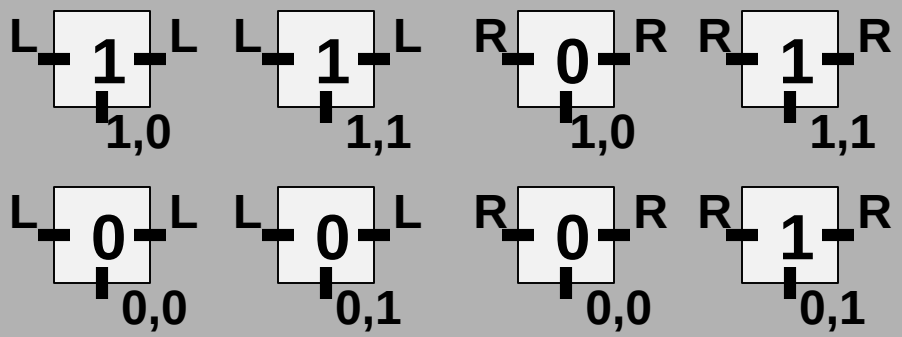
Copy:

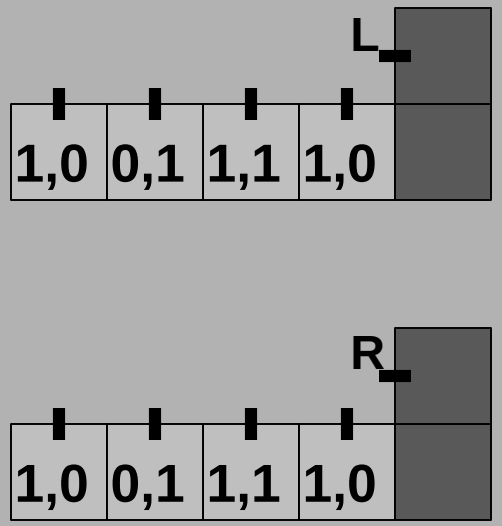


Increment:



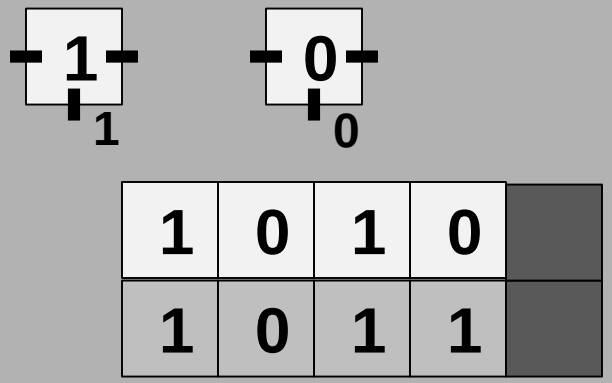
Select:



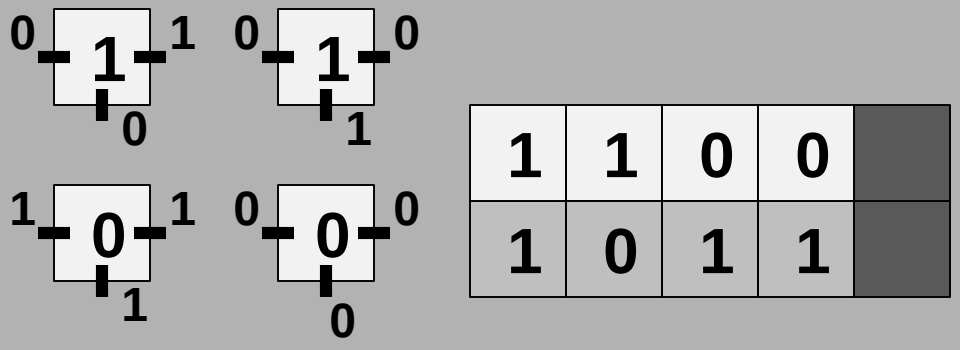


Yes. First, some primitives.

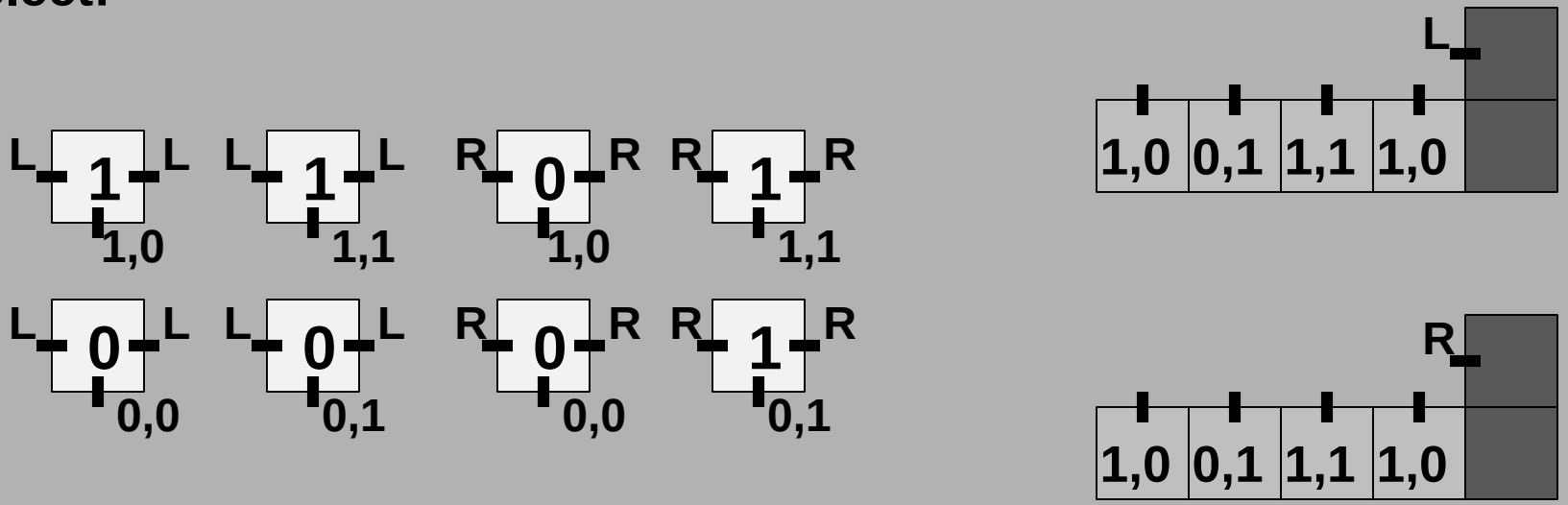
Copy:



Increment:

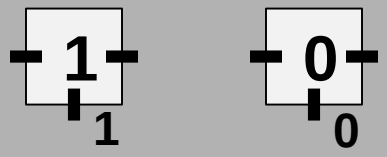


Select:



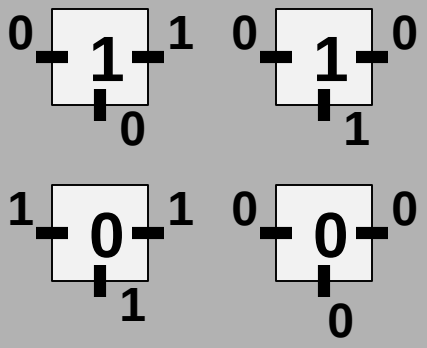
Yes. First, some primitives.

Copy:



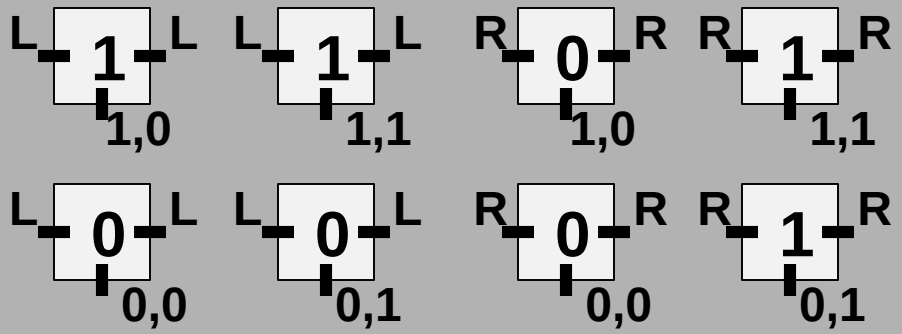
1	0	1	0	
1	0	1	1	

Increment:



1	1	0	0	
1	0	1	1	

Select:



