Fenwick Tree

(Binary Indexed Tree)

William Fiset

Outline

- **Discussion & Examples**
  - Data structure motivation
  - What is a Fenwick tree?
  - Complexity analysis
- **Implementation details**
  - Range query
  - Point Updates
  - Fenwick tree construction
- **Code Implementation**

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

Sum of A from [2,7) = 6 + 1 + 0 + −4 + 11 = 14

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

Sum of A from [2,7) = 6 + 1 + 0 + −4 + 11 = 14

Sum of A from [0,4) = 5 + −3 + 6 + 1 = 9

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	**5**	**-3**	**6**	**1**	**0**	**-4**	**11**	**6**	**2**	**7**

Sum of A from [2,7) = 6 + 1 + 0 + −4 + 11 = 14

Sum of A from [0,4) = 5 + −3 + 6 + 1 = 9

Sum of A from [7,8) = 6 = 6

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	∅	∅	∅	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	∅	∅	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	∅	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	0	-4	11	6	2	7

P =	0	5	2	8	9	∅	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	9	9	∅	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	9	9	5	∅	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	9	9	5	16	∅	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	9	9	5	16	22	∅	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	9	9	5	16	22	24	∅

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	9	9	5	16	22	24	31

Let **P** be an array containing all the **prefix sums** of A.

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	9	9	5	16	22	24	31

Sum of A from [2,7) = P[7] - P[2] = 16 - 2 = 14

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	9	9	5	16	22	24	31

Sum of A from [2,7) = P[7] - P[2] = 16 - 2 = 14

Sum of A from [0,4) = P[4] - P[0] = 9 - 0 = 9

Fenwick Tree Motivation

Given an array of integer values compute the range sum between index [i, j).

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	9	9	5	16	22	24	31

Sum of A from [2,7) = P[7] - P[2] = 16 - 2 = 14

Sum of A from [0,4) = P[4] - P[0] = 9 - 0 = 9

Sum of A from [7,8) = P[8] - P[7] = 22 - 16 = 6

Fenwick Tree Motivation

Question: What about if we want to update our initial array with some new value?

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	0	-4	11	6	2	7	
P =	0	5	2	8	9	9	5	16	22	24	31

Fenwick Tree Motivation

Question: What about if we want to update our initial array with some new value?

	0	1	2	3	4	5	6	7	8	9
A =	5	-3	6	1	3	-4	11	6	2	7

P =	0	5	2	8	9	12	8	19	25	27	34

A[4] = 3

Fenwick Tree Motivation

Question: What about if we want to update our initial array with some new value?

	0	1	2	3	4	5	6	7	8	9	
A =	5	-3	6	1	3	-4	11	6	2	7	
P =	0	5	2	8	9	12	8	19	25	27	34

A prefix sum array is great for **static arrays**, but takes **O(n)** for updates.

What is a Fenwick Tree?

A **Fenwick Tree** (also called Binary Indexed Tree) is a data structure that supports sum range queries as well as setting values in a static array and getting the value of the prefix sum up some index efficiently.

Complexity

Construction	O(n)
Point Update	O(log(n))
Range Sum	O(log(n))
Range Update	O(log(n))
Adding Index	N/A
Removing Index	N/A

Fenwick Tree Range Queries

William Fiset

16	10000_2
15	01111_2
14	01110_2
13	01101_2
12	01100_2
11	01011_2
10	01010_2
9	01001_2
8	01000_2
7	00111_2
6	00110_2
5	00101_2
4	00100_2
3	00011_2
2	00010_2
1	00001_2

Unlike a regular array, in a Fenwick tree a specific cell is responsible for other cells as well.

16	10000_2
15	01111_2
14	01110_2
13	01101_2
12	01100_2
11	01011_2
10	01010_2
9	01001_2
8	01000_2
7	00111_2
6	00110_2
5	00101_2
4	00100_2
3	00011_2
2	00010_2
1	00001_2

Unlike a regular array, in a Fenwick tree a specific cell is responsible for other cells as well.

The position of the **least significant bit (LSB)** determines the range of responsibility that cell has to the cells below itself.

16	10000_2
15	01111_2
14	01110_2
13	01101_2
12	01100_2
11	01011_2
10	01010_2
9	01001_2
8	01000_2
7	00111_2
6	00110_2
5	00101_2
4	00100_2
3	00011_2
2	00010_2
1	00001_2

Unlike a regular array, in a Fenwick tree a specific cell is responsible for other cells as well.

The position of the **least significant bit (LSB)** determines the range of responsibility that cell has to the cells below itself.

Index 12 in binary is: 1100_2

LSB is at position 3, so this index is responsible for $2^{3-1} = 4$ cells below itself.

16	10000_2
15	01111_2
14	01110_2
13	01101_2
12	01100_2
11	01011_2
10	01010_2
9	01001_2
8	01000_2
7	00111_2
6	00110_2
5	00101_2
4	00100_2
3	00011_2
2	00010_2
1	00001_2

Unlike a regular array, in a Fenwick tree a specific cell is responsible for other cells as well.

The position of the **least significant bit (LSB)** determines the range of responsibility that cell has to the cells below itself.

Index 10 in binary is: 1010_2

LSB is at position 2, so this index is responsible for $2^{2-1} = 2$ cells below itself.

16	10000_2
15	01111_2
14	01110_2
13	01101_2
12	01100_2
11	01011_2
10	01010_2
9	01001_2
8	01000_2
7	00111_2
6	00110_2
5	00101_2
4	00100_2
3	00011_2
2	00010_2
1	00001_2

Unlike a regular array, in a Fenwick tree a specific cell is responsible for other cells as well.

The position of the **least significant bit (LSB)** determines the range of responsibility that cell has to the cells below itself.

Index 11 in binary is: 1011_2

LSB is at position 1, so this index is responsible for $2^{1-1} = 1$ cell (itself).

16	10000_2
15	01111_2
14	01110_2
13	01101_2
12	01100_2
11	01011_2
10	01010_2
9	01001_2
8	01000_2
7	00111_2
6	00110_2
5	00101_2
4	00100_2
3	00011_2
2	00010_2
1	00001_2

Blue bars represent the **range of responsibility** for that cell **NOT value**.

All odd numbers have a their first least significant bit set in the ones position, so they are only responsible for themselves.

Blue bars represent the **range of responsibility** for that cell **NOT value**.

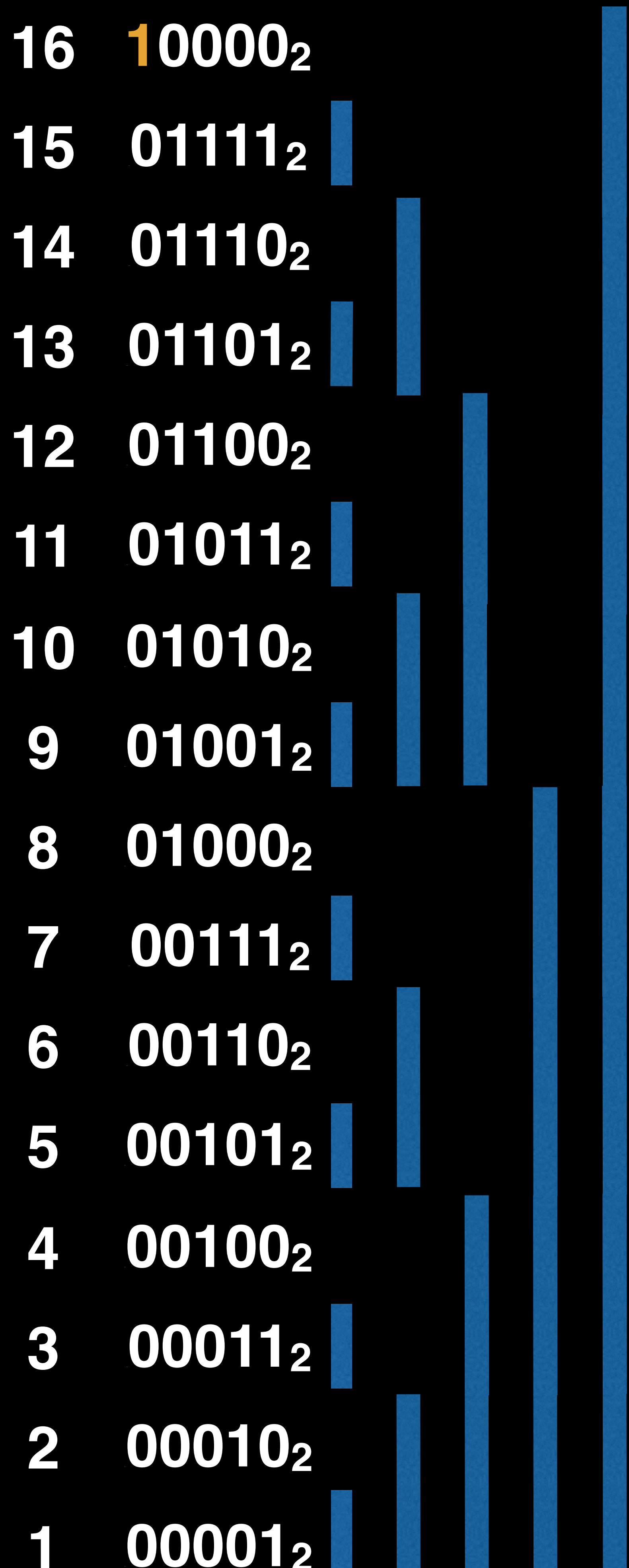
Numbers with their least significant bit in the second position have a range of two.

Blue bars represent the **range of responsibility** for that cell **NOT value**.

Numbers with their least significant bit in the third position have a range of four.

Blue bars represent the **range of responsibility** for that cell **NOT value**.

Numbers with their least significant bit in the fourth position have a range of eight.

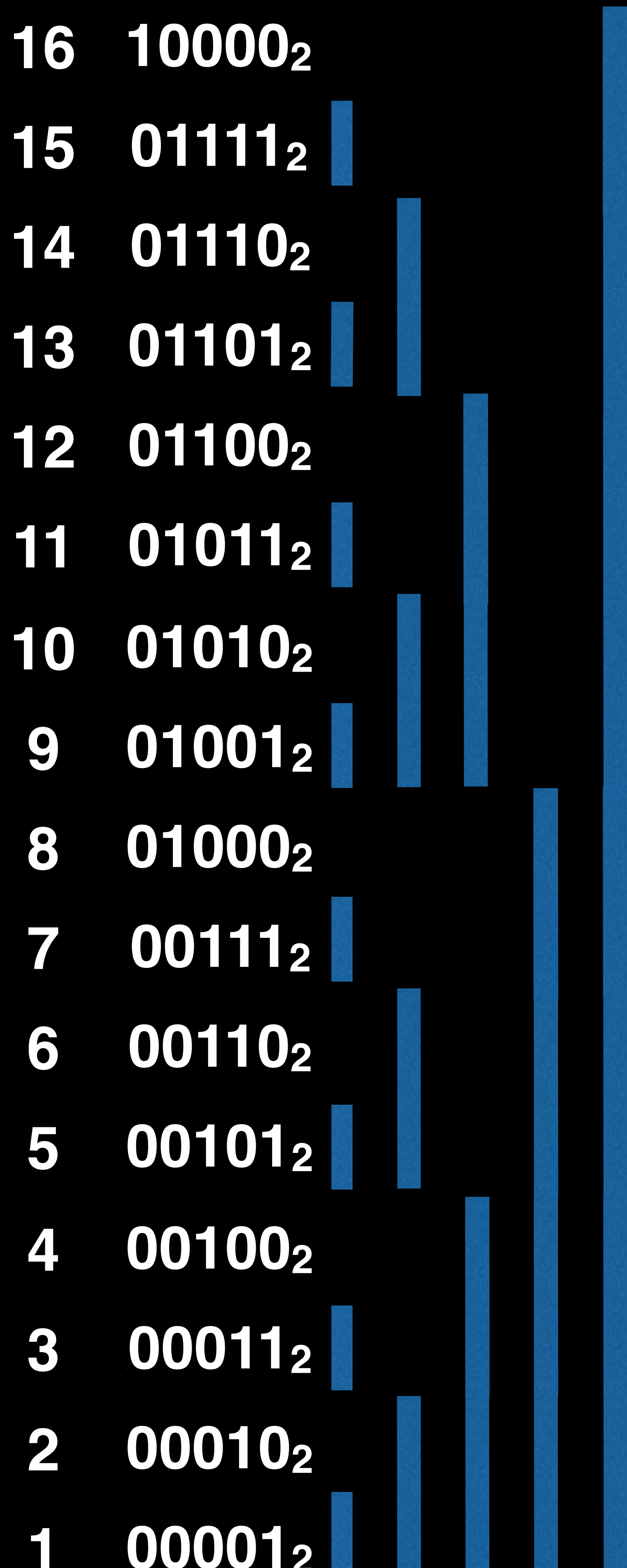


Blue bars represent the **range of responsibility** for that cell **NOT value**.

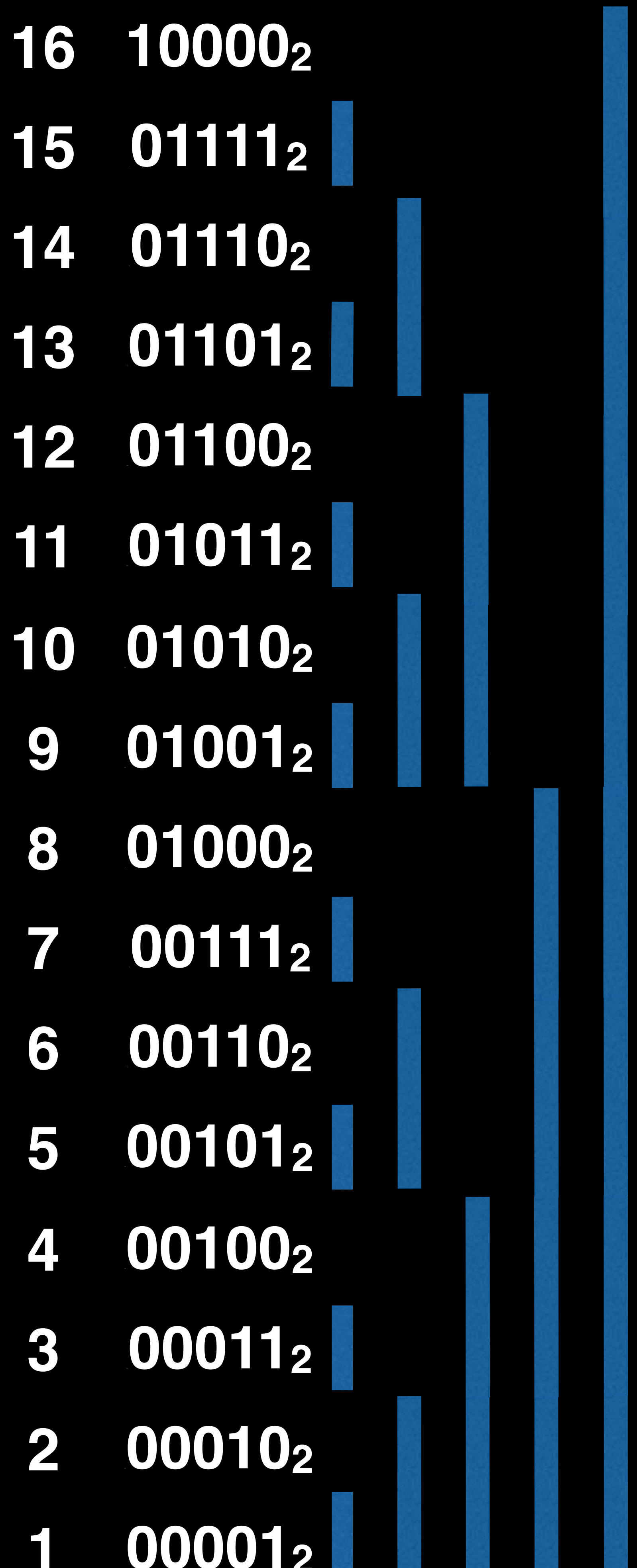
Numbers with their least significant bit in the fifth position have a range of sixteen.

Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

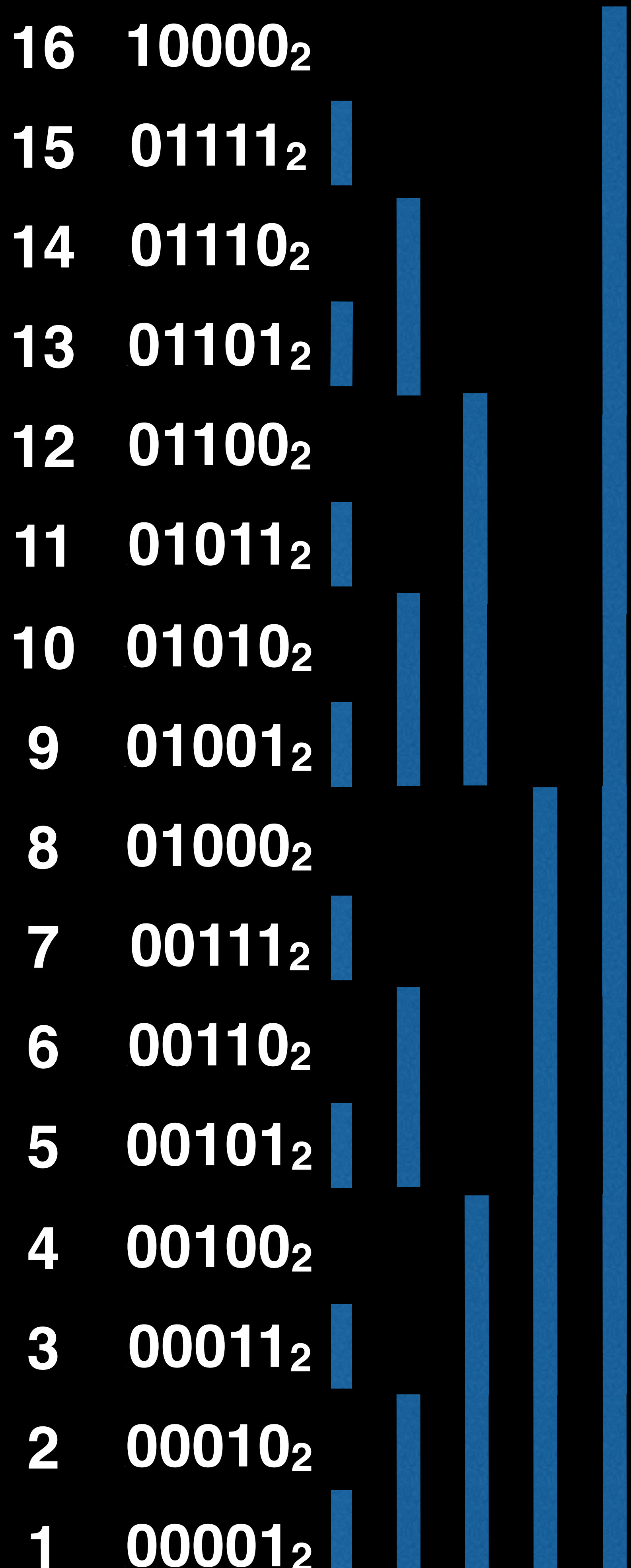


Range Queries



In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Idea: Suppose you want to find the prefix sum of [1, i], then you **start at i and cascade downwards** until you reach zero adding the value at each of the indices you encounter.



Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 7.

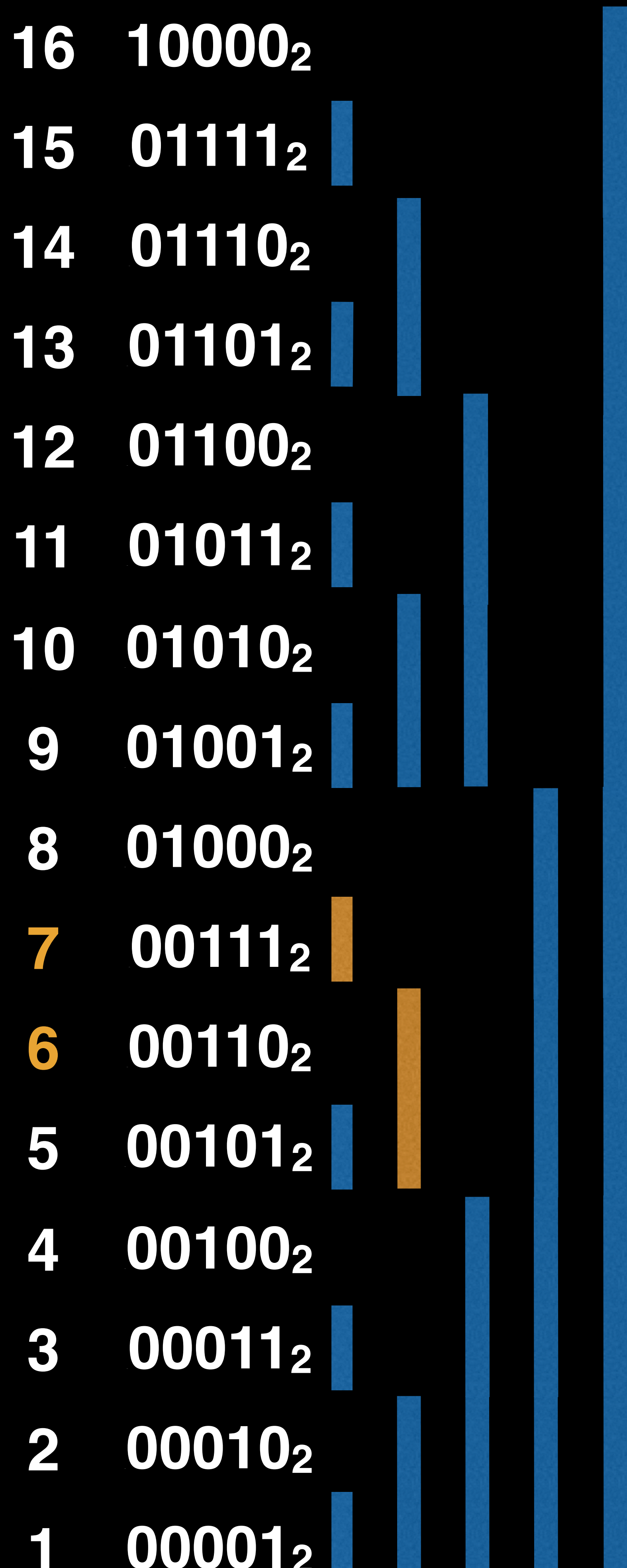
Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 7.

sum = A[7]

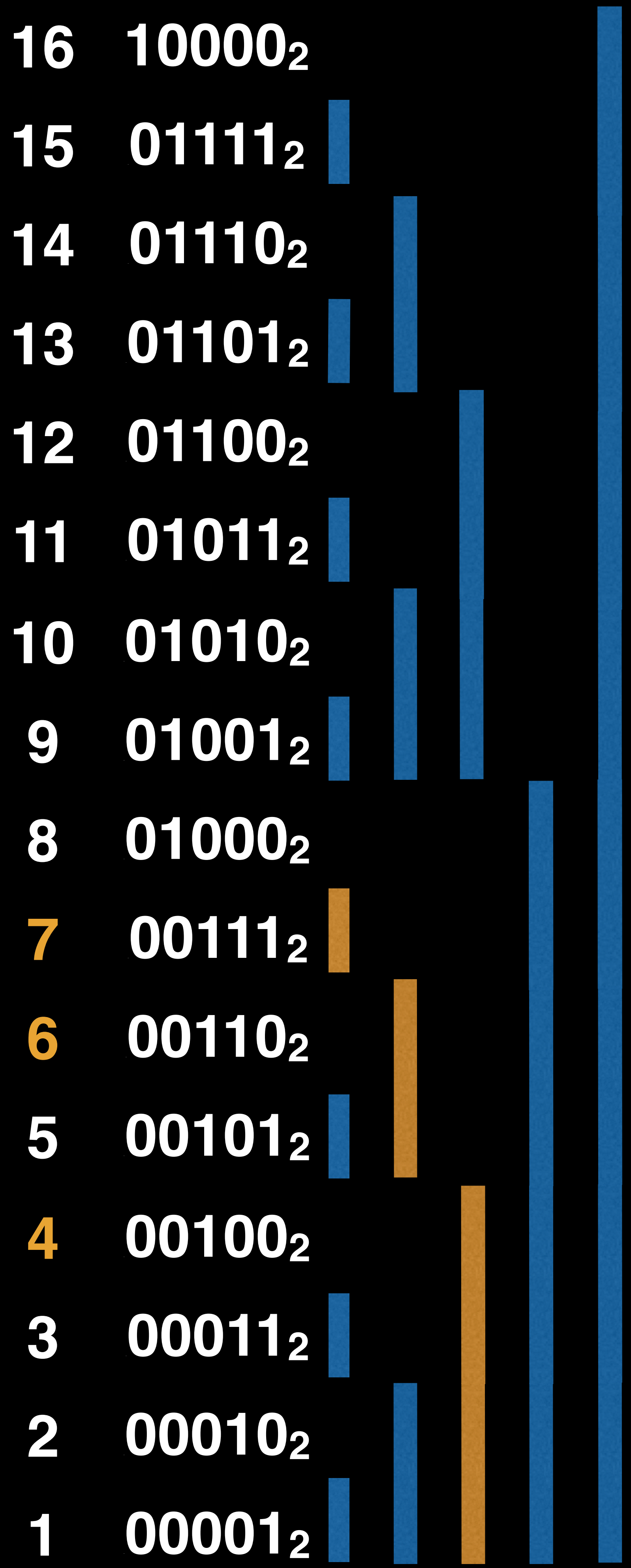
Range Queries



In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 7.

sum = A[7] + A[6]

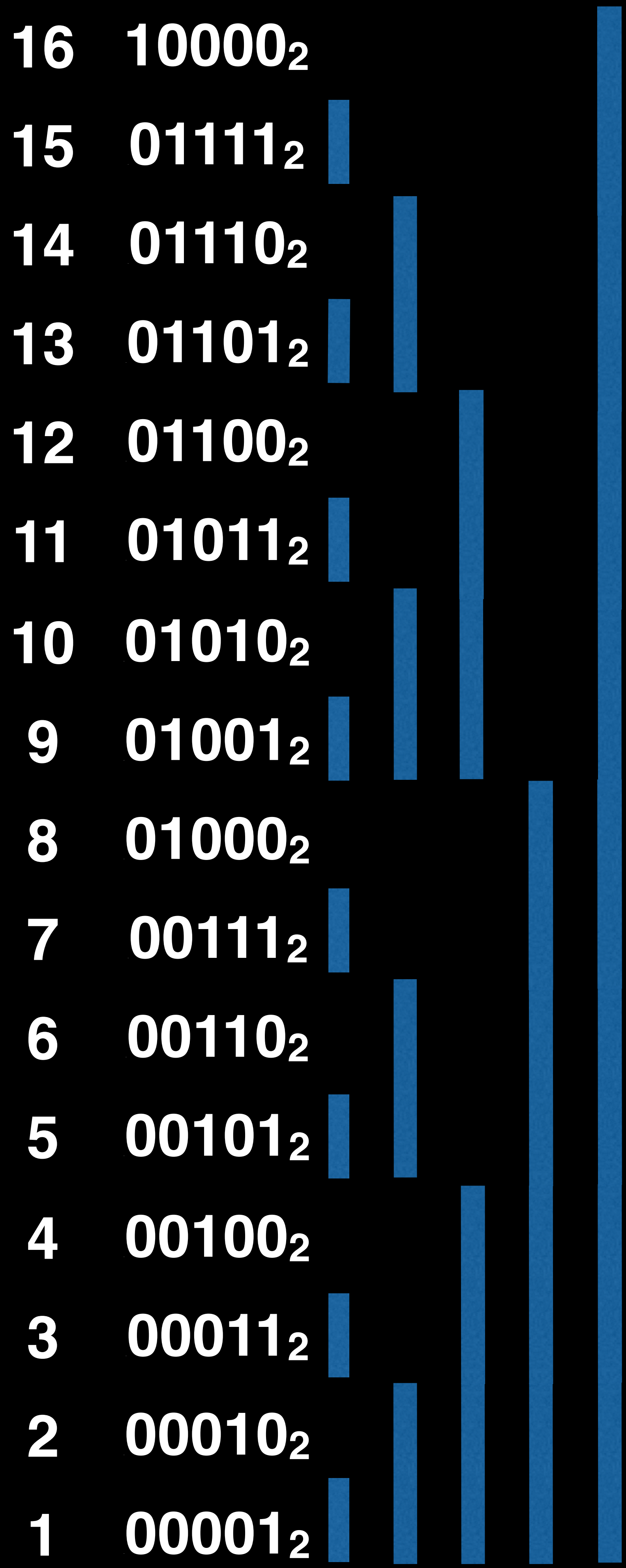


Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 7.

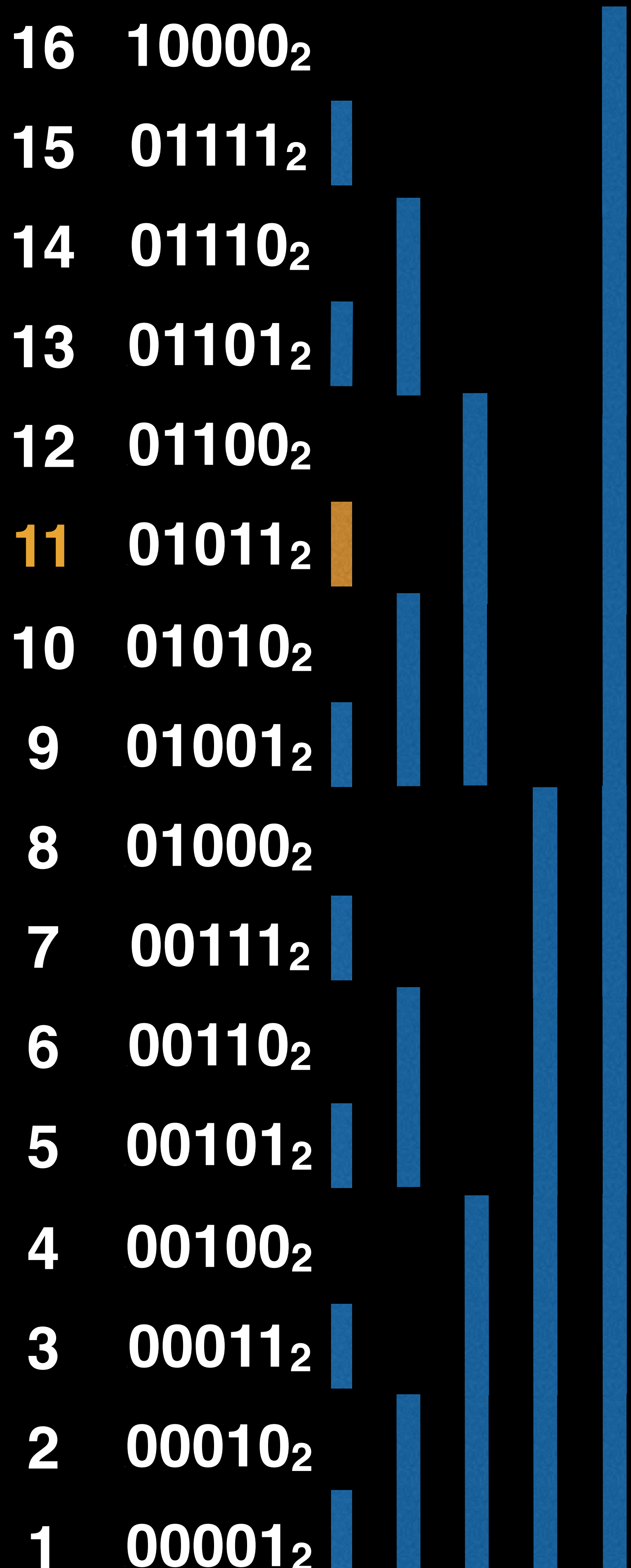
sum = A[7] + A[6] + A[4]



Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 11.

