Radix Sort

4	2	7	6	5	6	1

1	2	4	5	6	6	7

Radix Sort

4	
2	
7	
6	
5	
6	
1	

1	0	0
0	1	0
1	1	1
1	1	0
1	0	1
1	1	0
0	0	1

Radix Sort

4		1	0	*0*
2		0	1	*0*
7		1	1	*1*
6		1	1	*0*
5		1	0	*1*
6		1	1	*0*
1		0	0	*1*

Sort by least significant bit

Radix Sort

4		1	0	***0***
2		0	1	***0***
6		1	1	***0***
6		1	1	***0***
7		1	1	***1***
5		1	0	***1***
1		0	0	***1***

Sort by least significant bit

Radix Sort

4	1	***0***	0
2	0	***1***	0
6	1	***1***	0
6	1	***1***	0
7	1	***1***	1
5	1	***0***	1
1	0	***0***	1

Sort by second least significant bit

Radix Sort

4	1	***0***	0
5	1	***0***	1
1	0	***0***	1
2	0	***1***	0
6	1	***1***	0
6	1	***1***	0
7	1	***1***	1

Sort by second least significant bit

Radix Sort

4
5
1
2
6
6
7

1	0	0
1	0	1
0	0	1
0	1	0
1	1	0
1	1	0
1	1	1

Sort by third least significant bit

Radix Sort

1		*0*	0	1
2		*0*	1	0
4		*1*	0	0
5		*1*	0	1
6		*1*	1	0
6		*1*	1	0
7		*1*	1	1

Sort by third least significant bit

Parallel Radix Sort

4
2
7
6
5
6
1

1	0	***0***
0	1	***0***
1	1	***1***
1	1	***0***
1	0	***1***
1	1	***0***
0	0	***1***

Sort by least significant bit

Parallel Radix Sort

4	*0*
2	*0*
7	*1*
6	*0*
5	*1*
6	*0*
1	*1*

Extracting least significant bit can be done in parallel O(1) when p=n

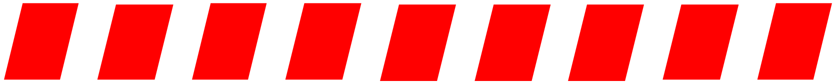
Parallel Radix Sort

		!
4	*0*	*1*
2	*0*	*1*
7	*1*	*0*
6	*0*	*1*
5	*1*	*0*
6	*0*	*1*
1	*1*	*0*
		!

Using not (!) we can “select” the ones with 0.

O(1) when p=n

Parallel Radix Sort



Parallel Radix Sort

4
2
7
6
5
6
1

!

0	*1*
0	*1*
1	*0*
0	*1*
1	*0*
0	*1*
1	*0*

!

We can calculate the new positions using scan (partial sums)
O(log(N)) when p=n

Parallel Radix Sort

New positions for bits with value 0

4
2
7
6
5
6
1

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

MAX

We can calculate the new positions using scan (partial sums)

O(log(N)) when p=n

Parallel Radix Sort

Parallel Radix Sort

New positions for bits with value 1

4
2
7
6
5
6
1

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

We can calculate the new positions using scan (partial sums) and adding the MAX new position from the 0 bits.

O(log(N)) when p=n

Parallel Radix Sort

Parallel Radix Sort

4
2
7
6
5
6
1

New positions for all

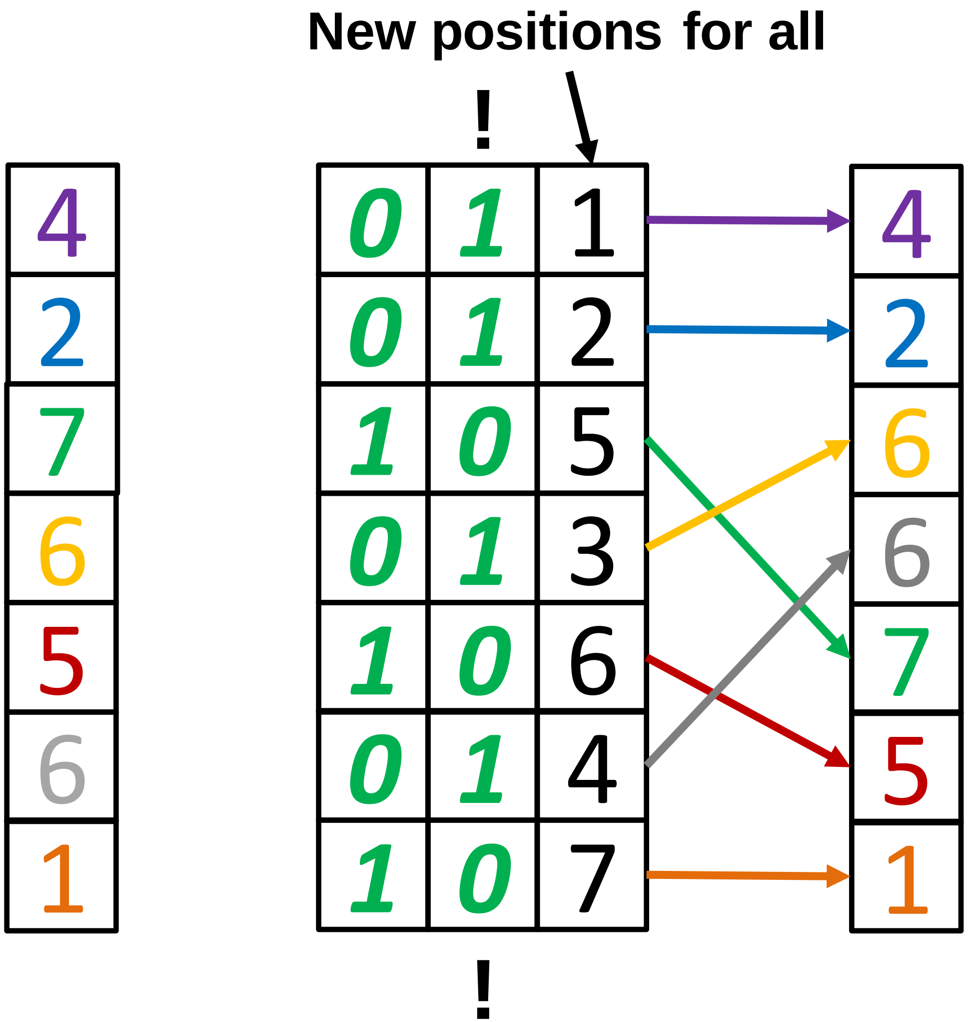
!

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

!

This step does not need to exit in the code. Smart uses of masks permits one to skip it.
However it is a good way to visualize our result.

Parallel Radix Sort



Move the values in the new positions.
O(1) when p=n

Radix Sort

4
2
6
6
7
5
1

1	0	***0***
0	1	***0***
1	1	***0***
1	1	***0***
1	1	***1***
1	0	***1***
0	0	***1***

The elements are now in correct positions as expected after the first step of sequential radix sort.

Parallel Radix Sort

Repeat all operations with second, then third, and so on, least significant bits.