1	0	1	1	L
1,0	0,1	1,1	1,0	

0	1	1	0	R
1,0	0,1	1,1	1,0	

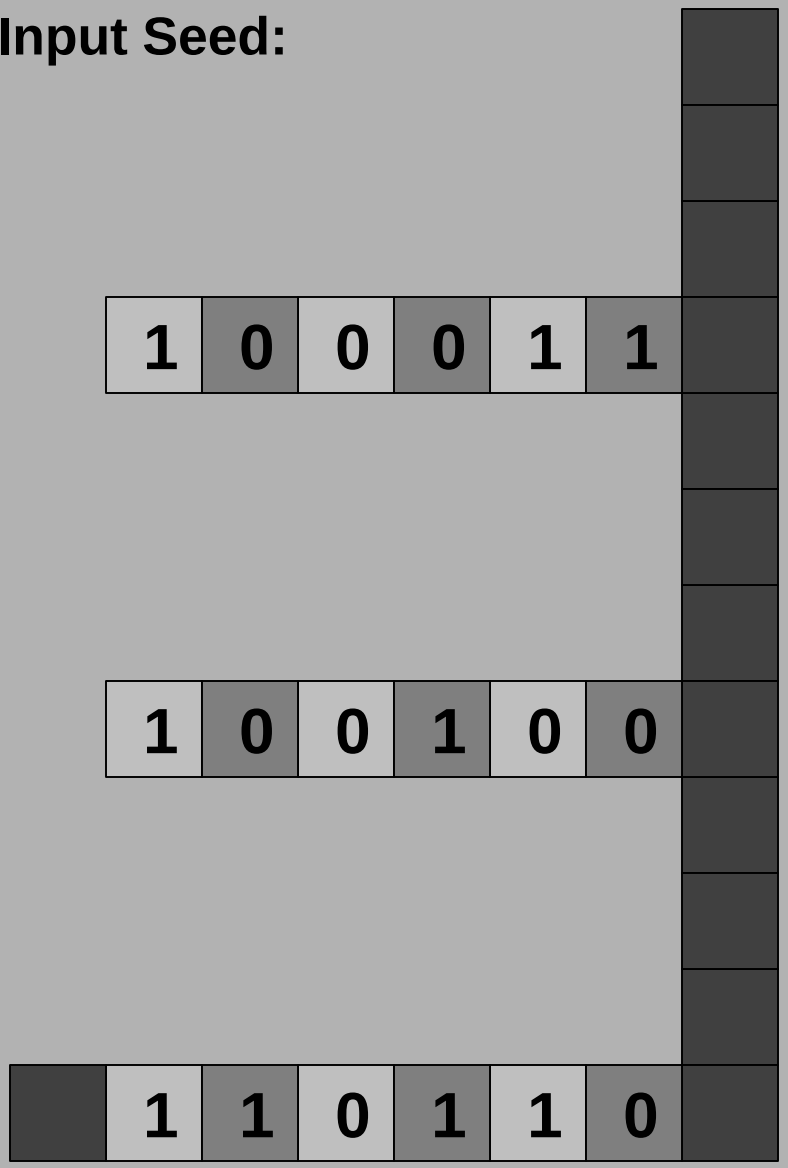
Addition

Faster **worst case**

Input Seed:

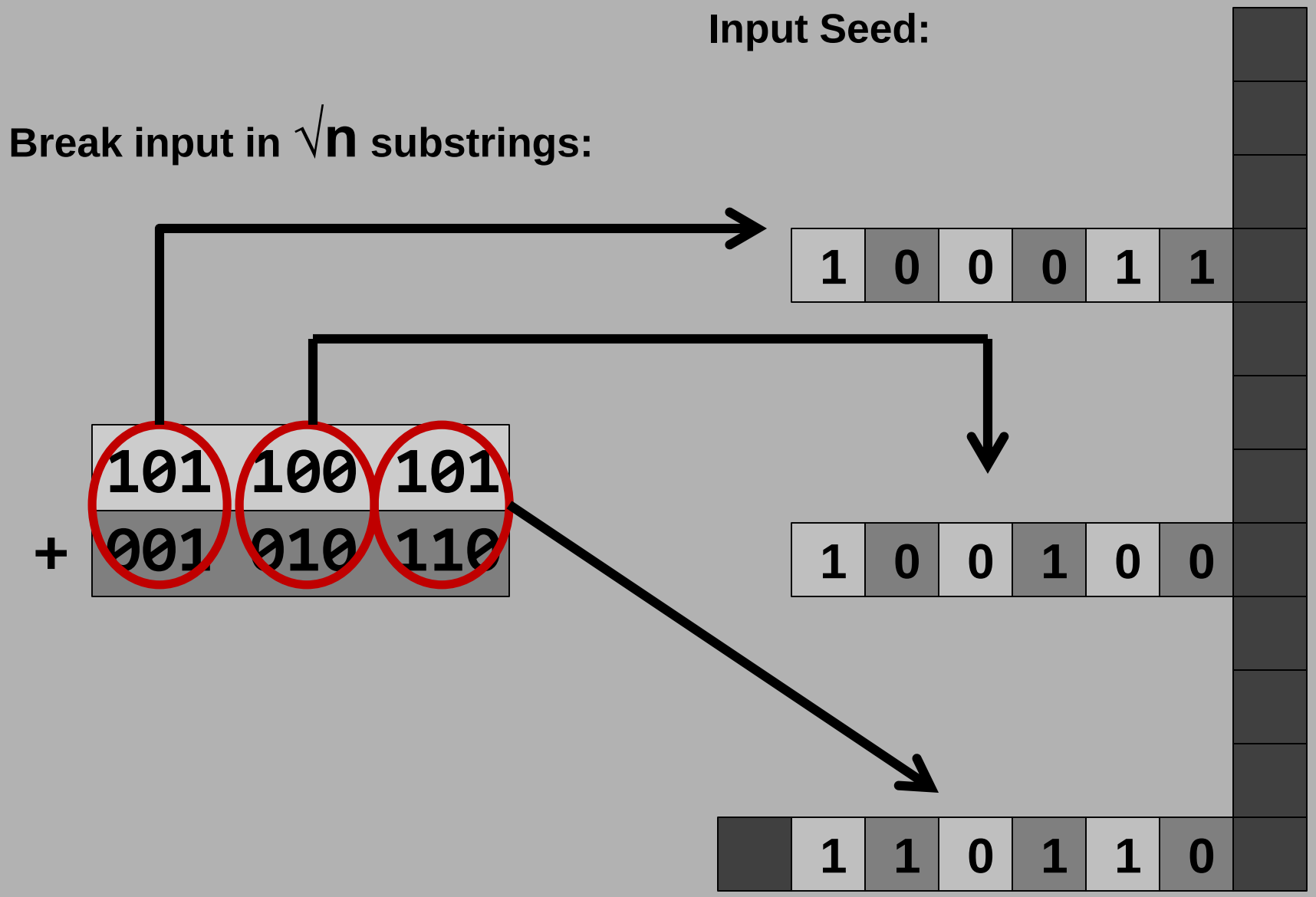
Break input in $\sqrt{n}$ substrings:

	101	100	101
+	001	010	110



Addition

Faster **worst case**



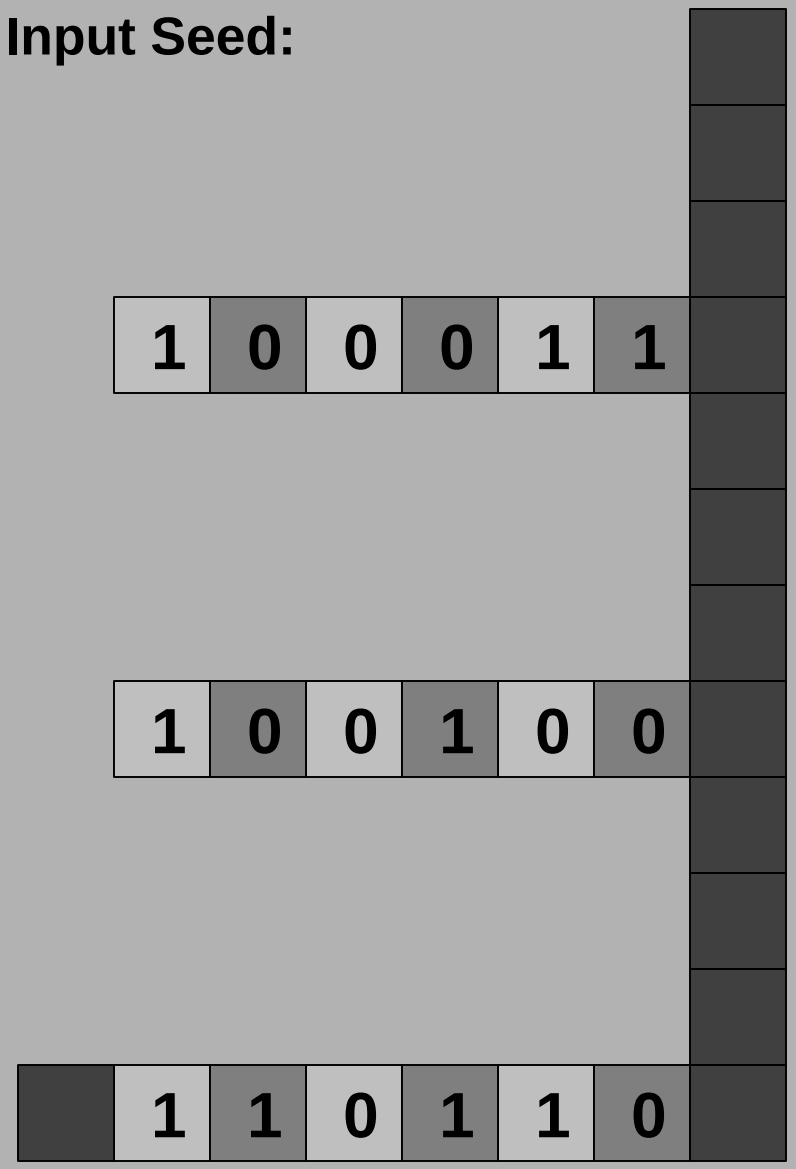
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row

Phase 2:

Phase 3:



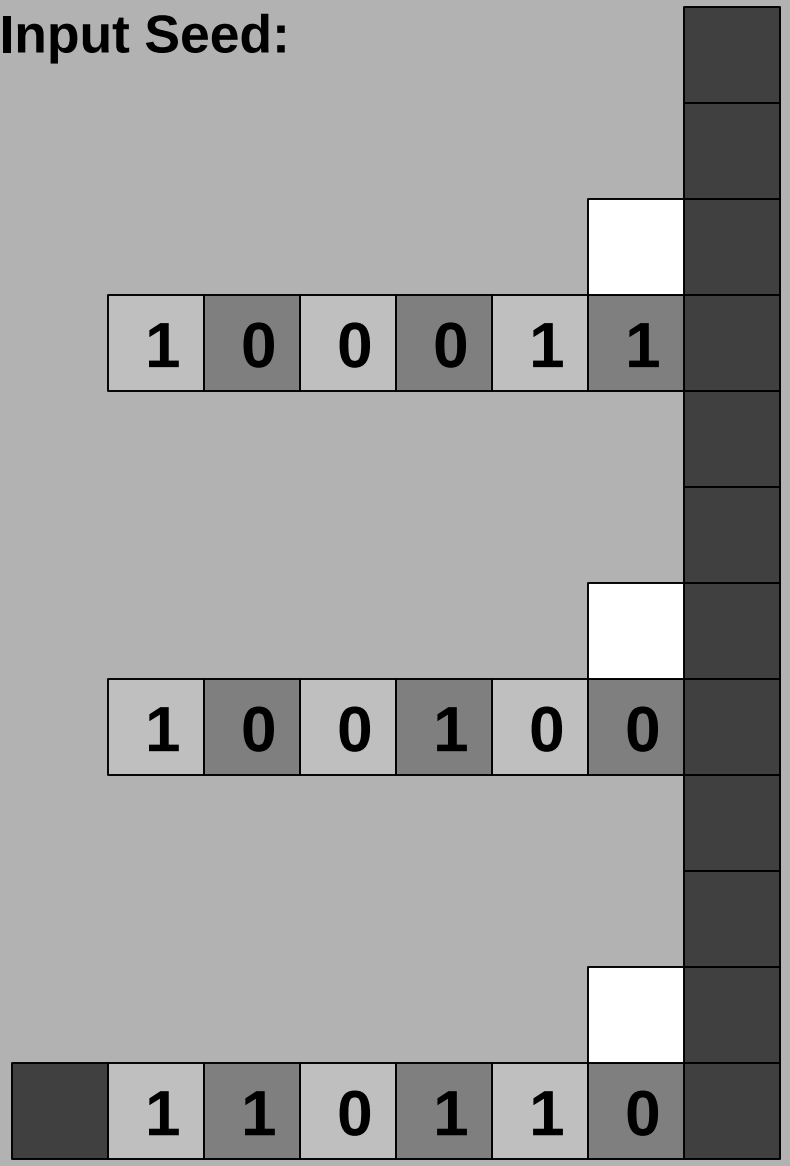
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row

Phase 2:

Phase 3:



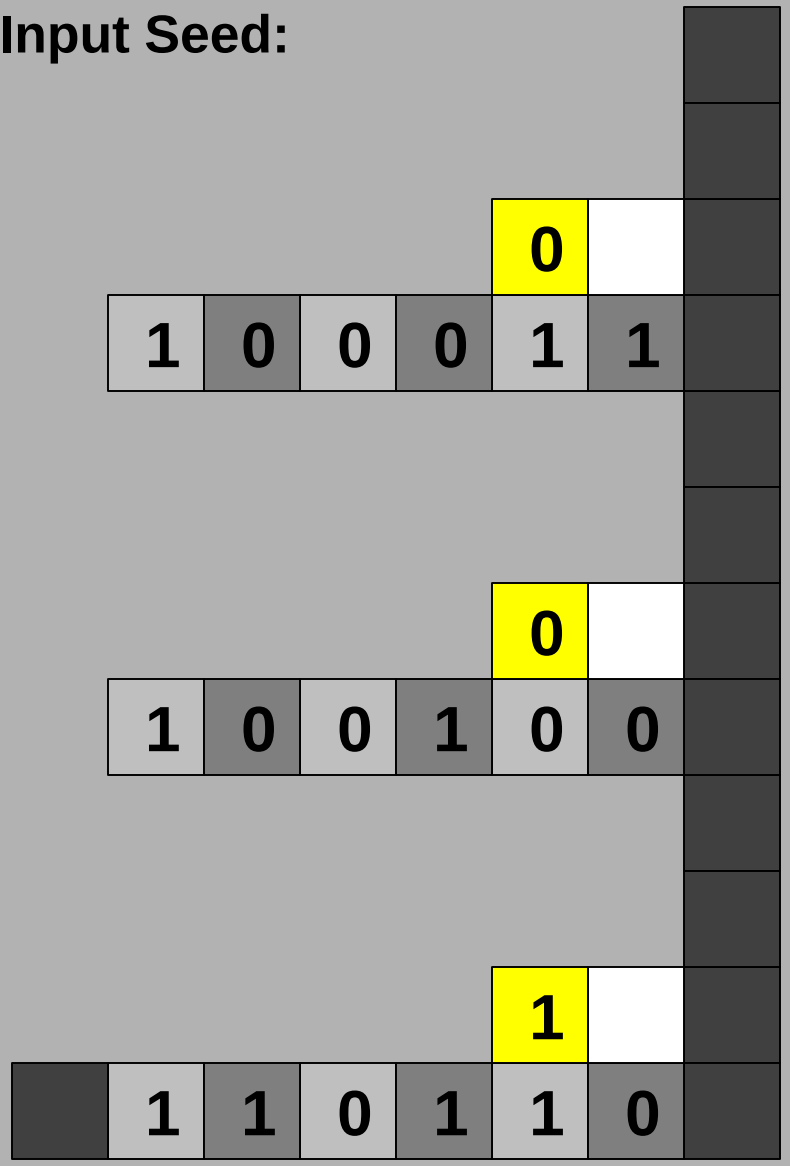
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row

Phase 2:

Phase 3:



Addition

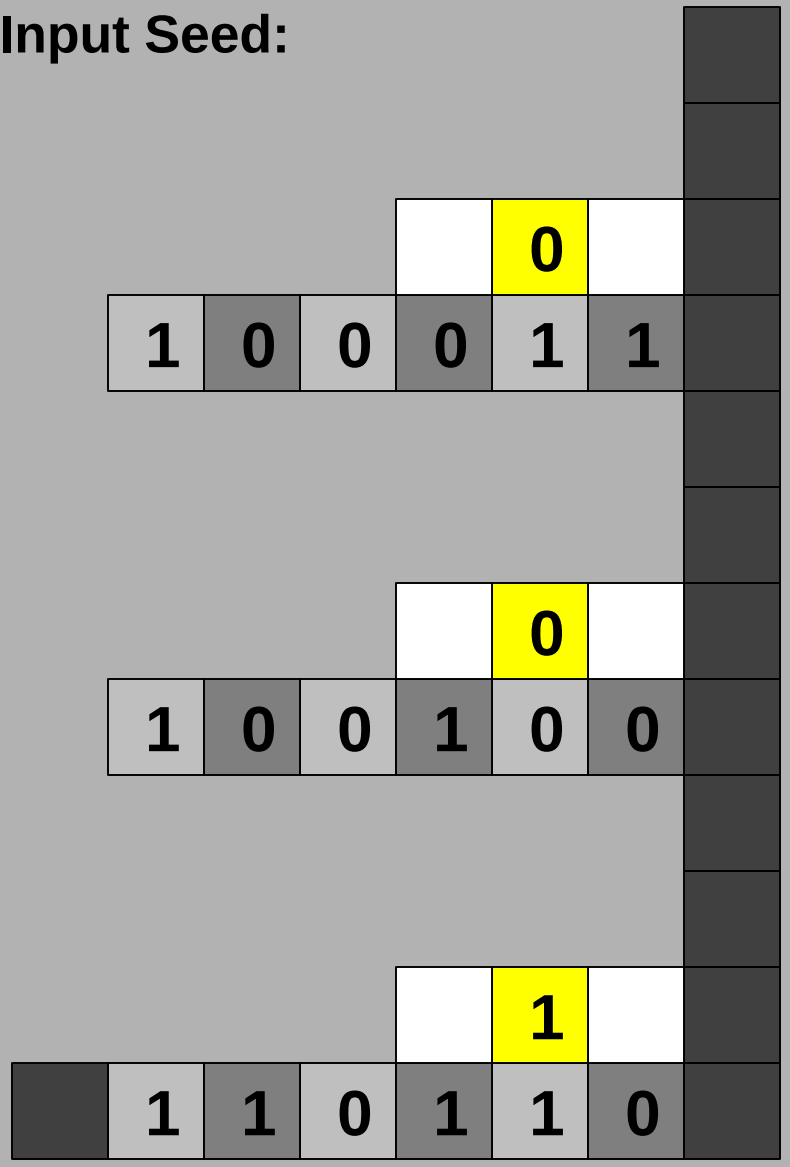
Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row

Phase 2:

Phase 3:

Input Seed:



Addition

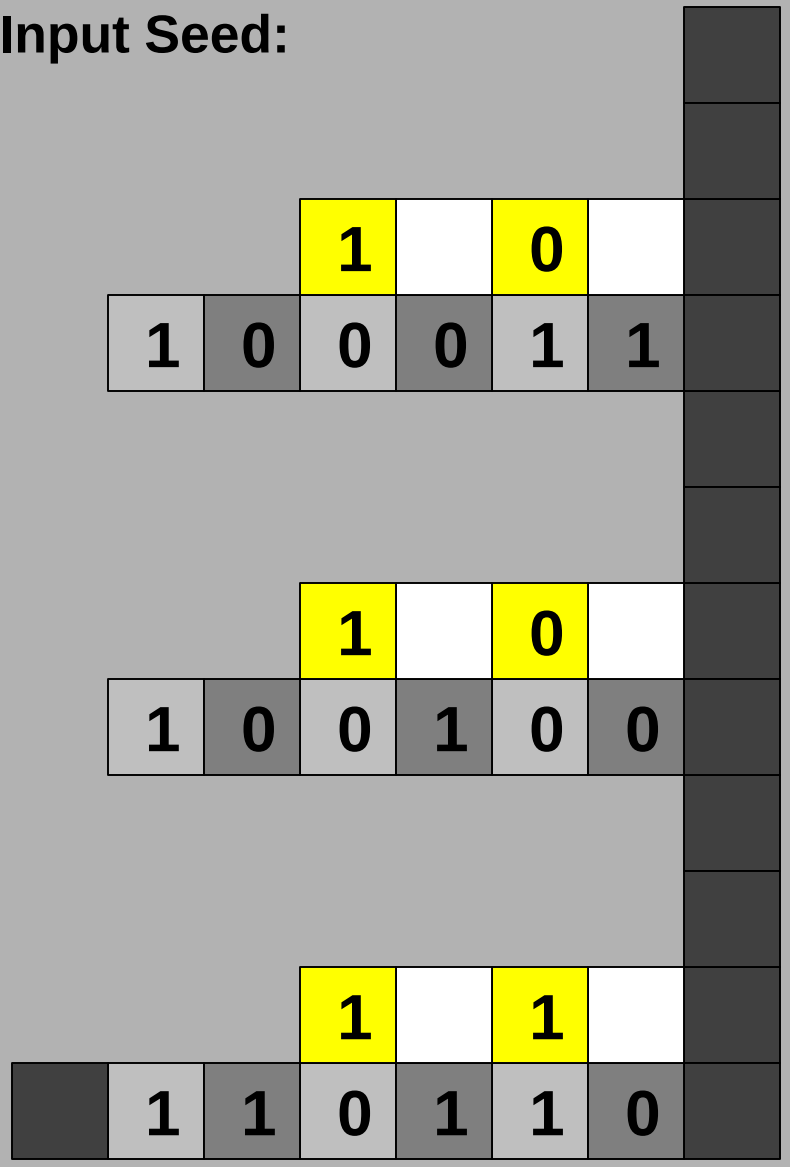
Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row