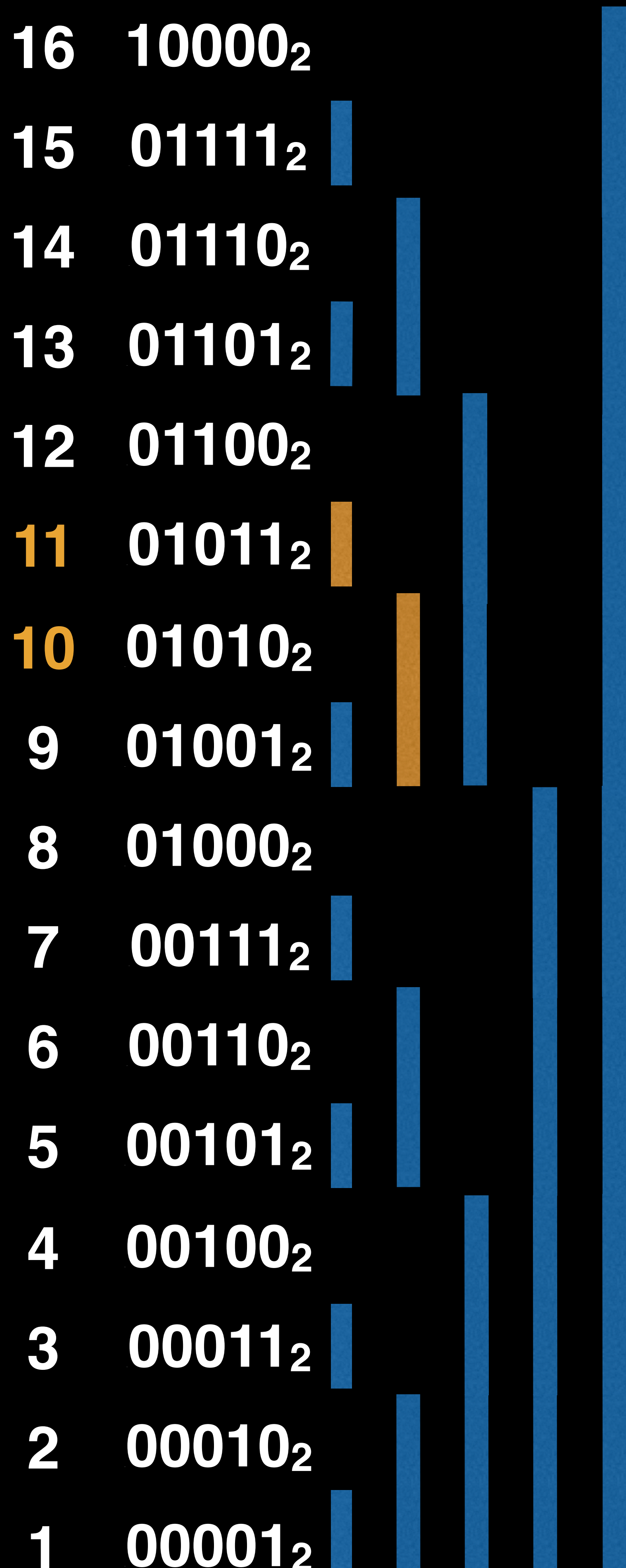


Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 11.

sum = A[11]

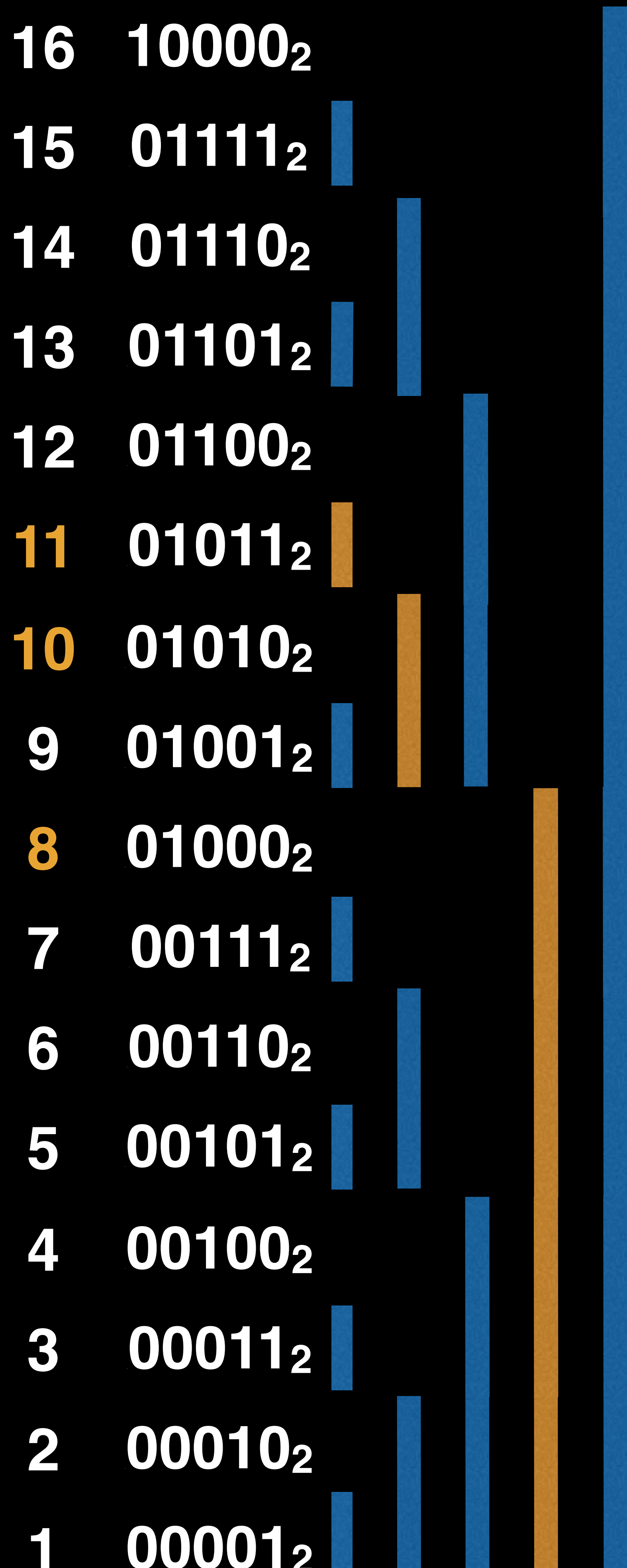


Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 11.

sum = A[11] + A[10]

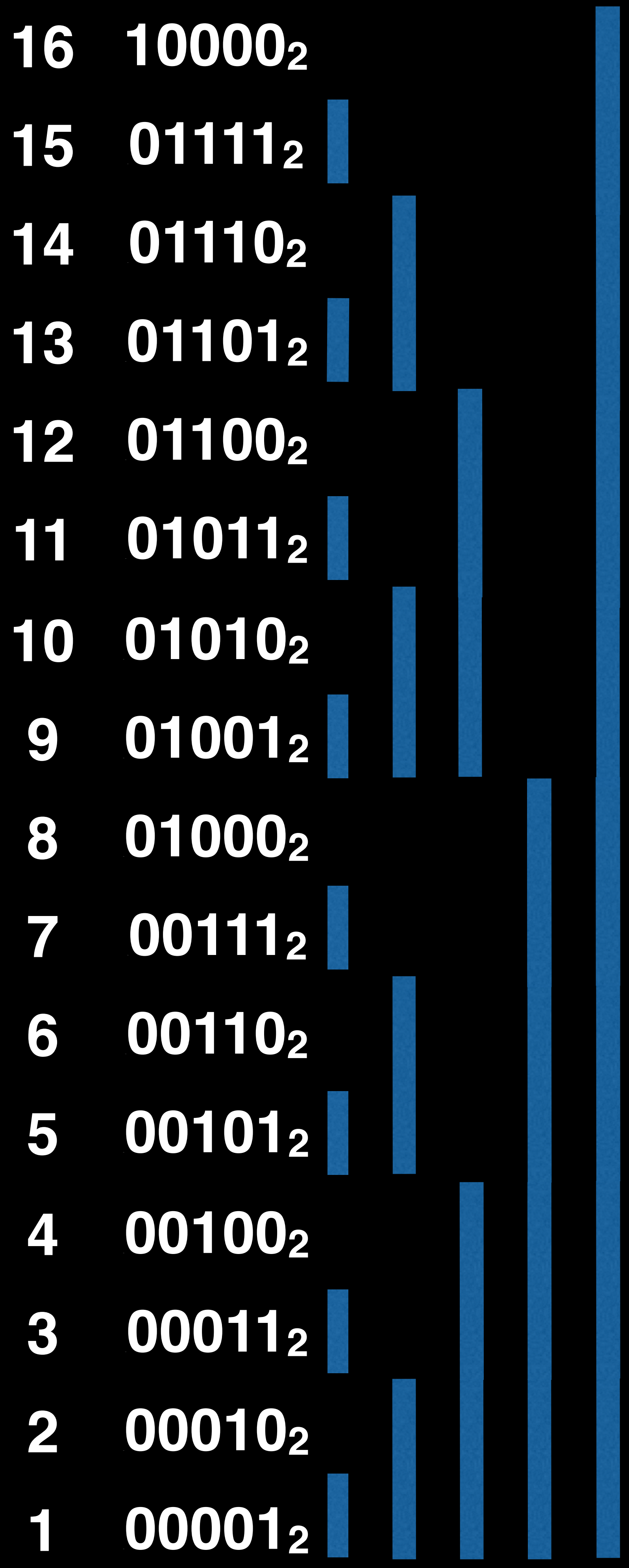


Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 11.

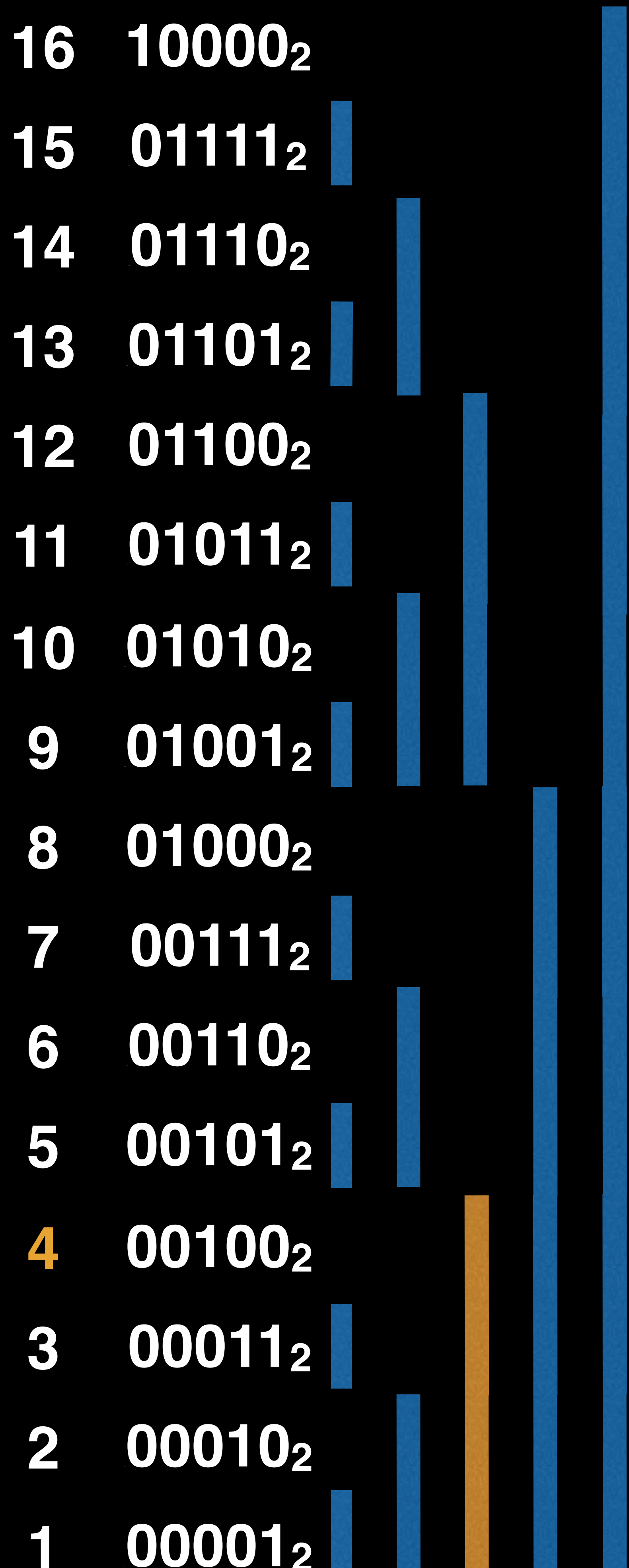
sum = A[11] + A[10] + A[8]



Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

Find the prefix sum up to index 4.

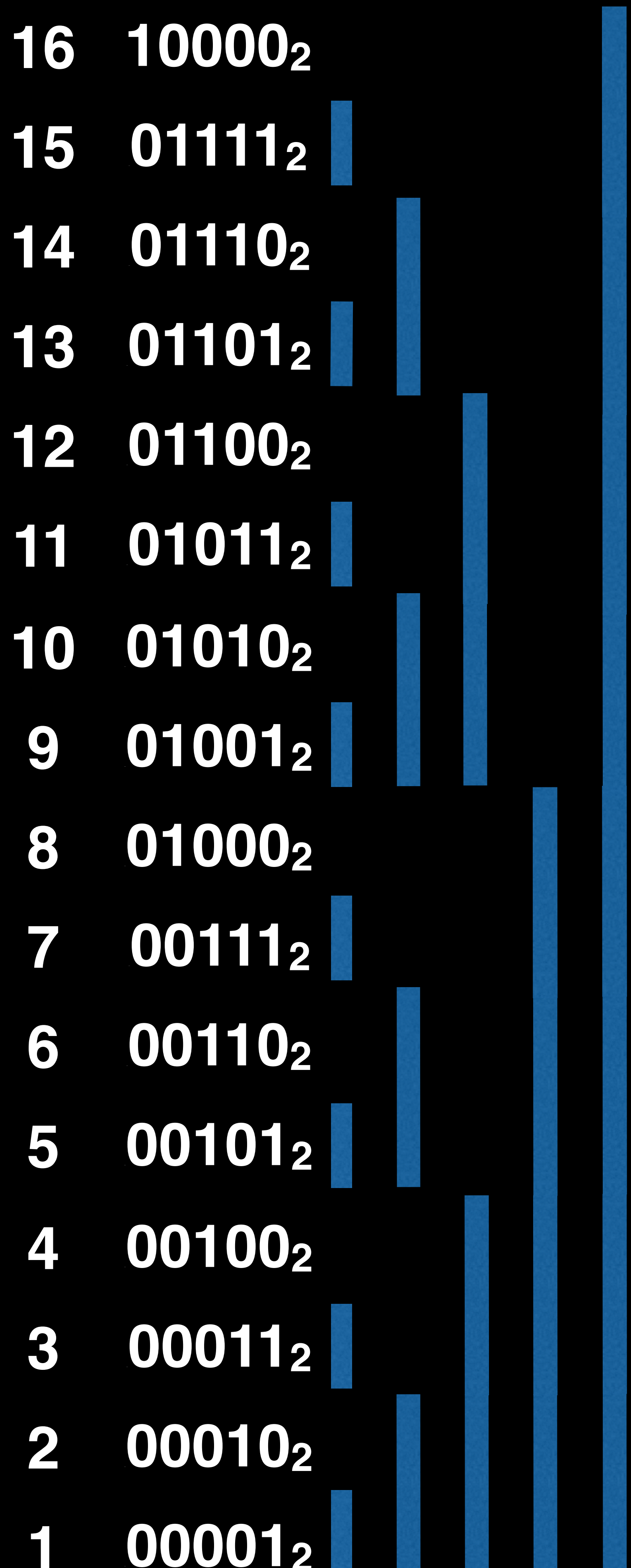


Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

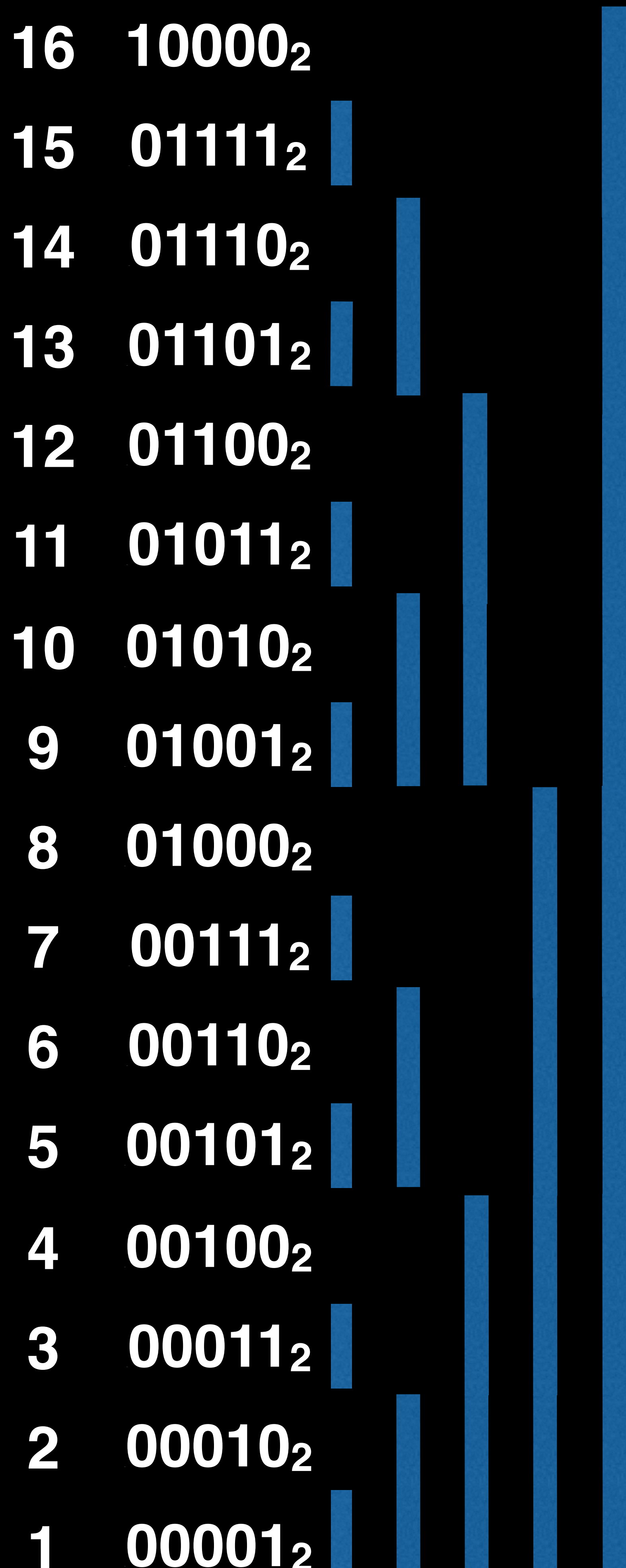
Find the prefix sum up to index 4.

sum = A[4]



Range Queries

In a Fenwick tree we may compute the **prefix sum** up to a certain index, which ultimately lets us perform range sum queries.

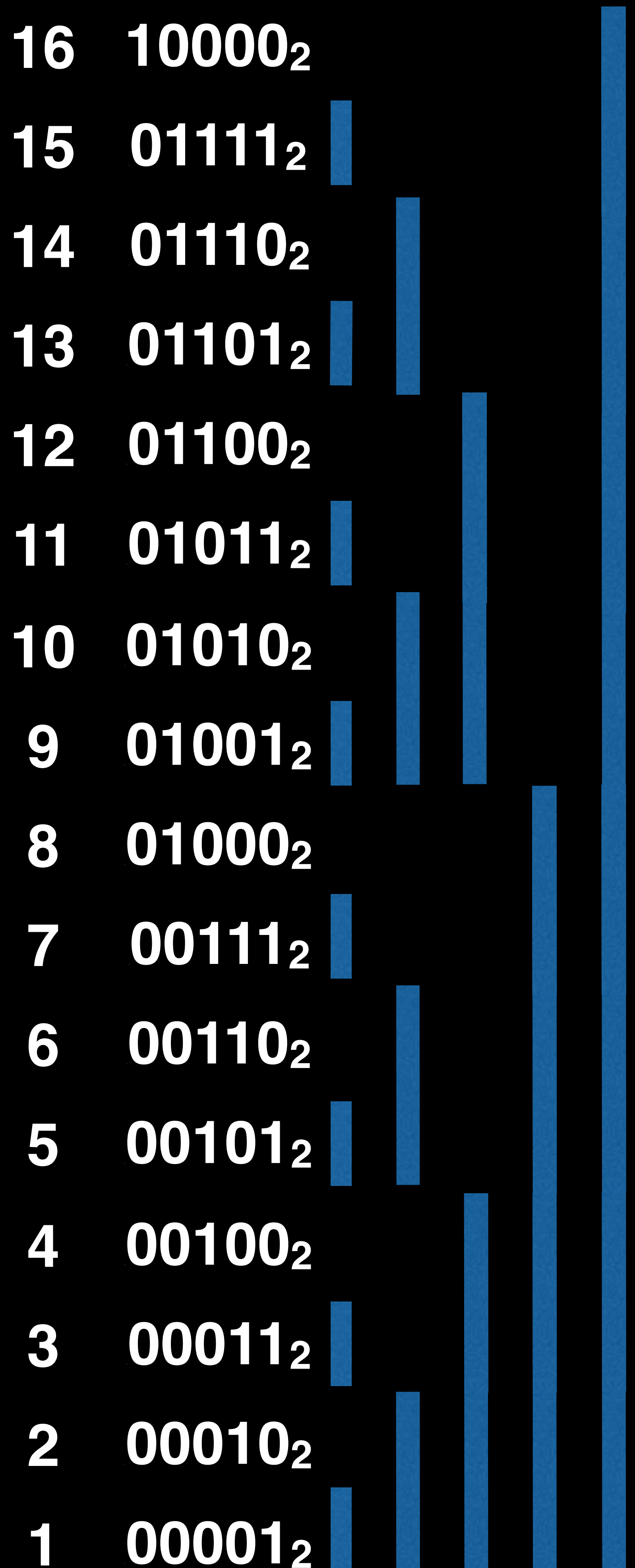


Range Queries

Let’s use prefix sums to compute the interval sum between [i, j].

Compute the interval sum between [11, 15].

Range Queries

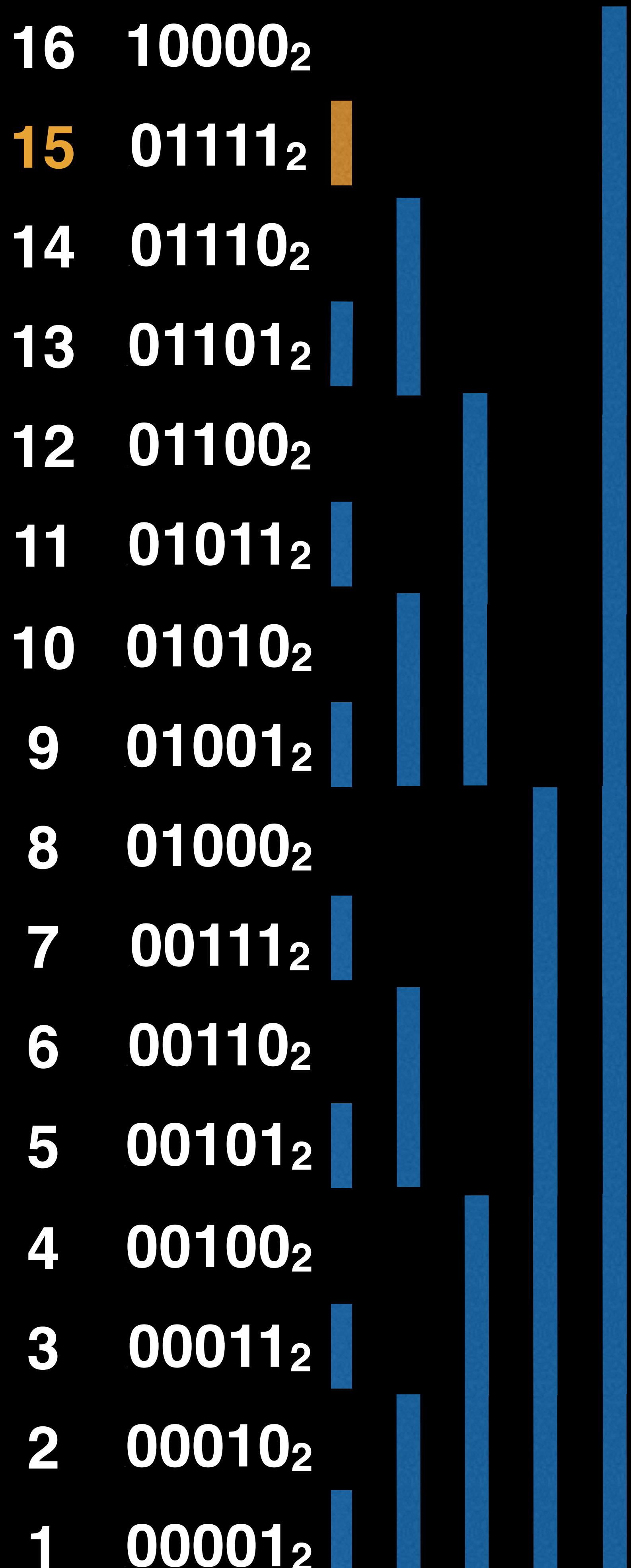


Let's use prefix sums to compute the interval sum between [i, j].

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

Not inclusive! We want the value at position 11.



Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

Sum of [1,15] = A[15]

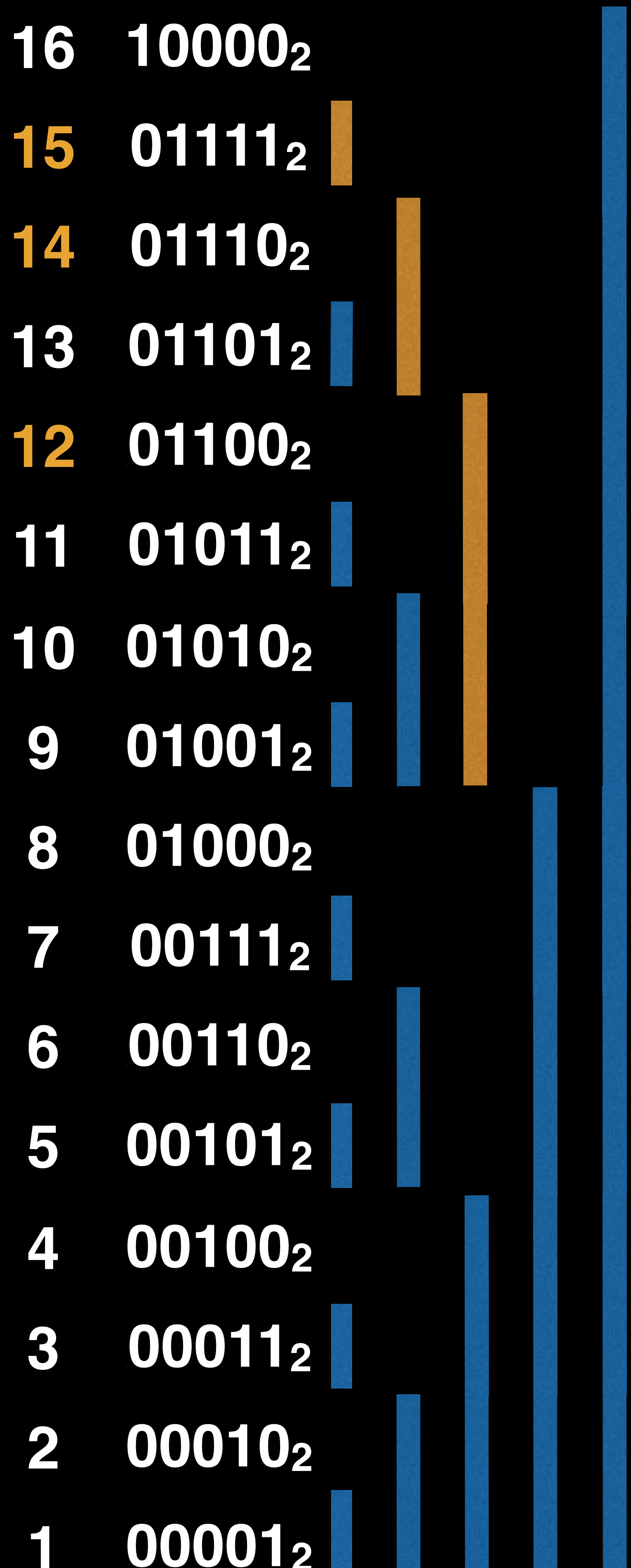
Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

Sum of [1,15] = A[15]+A[14]

Range Queries



Compute the interval sum between [11, 15].

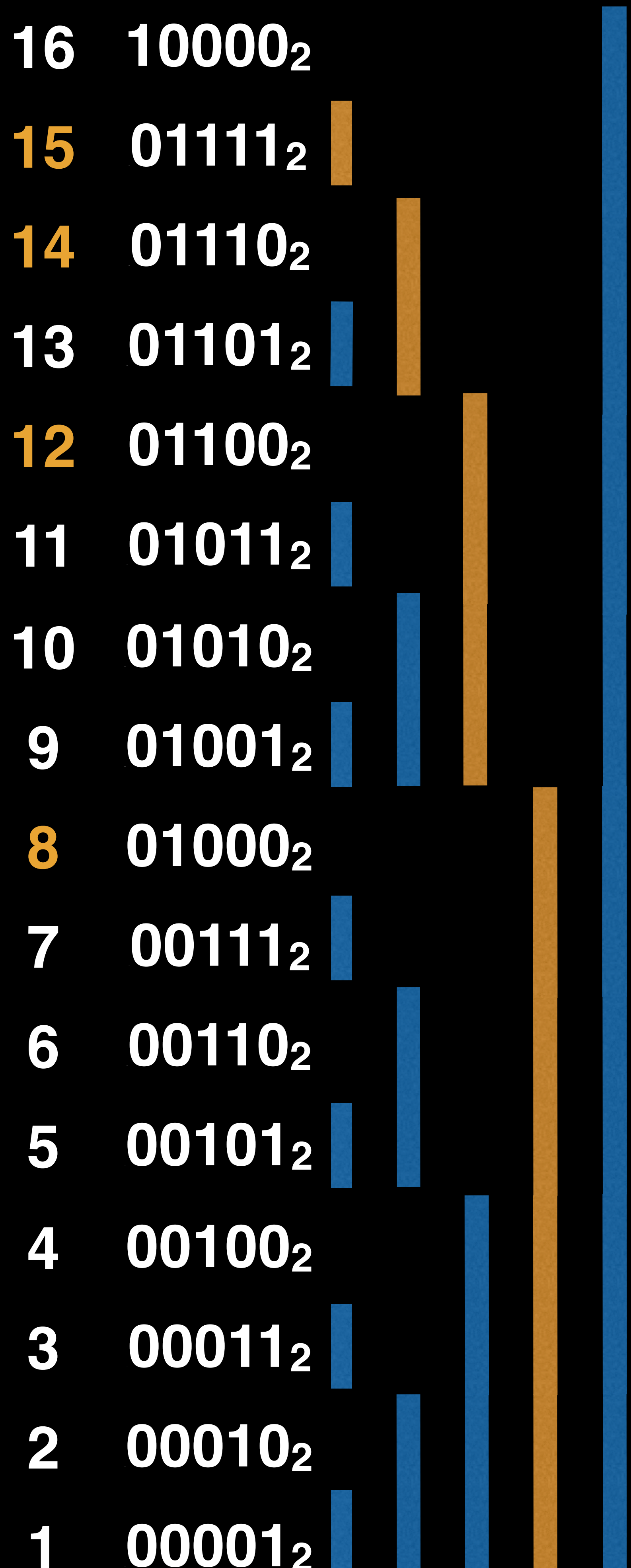
First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

Sum of [1,15] = A[15]+A[14]+A[12]

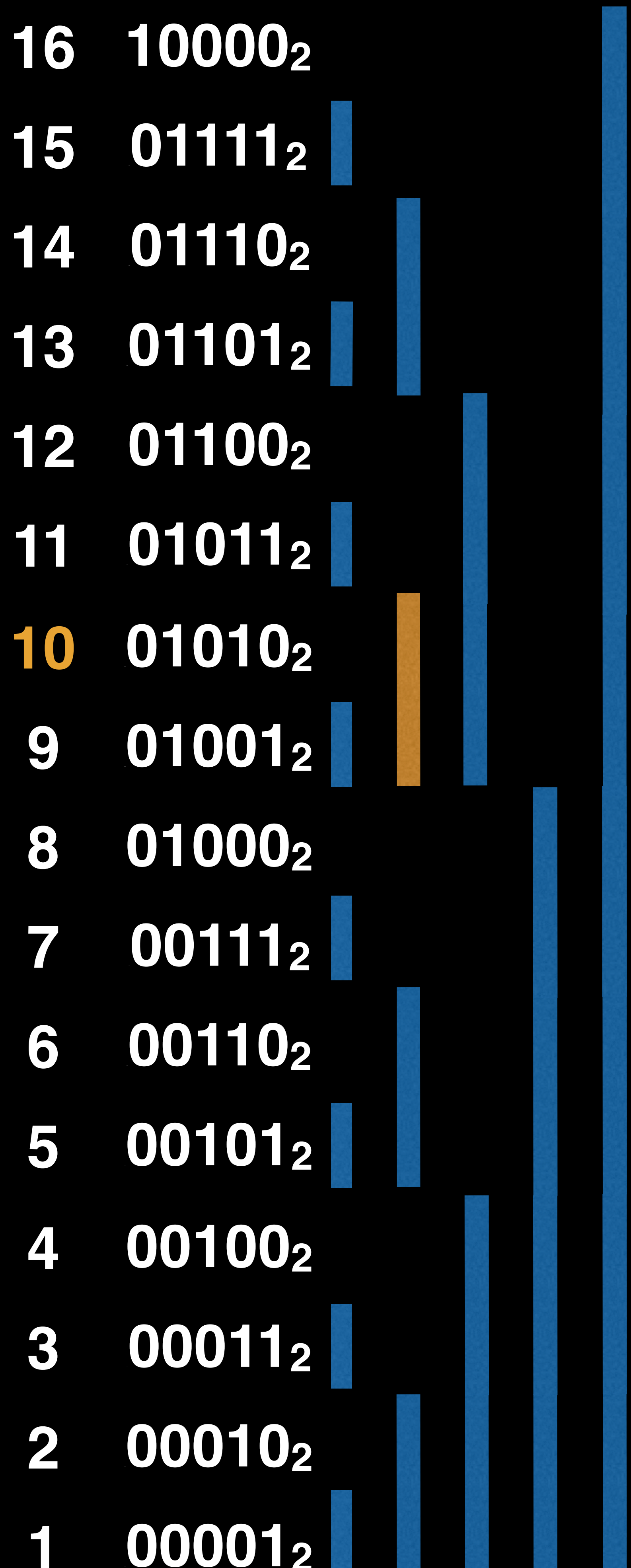
Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.