Phase 2:

Phase 3:

Input Seed:



Addition

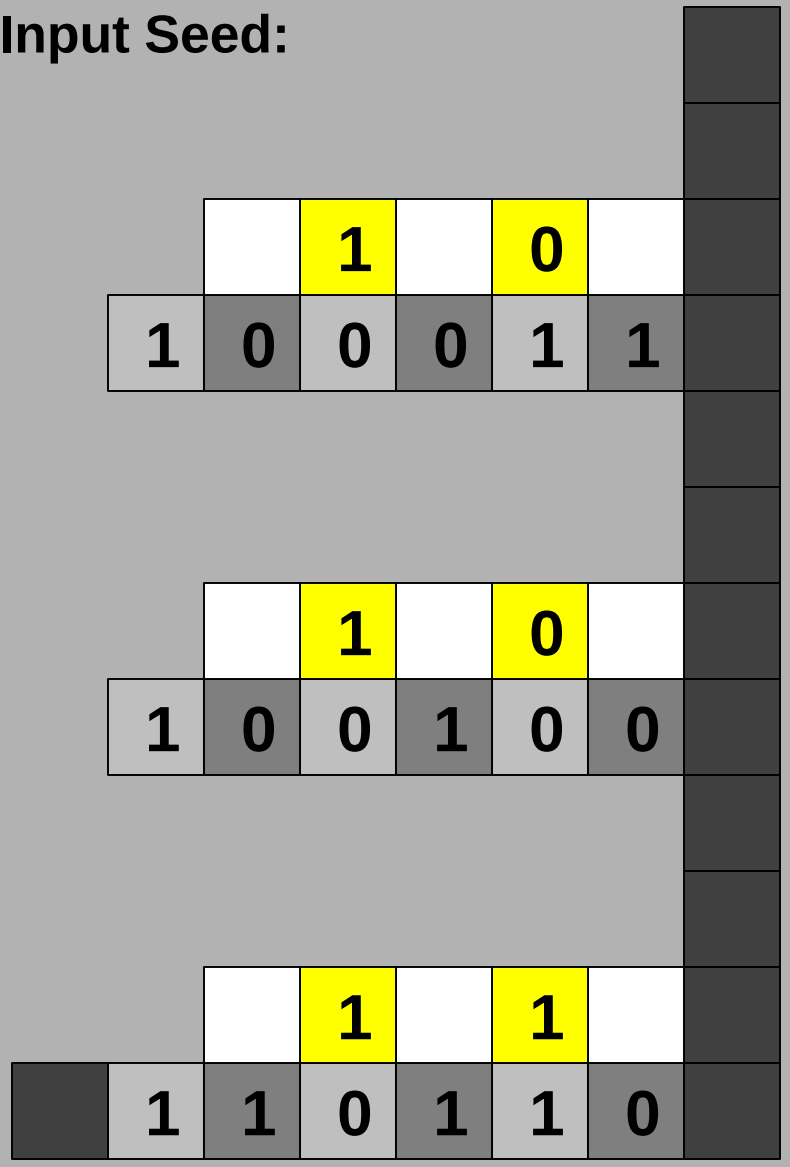
Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row

Phase 2:

Phase 3:

Input Seed:



Addition

Faster **worst case**

O(√n) **Phase 1:**

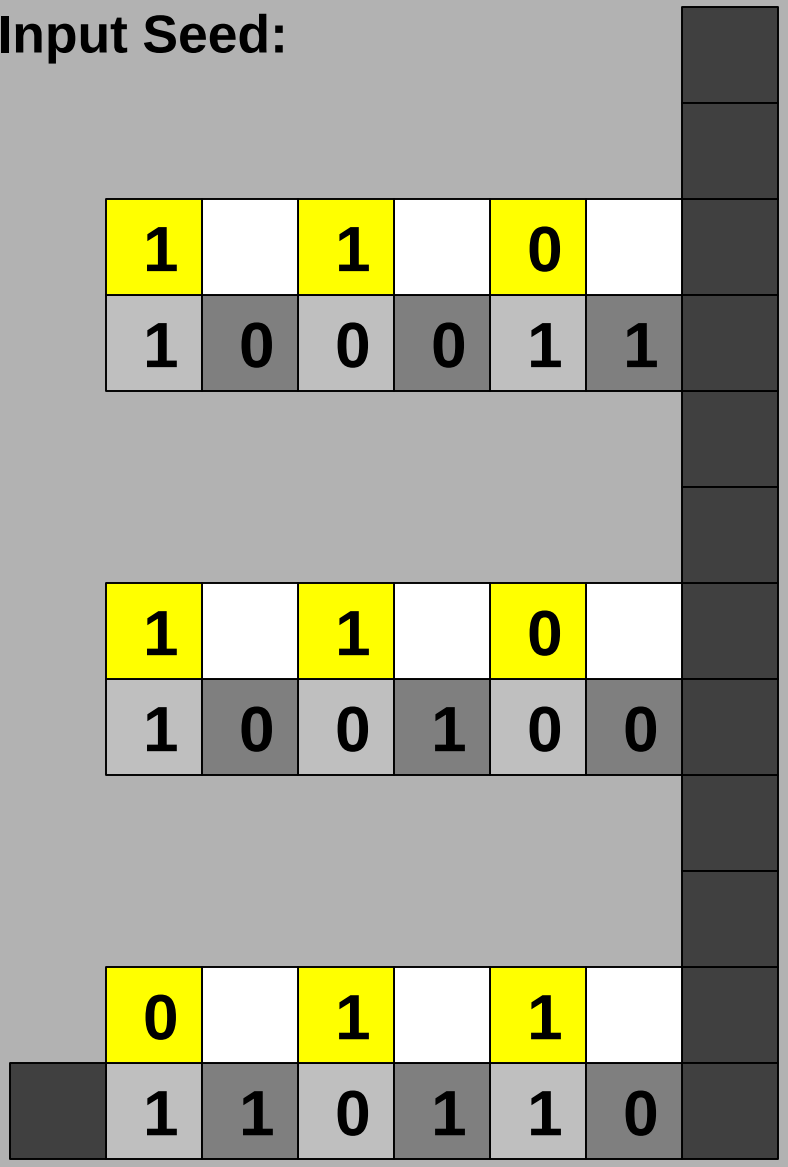
1.1 Sum each row

1.2 **Increment/copy**

Phase 2:

Phase 3:

Input Seed:



Addition

Faster **worst case**

O(√n) **Phase 1:**

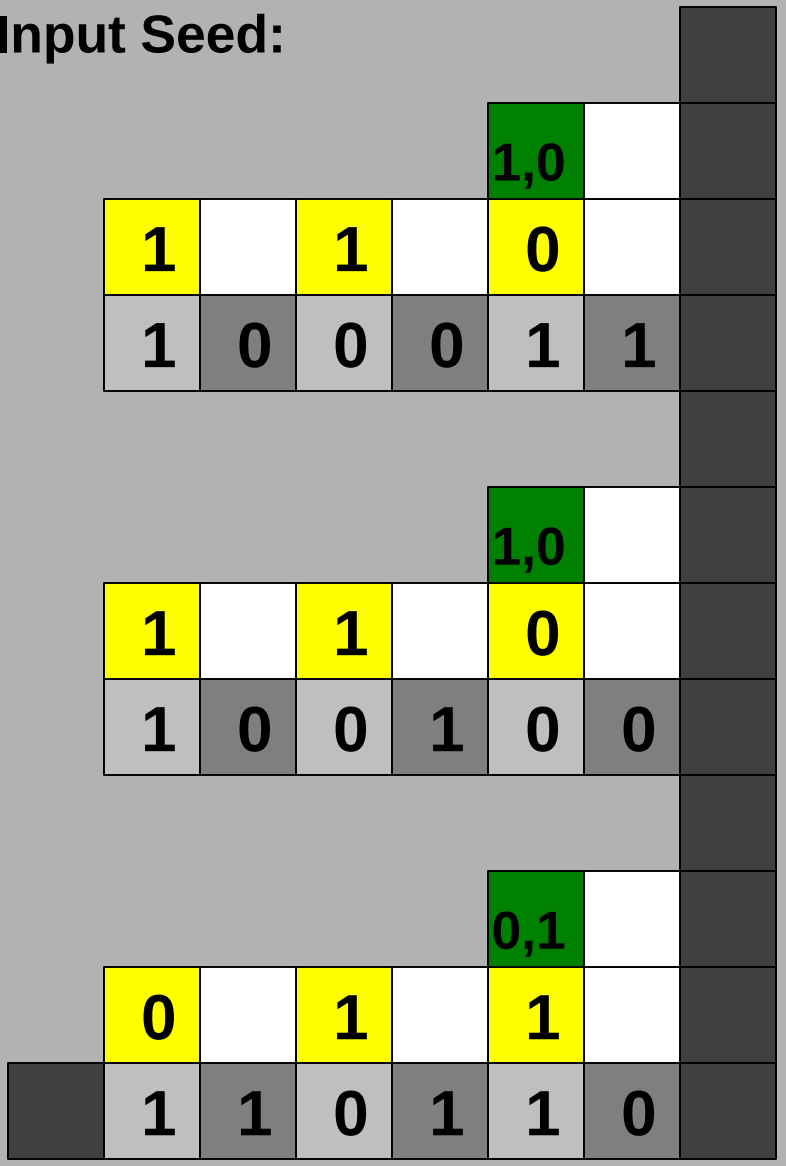
1.1 Sum each row

1.2 **Increment/copy**

Phase 2:

Phase 3:

Input Seed:



Addition

Faster **worst case**

O(√n) **Phase 1:**

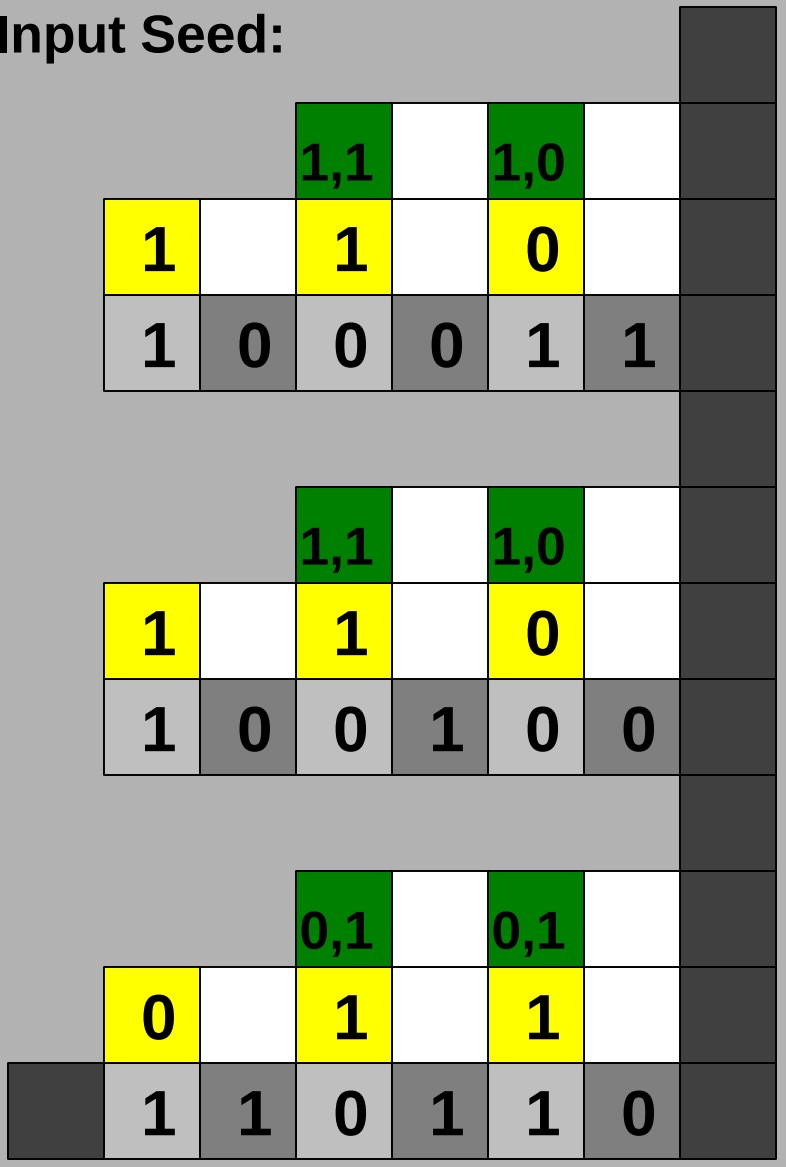
1.1 Sum each row

1.2 **Increment/copy**

Phase 2:

Phase 3:

Input Seed:



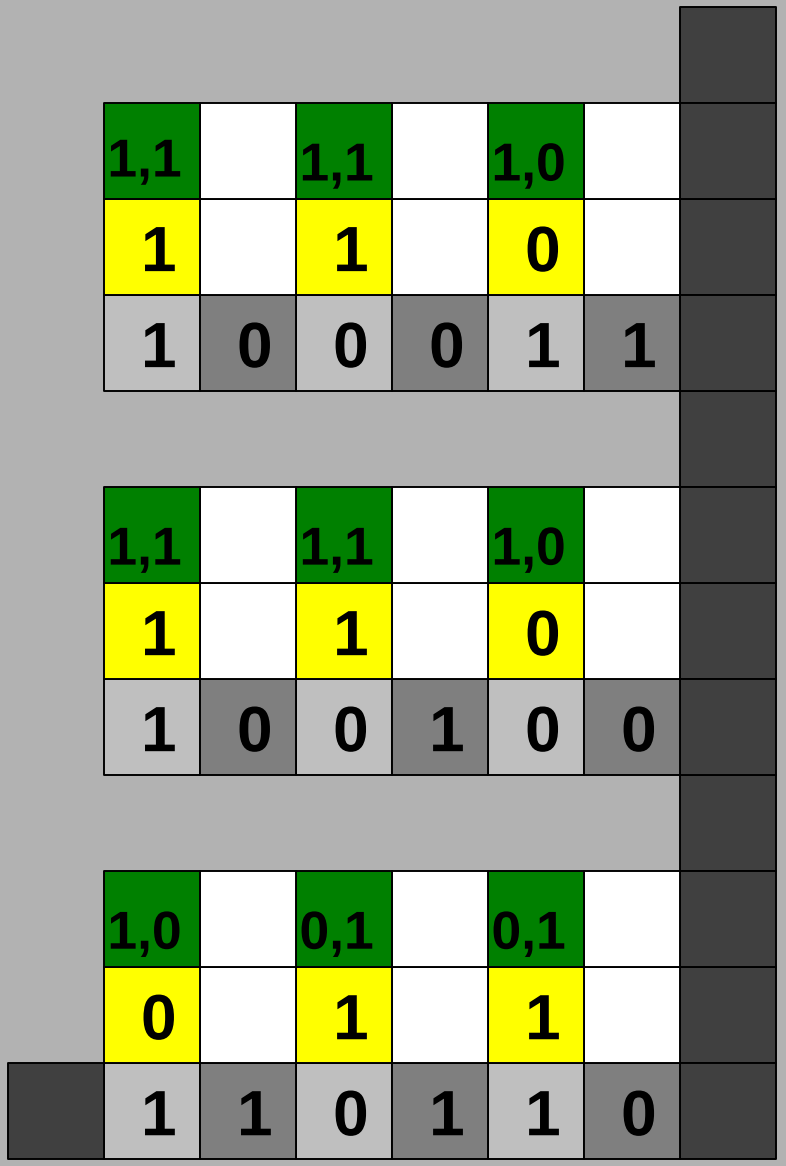
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row
1.2 Increment/copy

O(√n) **Phase 2:**
Grow carry-selector

Phase 3:



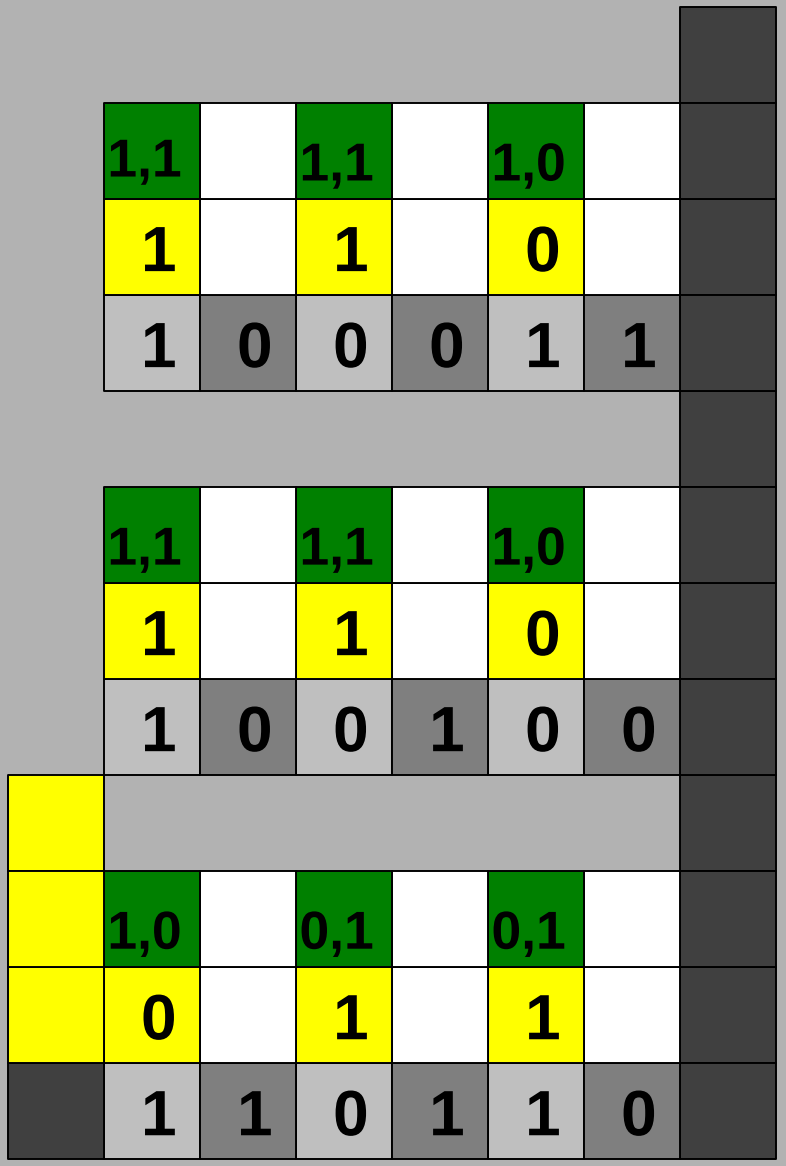
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row
1.2 Increment/copy