Sum of [1,15] = A[15]+A[14]+A[12]+A[8]



Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

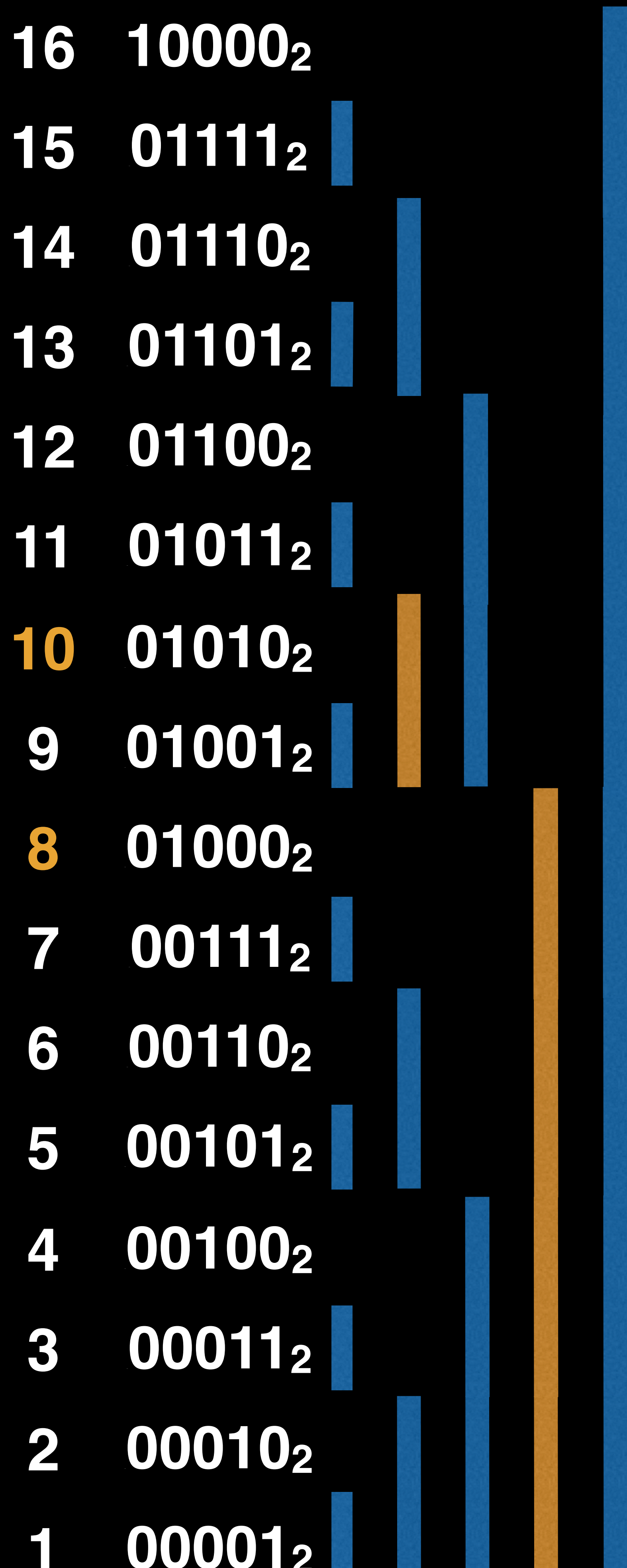
Sum of [1,15] = A[15]+A[14]+A[12]+A[8]

Sum of [1,11) = A[10]

Range Queries

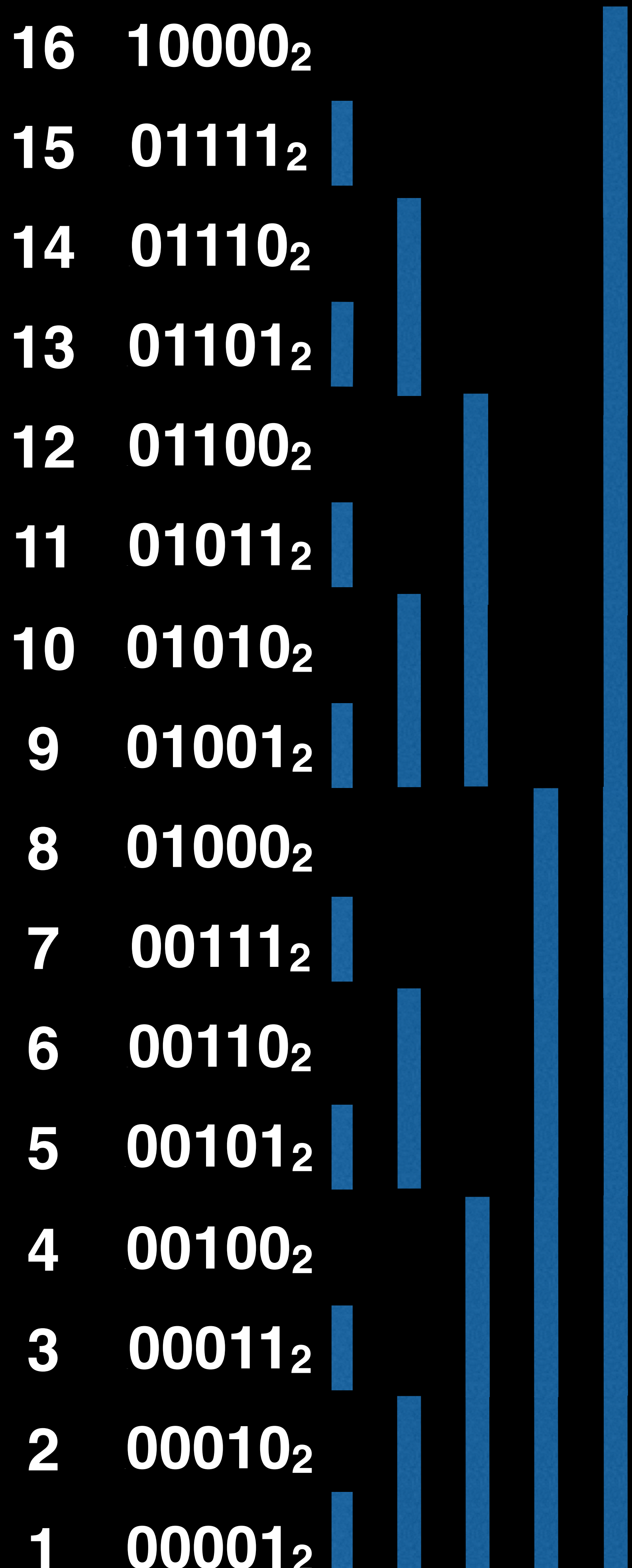
Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.



Sum of [1,15] = A[15]+A[14]+A[12]+A[8]

Sum of [1,11) = A[10]+A[8]



Range Queries

Compute the interval sum between [11, 15].

First we compute the prefix sum of [1, 15], then we will compute the prefix sum of [1,11) and get the difference.

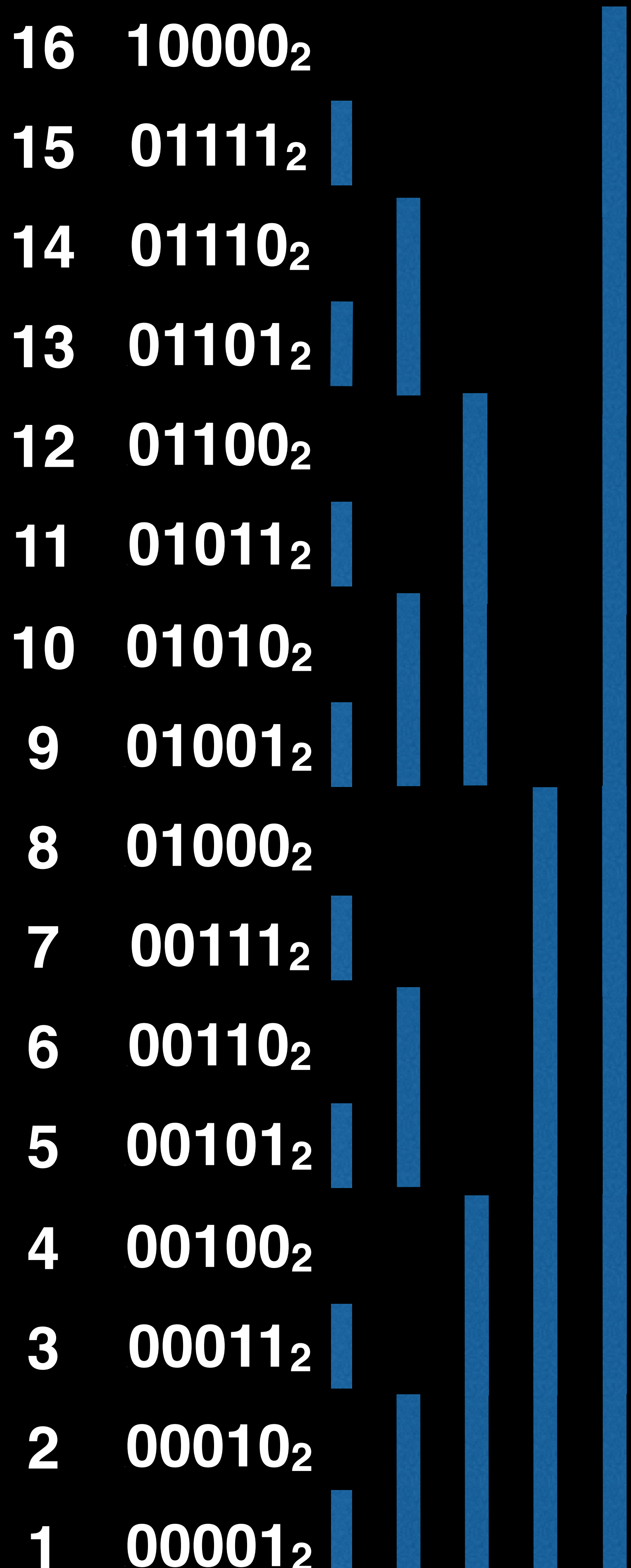
Sum of [1,15] = A[15]+A[14]+A[12]+A[8]

Sum of [1,11) = A[10]+A[8]

Range sum:

(A[15]+A[14]+A[12]+A[8])−(A[10]+A[8])

Range Queries



Notice that in the worst case the cell we're querying has a binary representation of all ones (numbers of the form 2^n-1)

Hence, it's easy to see that in the worst case a range query might make us have to do two queries that cost $\log_2(n)$ operations.

Range query algorithm

To do a range query from [i,j] both inclusive a Fenwick tree of size N:

```
function prefixSum(i):
    sum := 0
    while i != 0:
        sum = sum + tree[i]
        i = i - LSB(i)
    return sum


function rangeQuery(i, j):
    return prefixSum(j) - prefixSum(i-1)
```

Where **LSB** returns the value of the least significant bit.

next video: Fenwick Tree point updates!

Implementation source code and tests can all be found at the following link:

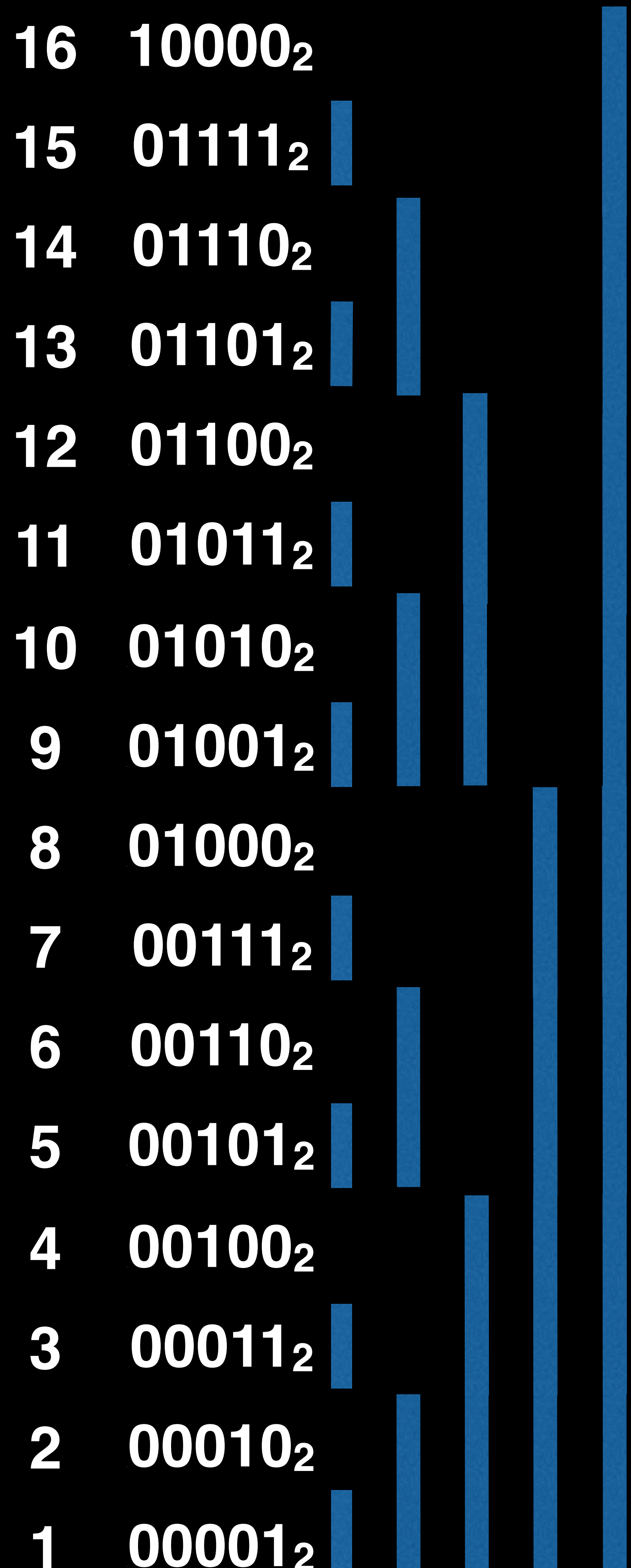
github.com/williamfiset/data-structures

Fenwick Tree Point Updates

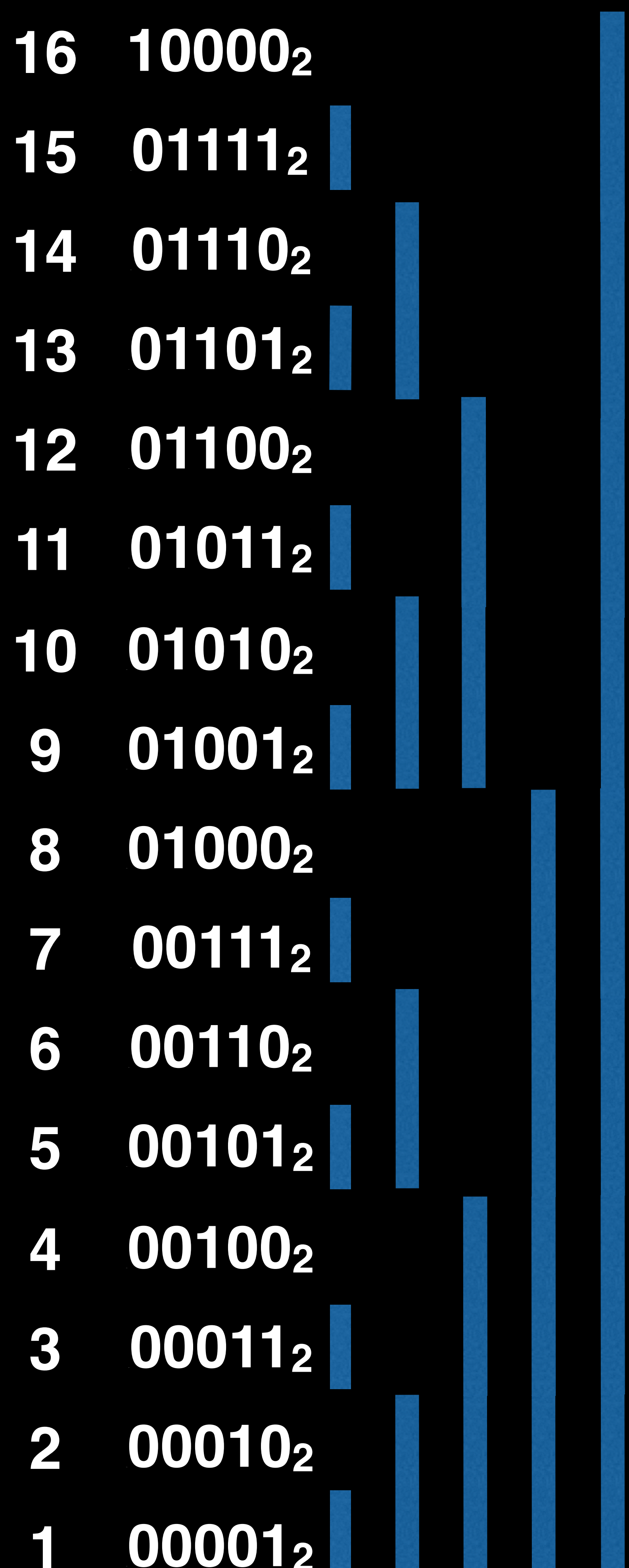
William Fiset

Last Video: Fenwick Tree Range Queries

Point Updates



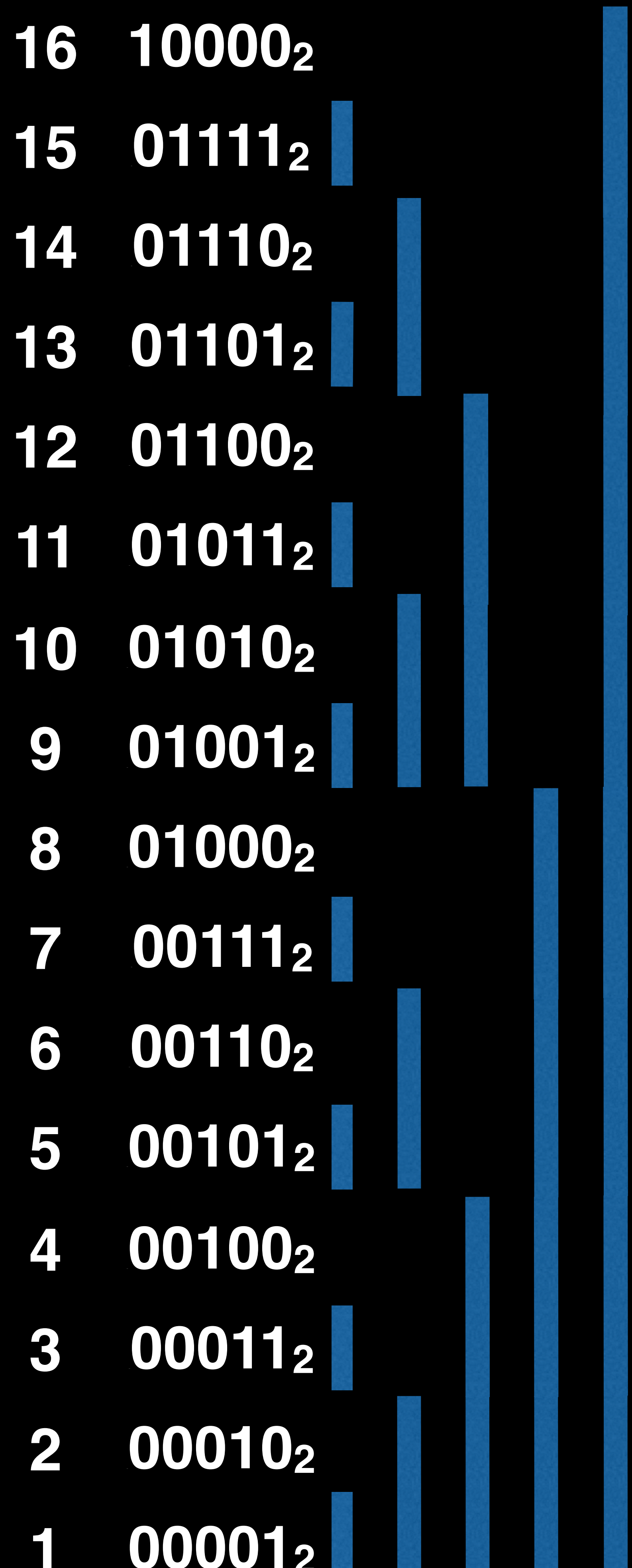
Instead of querying a range to find the interval sum we want to update a cell in our array.



Point Updates

Instead of querying a range to find the interval sum we want to update a cell in our array.

Recall that with range queries we cascaded down from the current index by **continuously removing the LSB.**



Point Updates

Instead of querying a range to find the interval sum we want to update a cell in our array.

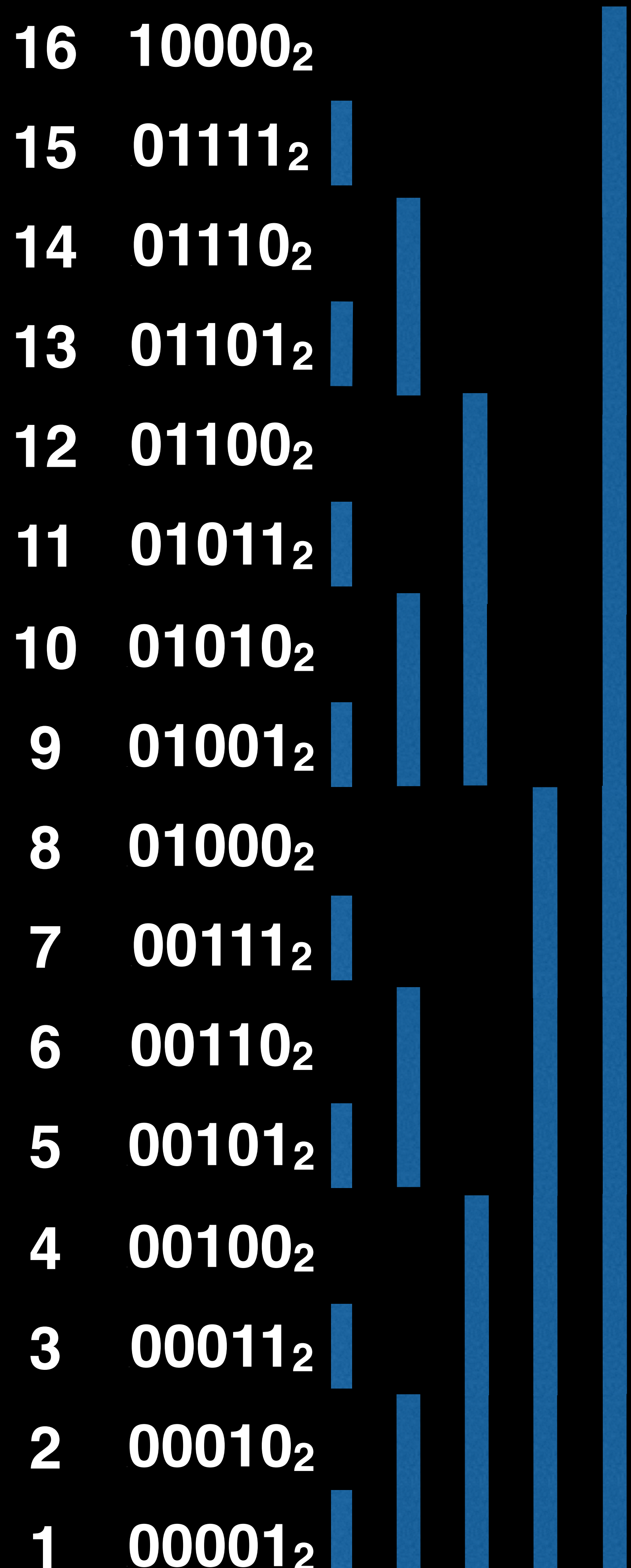
Recall that with range queries we cascaded down from the current index by **continuously removing the LSB.**

$$13 = 1101_2,\ 1101_2 - 0001_2 = 1100_2$$

$$\downarrow$$

$$12 = 1100_2$$

Point Updates



Instead of querying a range to find the interval sum we want to update a cell in our array.

Recall that with range queries we cascaded down from the current index by **continuously removing the LSB.**

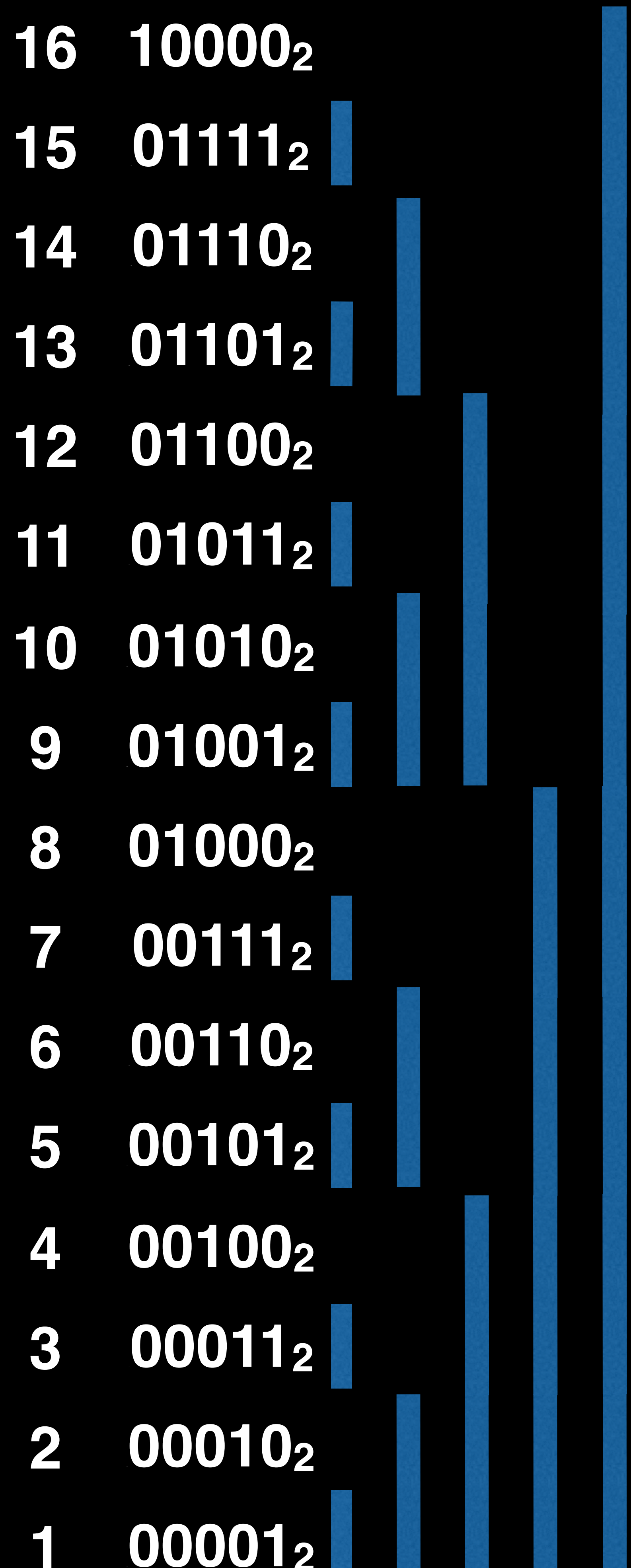
$$13 = 1101_2,\ 1101_2 - 0001_2 = 1100_2$$

$$\downarrow$$

$$12 = 1100_2,\ 1100_2 - 0100_2 = 1000_2$$

$$\downarrow$$

$$8 = 1000_2$$



Point Updates

Instead of querying a range to find the interval sum we want to update a cell in our array.

Recall that with range queries we cascaded down from the current index by **continuously removing the LSB.**

$$13 = 1101_2,\ 1101_2 - 0001_2 = 1100_2$$

$$\downarrow$$

$$12 = 1100_2,\ 1100_2 - 0100_2 = 1000_2$$

$$\downarrow$$

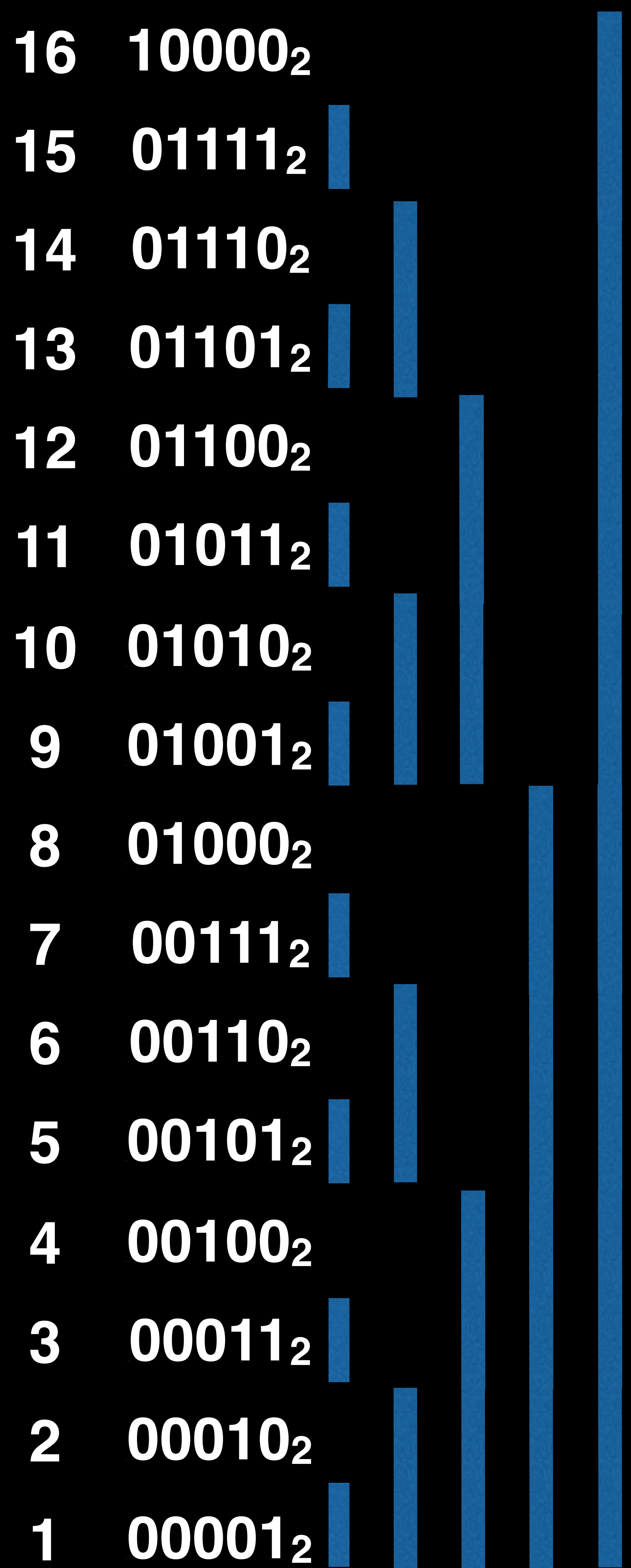
$$8 = 1000_2,\ 1000_2 - 1000_2 = 0000_2$$

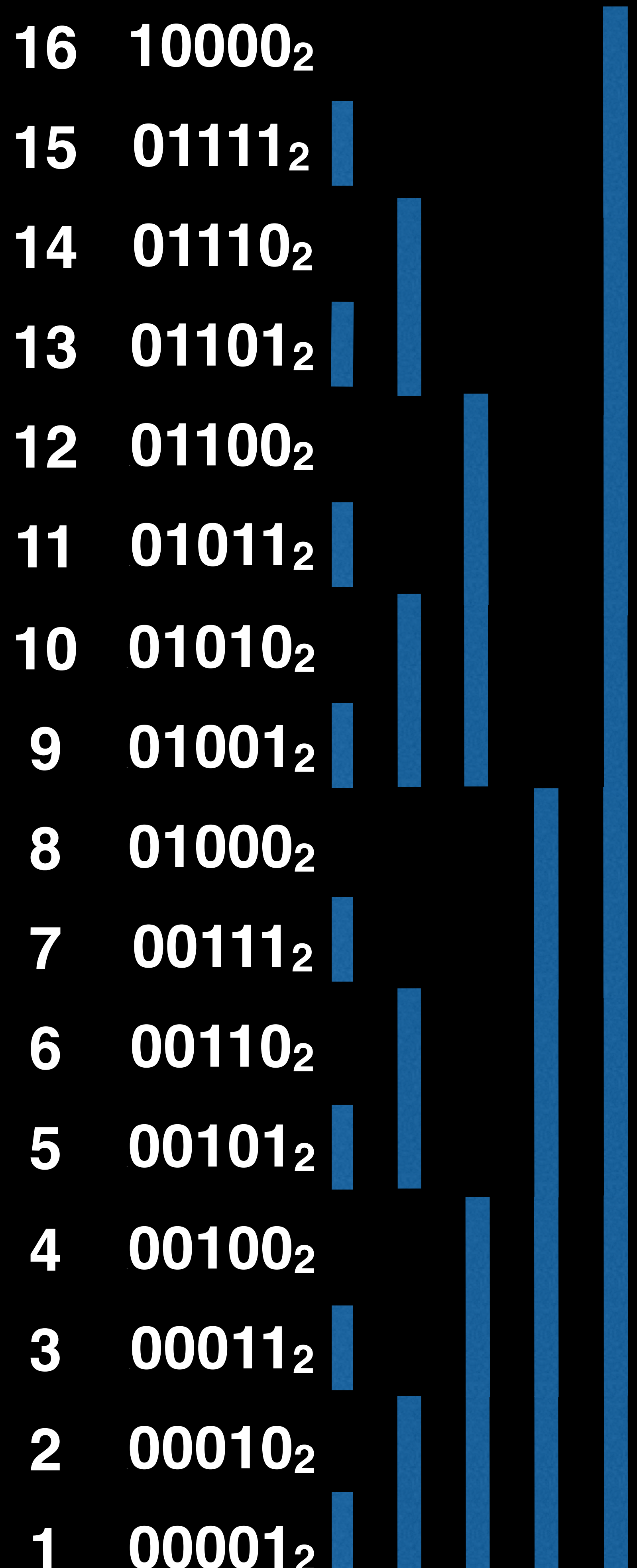
$$\downarrow$$

$$0 = 0000_2$$

Point Updates

Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.