O(√n) **Phase 2:**
Grow carry-selector

Phase 3:



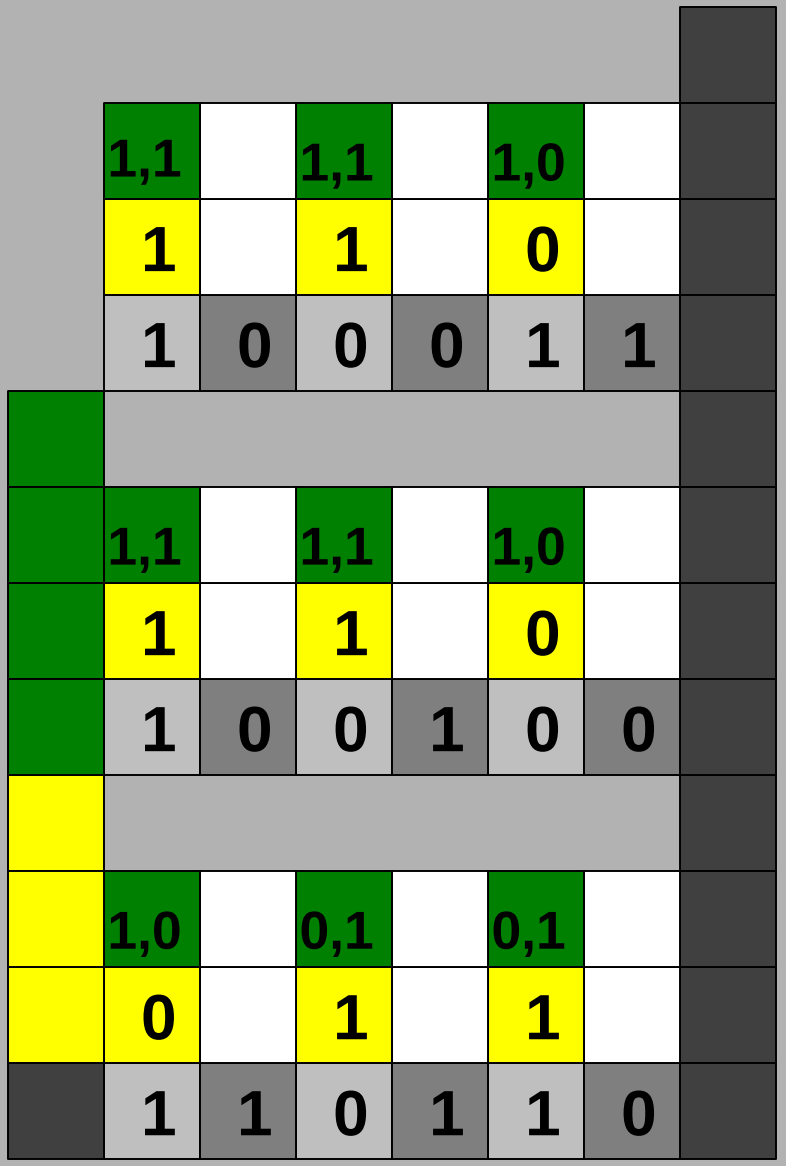
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row
1.2 Increment/copy

O(√n) **Phase 2:**
Grow carry-selector

Phase 3:



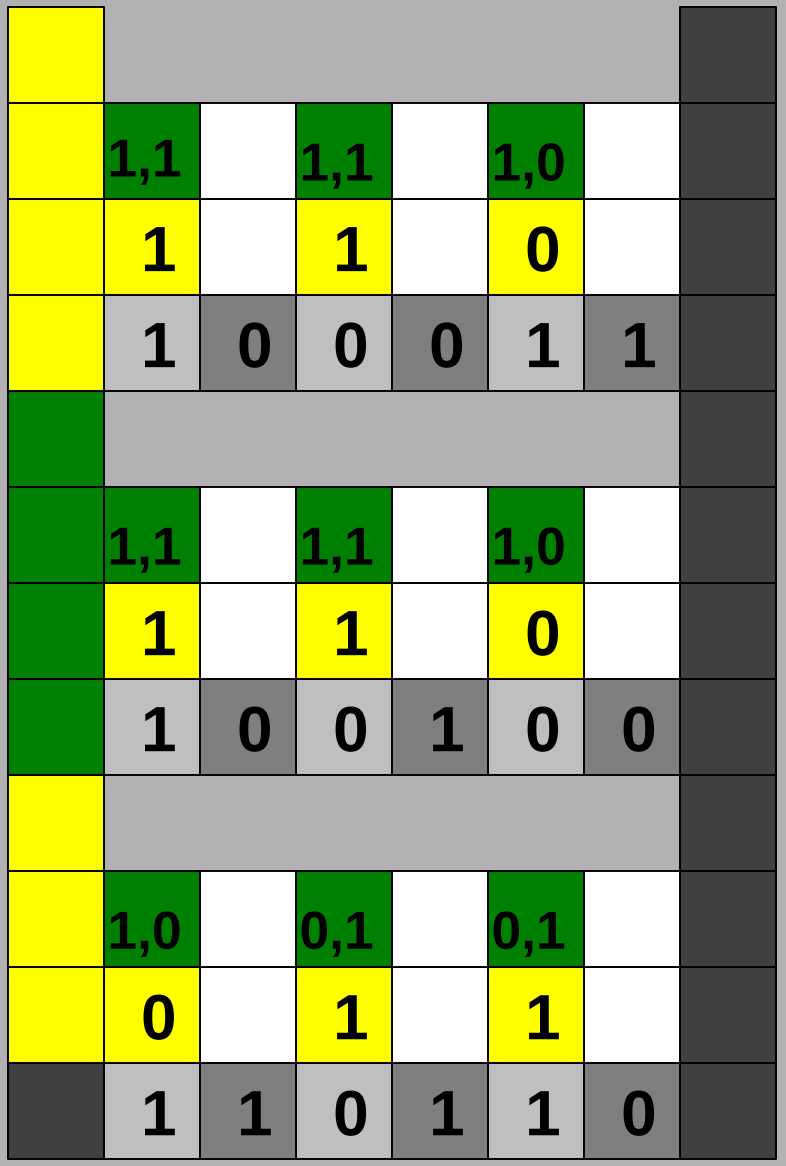
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row
1.2 Increment/copy

O(√n) **Phase 2:**
Grow carry-selector

O(√n) **Phase 3:**
Extract each row's output



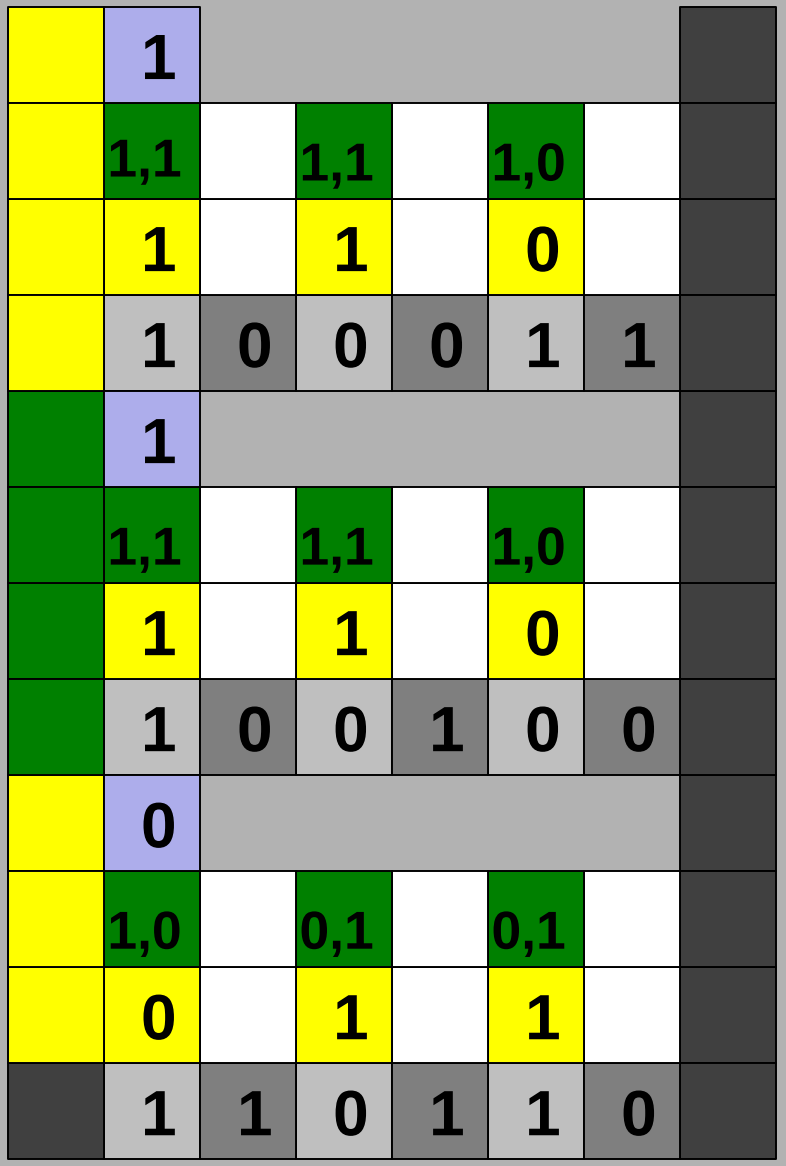
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row
1.2 Increment/copy

O(√n) **Phase 2:**
Grow carry-selector

O(√n) **Phase 3:**
Extract each row's output



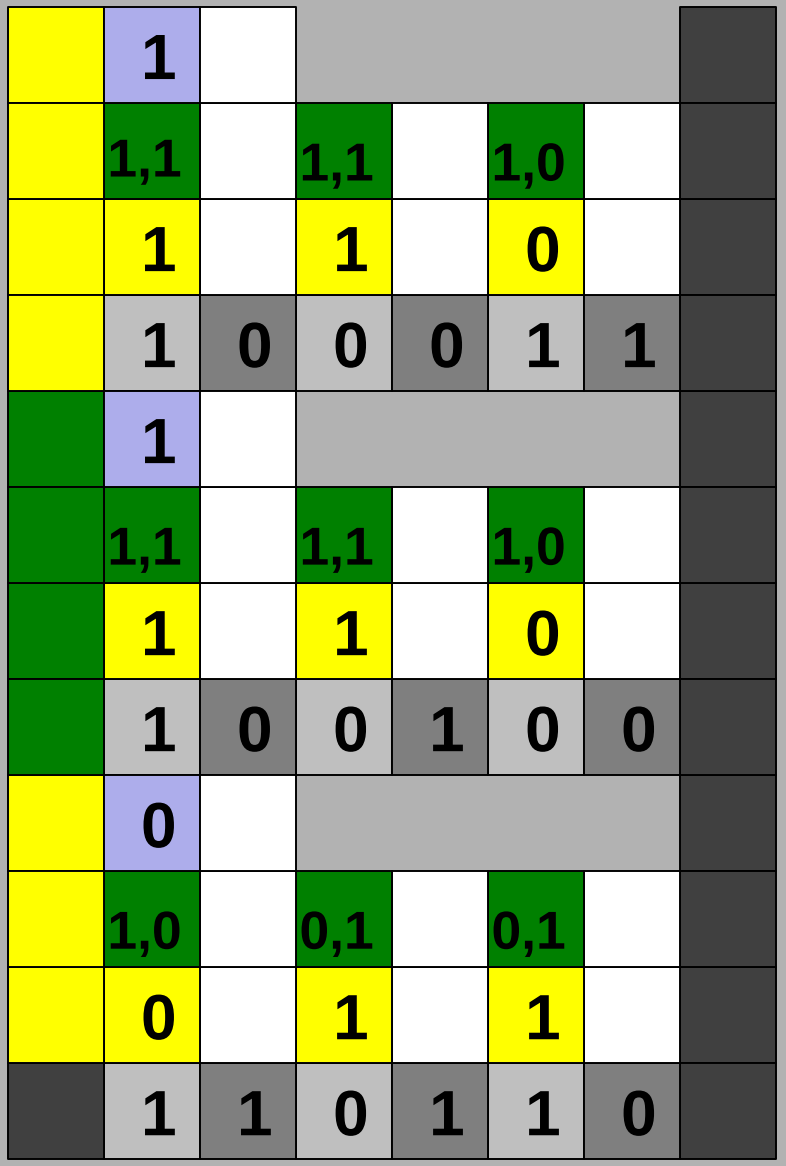
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row
1.2 Increment/copy

O(√n) **Phase 2:**
Grow carry-selector

O(√n) **Phase 3:**
Extract each row's output



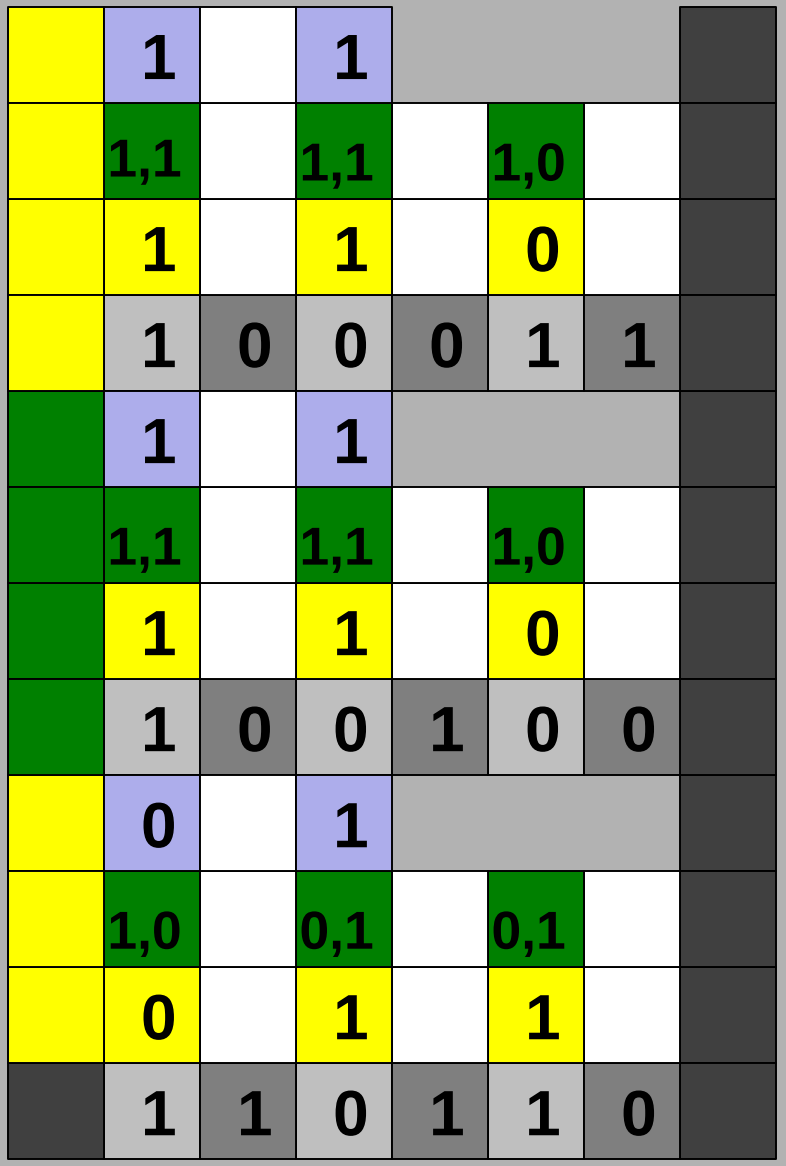
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row
1.2 Increment/copy

O(√n) **Phase 2:**
Grow carry-selector

O(√n) **Phase 3:**
Extract each row's output



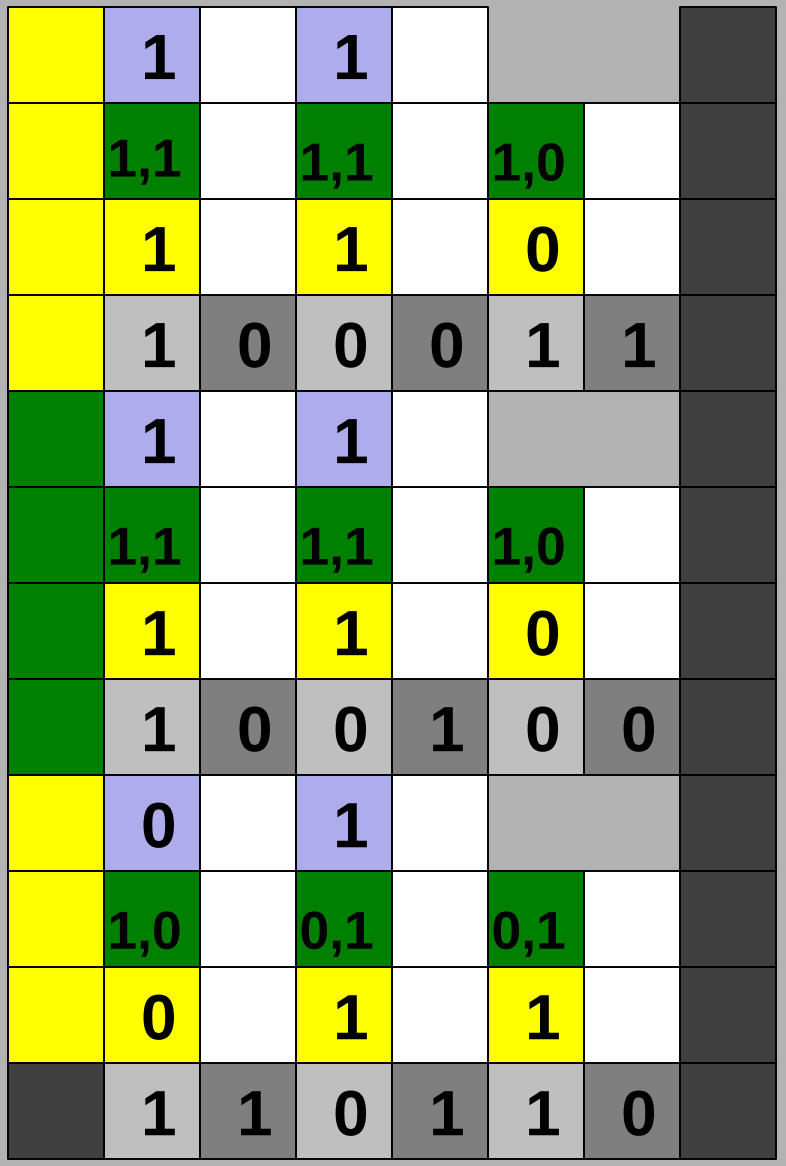
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row
1.2 Increment/copy