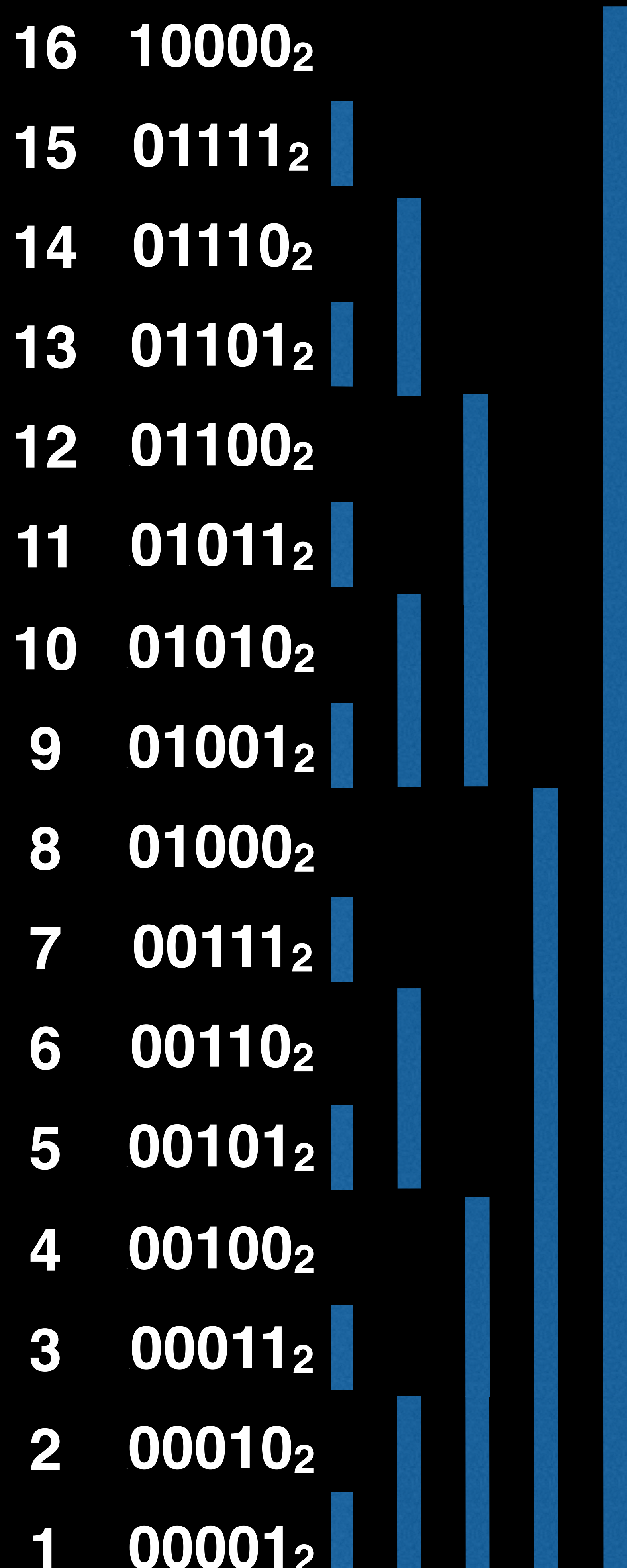
Point Updates

Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.

$9 = 1001_2,\ 1001_2 + 0001_2 = 1010_2$

$\downarrow$

$10 = 1010_2$



Point Updates

Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.

$9 = 1001_2,\ 1001_2 + 0001_2 = 1010_2$

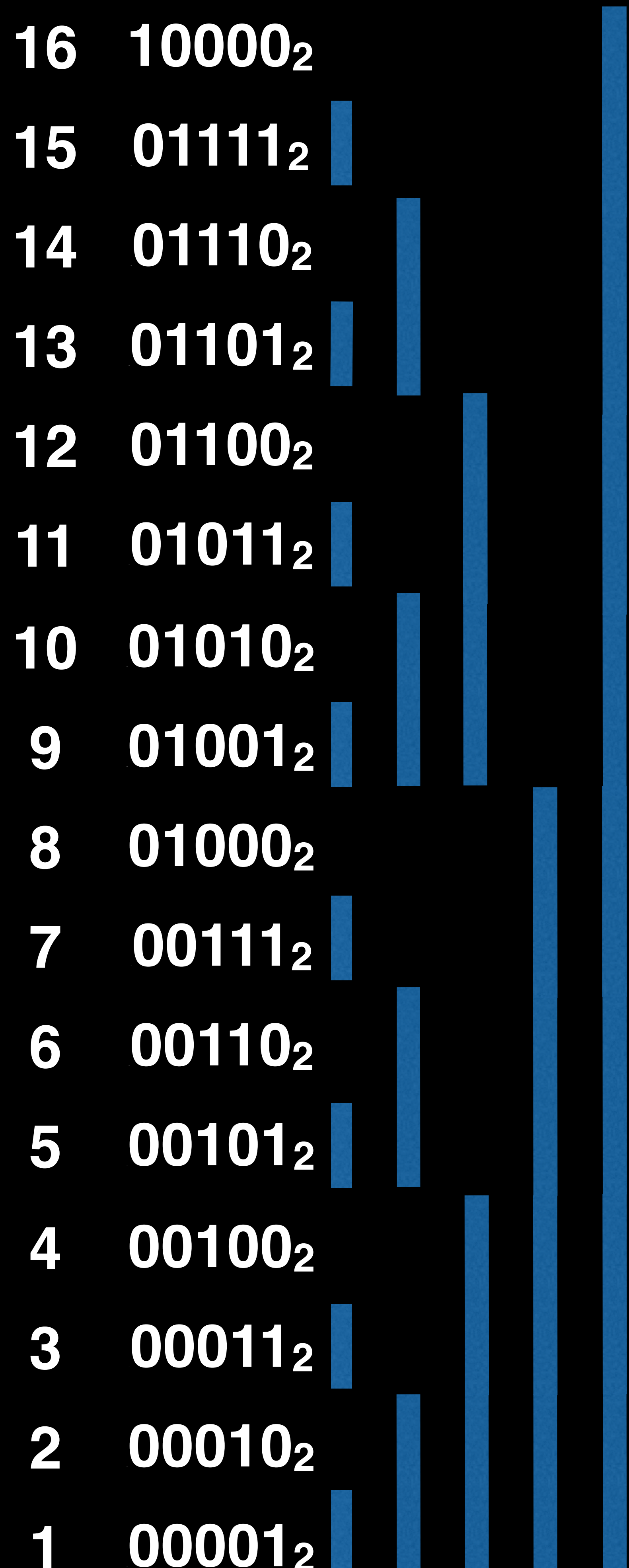
$\downarrow$

$10 = 1010_2,\ 1010_2 + 0010_2 = 1100_2$

$\downarrow$

$12 = 1100_2$

Point Updates



Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.

$9 = 1001_2,\ 1001_2 + 0001_2 = 1010_2$

$\downarrow$

$10 = 1010_2,\ 1010_2 + 0010_2 = 1100_2$

$\downarrow$

$12 = 1100_2,\ 1100_2 + 0100_2 = 10000_2$

$\downarrow$

$16 = 10000_2$

Point Updates

Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.

$$9 = 1001_2,\ 1001_2 + 0001_2 = 1010_2$$

$$\downarrow$$

$$10 = 1010_2,\ 1010_2 + 0010_2 = 1100_2$$

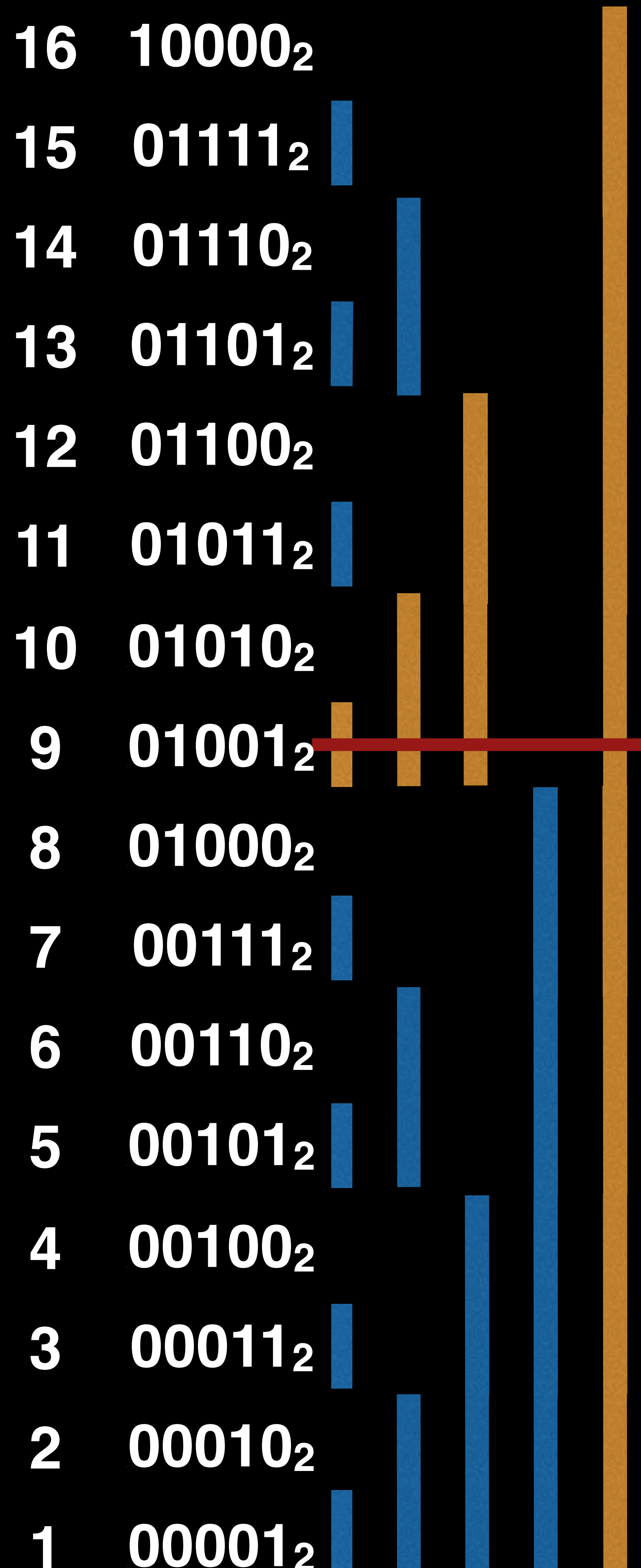
$$\downarrow$$

$$12 = 1100_2,\ 1100_2 + 0100_2 = 10000_2$$

$$\downarrow$$

$$16 = 10000_2$$

Point Updates



Point updates are the opposite of this, we want to **add the LSB** to propagate the value up to the cells responsible for us.