O(√n) **Phase 2:**
Grow carry-selector

O(√n) **Phase 3:**
Extract each row's output



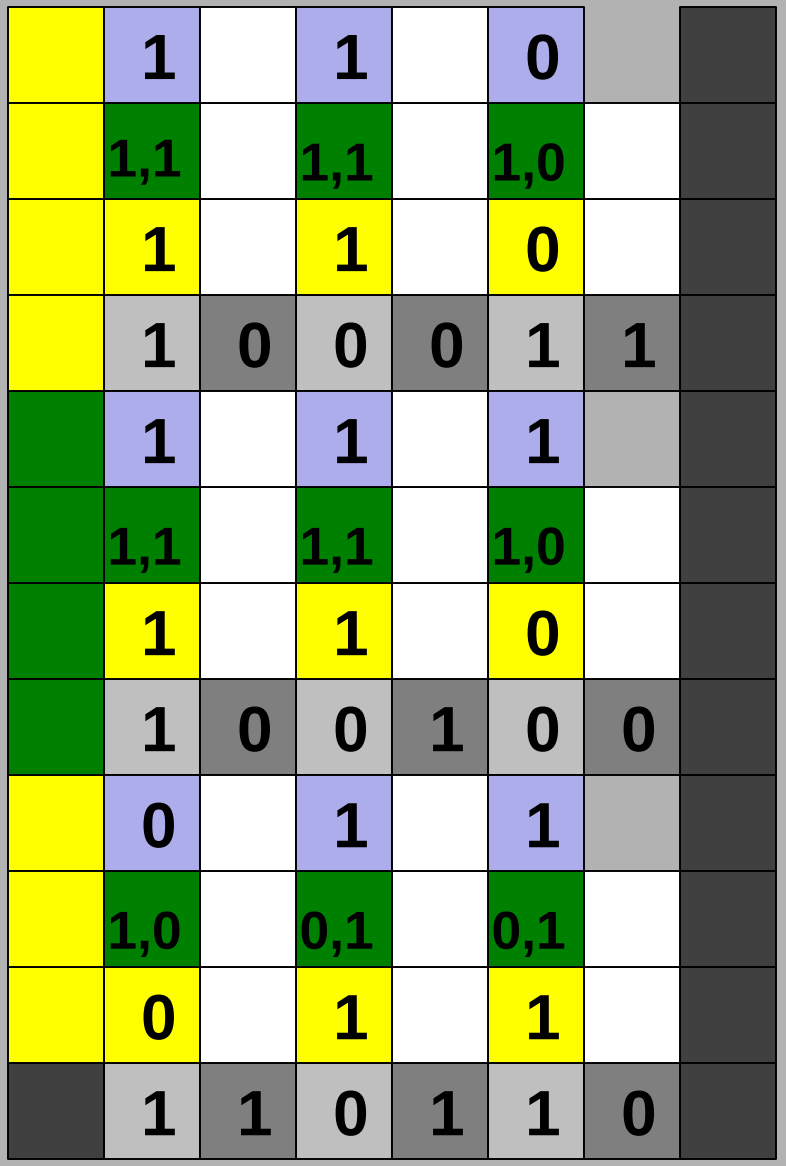
Addition

Faster **worst case**

O(√n) **Phase 1:**
1.1 Sum each row
1.2 Increment/copy

O(√n) **Phase 2:**
Grow carry-selector

O(√n) **Phase 3:**
Extract each row's output



Addition

Faster **worst case**

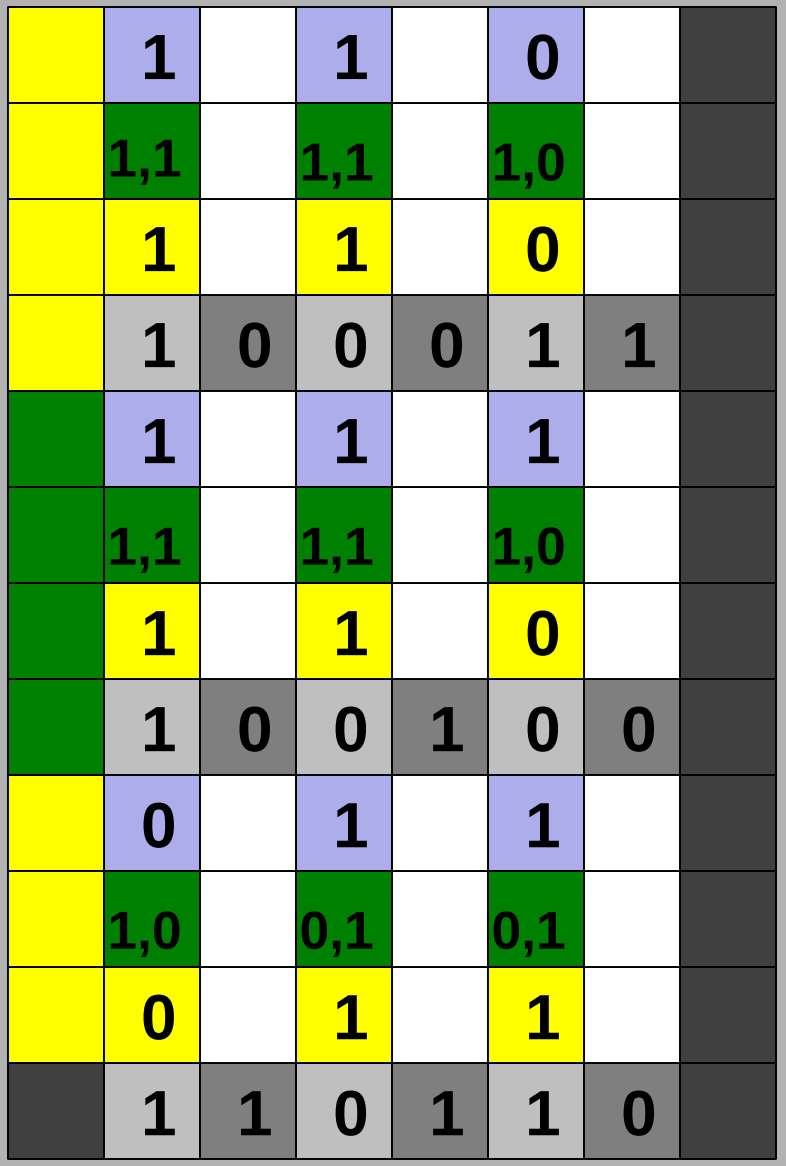
O(√n) **Phase 1:**
1.1 Sum each row
1.2 Increment/copy

O(√n) **Phase 2:**
Grow carry-selector

O(√n) **Phase 3:**
Extract each row's output

Total run-time:

O(√n)



	Addition Complexity	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm	O(n)	O(log n)
Worst-case algorithm	Θ(√n) optimal!	
Combo algorithm		
3D		

	Addition Complexity	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm	O(n)	O(log n)
Worst-case algorithm	Θ(√n) optimal!	
Combo algorithm		
3D		

Can we get a great average case and an optimal worst case?

	Addition Complexity	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm	O(n)	O(log n)
Worst-case algorithm	Θ(√n) optimal!	
Combo algorithm	Θ(√n) optimal!	O(log n)
3D		

Can we get a great average case and an optimal worst case?

-Yes.

	Addition Complexity	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm	O(n)	O(log n)
Worst-case algorithm	$\Theta(\sqrt{n})$ optimal!	
Combo algorithm	$\Theta(\sqrt{n})$ optimal!	O(log n)
3D		

Well, what about 3D?

-lower bound is $\Omega(n^{\frac{1}{3}})$

-can it be achieved?

	Addition Complexity	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm	O(n)	O(log n)
Worst-case algorithm	$\Theta(\sqrt{n})$ optimal!	
Combo algorithm	$\Theta(\sqrt{n})$ optimal!	O(log n)
3D	$\Theta(n^{1/3})$ optimal!	O(log n)

Well, what about 3D?

-lower bound is $\Omega(n^{\frac{1}{3}})$

-can it be achieved?

	Addition Complexity	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm	O(n)	O(log n)
Worst-case algorithm	$\Theta(\sqrt{n})$ optimal!	
Combo algorithm	$\Theta(\sqrt{n})$ optimal!	O(log n)
3D	$\Theta(n^{1/3})$ optimal!	O(log n)

Ok… what about multiplication?

	Addition Complexity	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm	O(n)	O(log n)
Worst-case algorithm	$\Theta(\sqrt{n})$ optimal!	
Combo algorithm	$\Theta(\sqrt{n})$ optimal!	O(log n)
3D	$\Theta(n^{1/3})$ optimal!	O(log n)

	Multiplication Complexity	
2D [Brun 2007]	O(n)	$\Omega(\sqrt{n})$
3D	$O(n^{5/6})$	$\Omega(n^{1/3})$

Open Questions

- Multiplication
  - Can you beat $O(n^{\frac{5}{6}})$ in 3D?
  - Can you beat O(n) in 2D?
  - Better lower bound for multiplication?
- Sorting
  - O(√n log n) in 3D
- Other metrics
  - Space used up
  - Modularity
- Computing other functions
  - This is just a jumping off point.

Thanks for listening. Questions?

Alexandra Keenan	Wyle Science Technology and Engineering Group
Robert Schweller	University of Texas - Pan American
Michael Sherman	Amazon
Xingsi Zhong	Clemson University

	Addition Complexity	
	Worst case upper lower	Average case
[Brun 2007]	O(n)	O(n)
Average-case algorithm	O(n)	O(log n)
Worst-case algorithm	$\Theta(\sqrt{n})$ optimal!	
Combo algorithm	$\Theta(\sqrt{n})$ optimal!	O(log n)
3D	$\Theta(n^{1/3})$ optimal!	O(log n)

	Multiplication Complexity	
2D [Brun 2007]	O(n)	$\Omega(\sqrt{n})$
3D	$O(n^{5/6})$	$\Omega(n^{1/3})$