$9 = 1001_2,\ 1001_2+0001_2 = 1010_2$

$\downarrow$

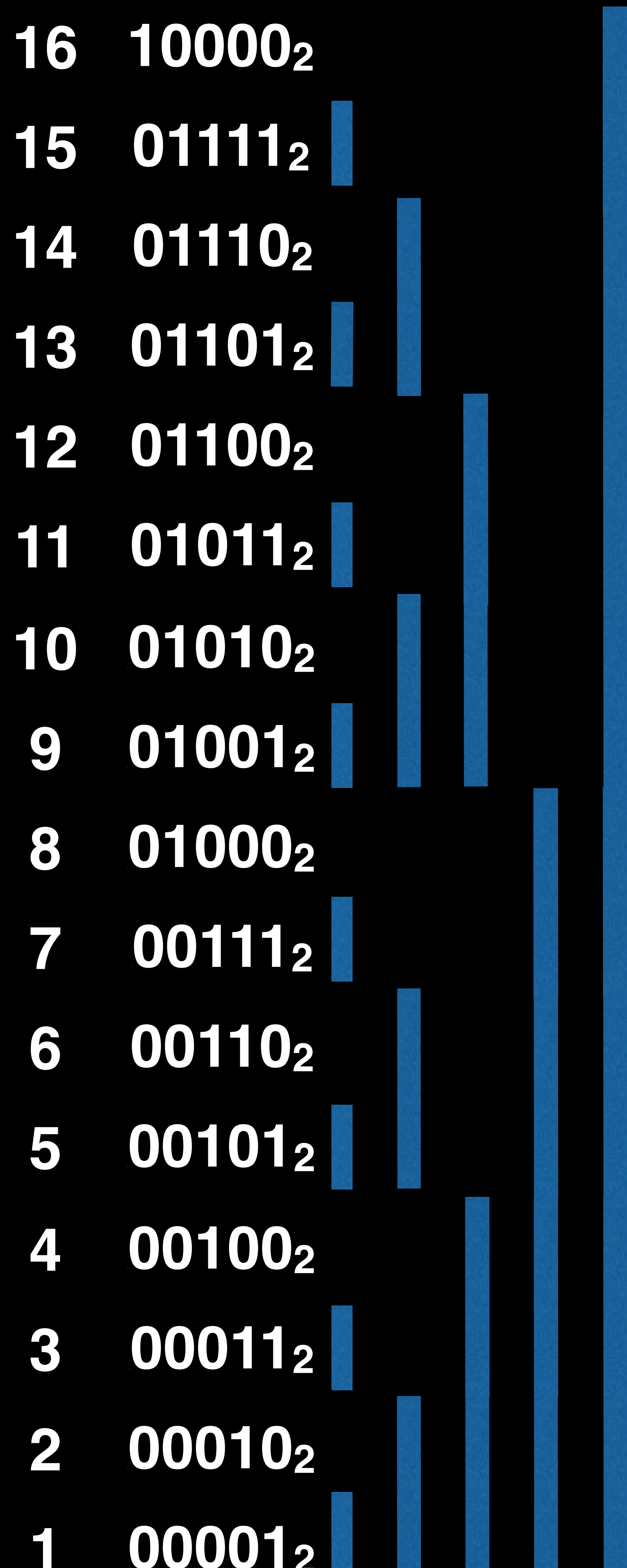
$10 = 1010_2,\ 1010_2+0010_2 = 1100_2$

$\downarrow$

$12 = 1100_2,\ 1100_2+0100_2 = 10000_2$

$\downarrow$

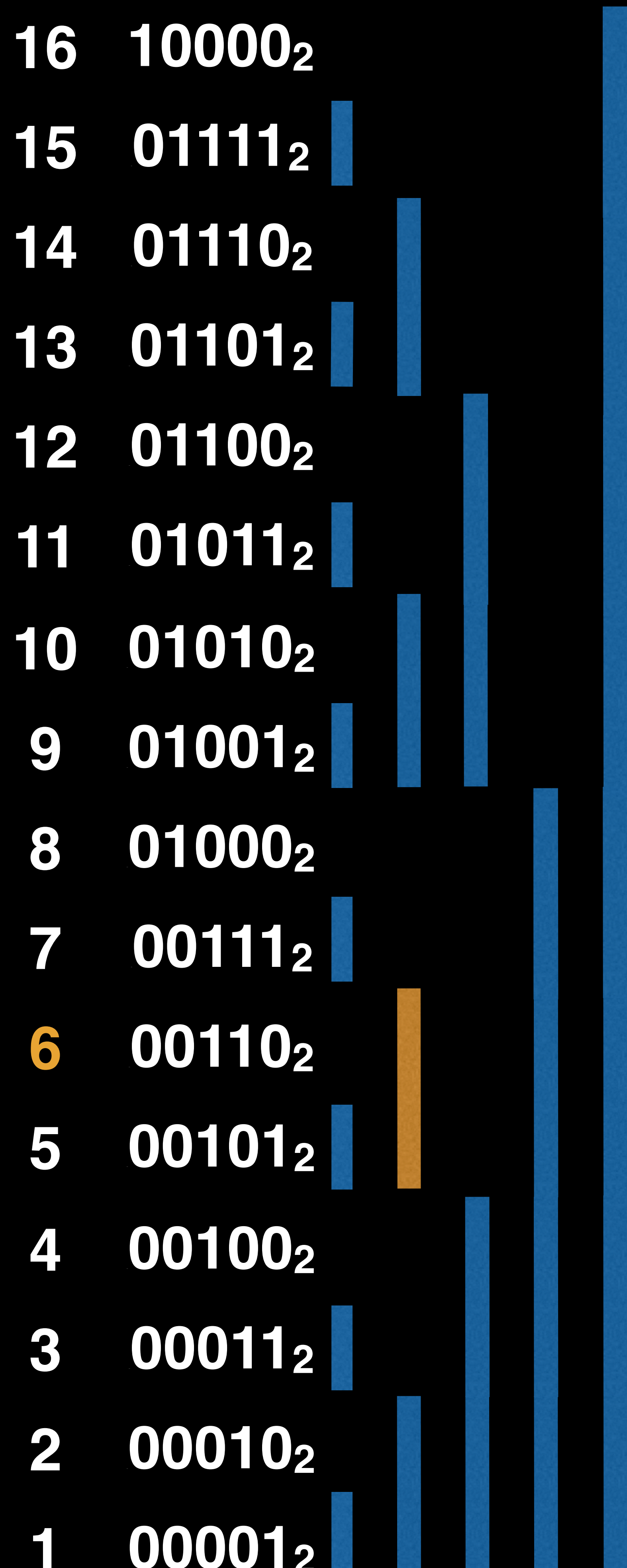
$16 = 10000_2$



Point Updates

If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

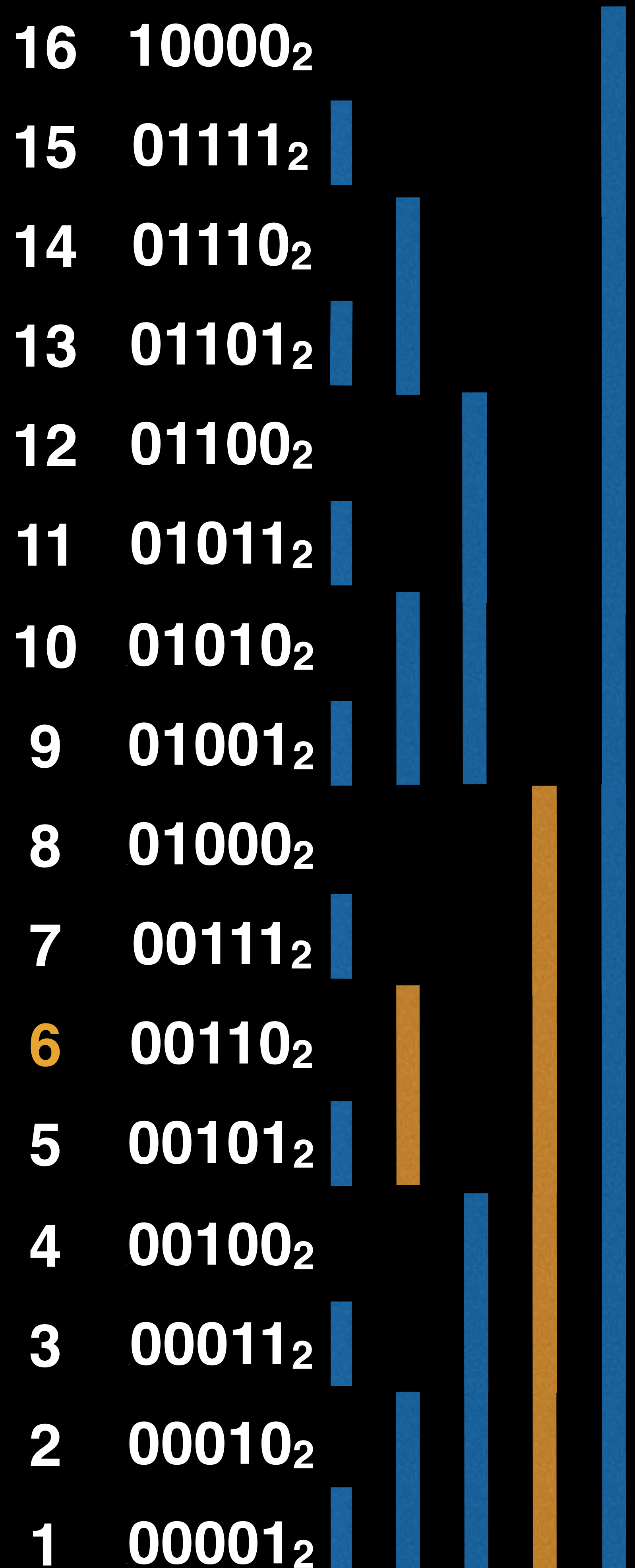
Point Updates



If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

$6 = 0110_2$

Point Updates

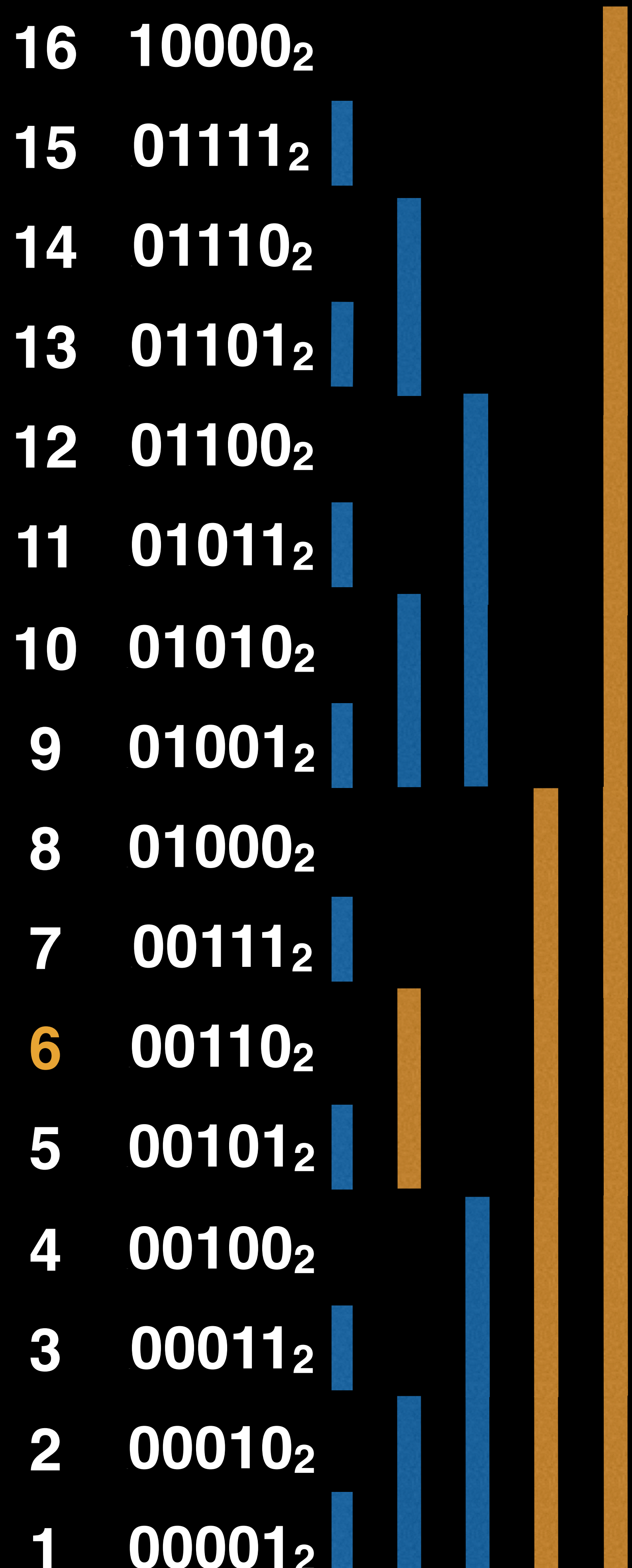


If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

$6 = 0110_2$, $0110_2 + 0010_2 = 1000_2$

$\downarrow$

$8 = 1000_2$



Point Updates

If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

$$6 = 0110_2,\ 0110_2+0010_2 = 1000_2$$

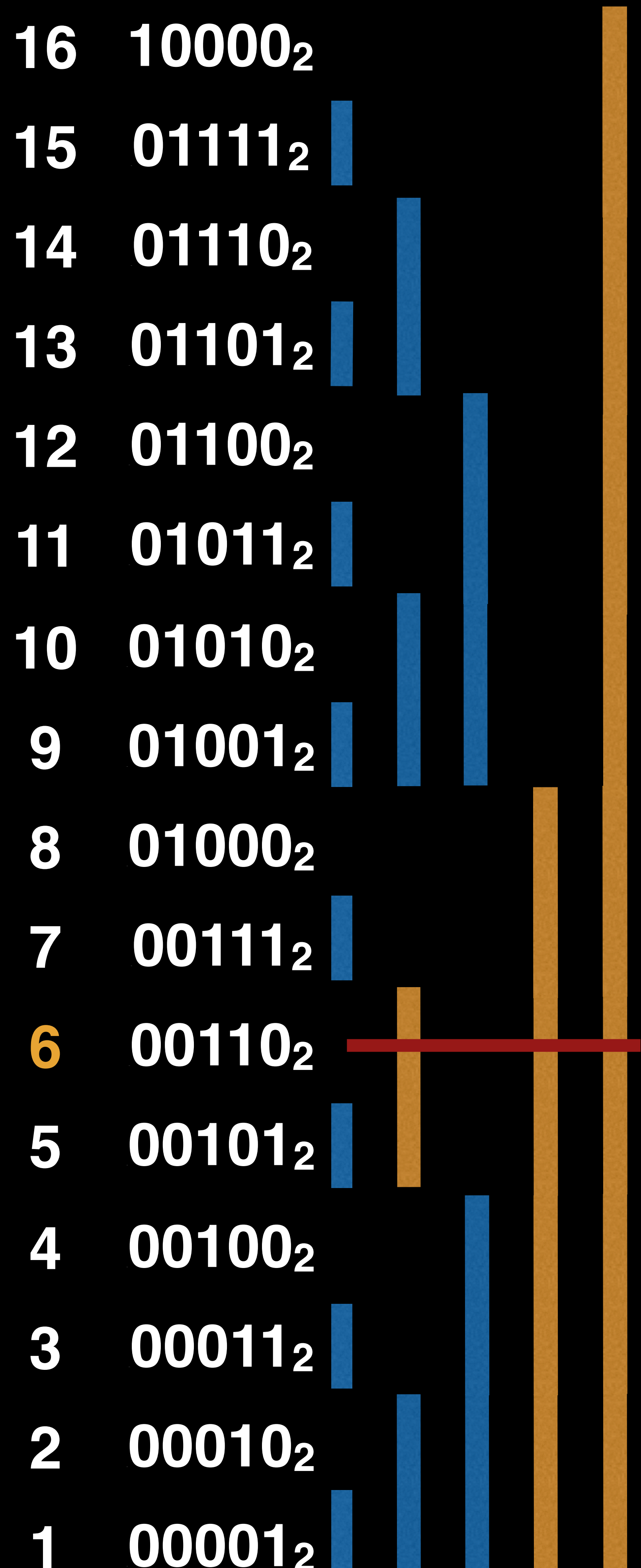
$$\downarrow$$

$$8 = 1000_2,\ 1000_2+1000_2 = 10000_2$$

$$\downarrow$$

$$16 = 10000_2$$

Point Updates



If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

$$6 = 0110_2,\quad 0110_2 + 0010_2 = 1000_2$$

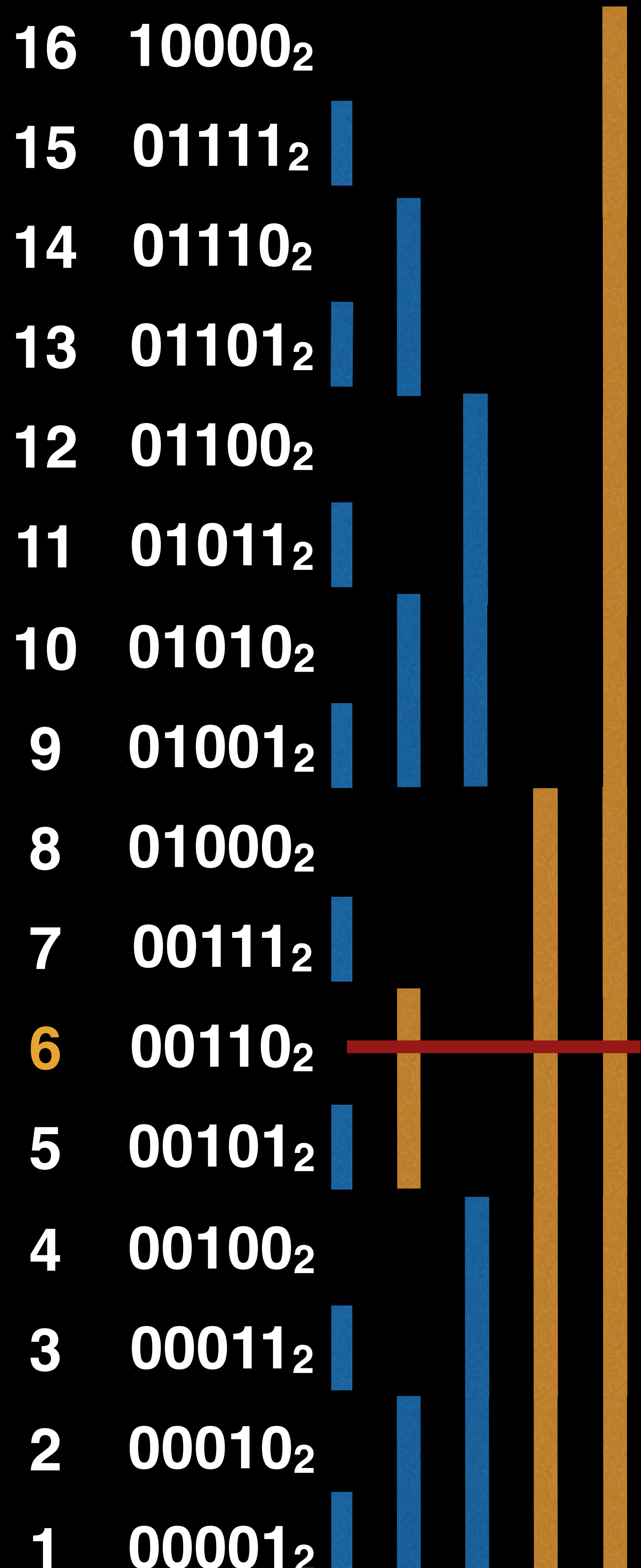
$$\downarrow$$

$$8 = 1000_2,\quad 1000_2 + 1000_2 = 10000_2$$

$$\downarrow$$

$$16 = 10000_2$$

Point Updates



If we add x to position 6 in the Fenwick tree which cells do we also need to modify?

$6 = 0110_2$, $0110_2 + 0010_2 = 1000_2$

$\downarrow$

$8 = 1000_2$, $1000_2 + 1000_2 = 10000_2$

$\downarrow$

$16 = 10000_2$

<u>Required Updates:</u>

A[6] = A[6] + x
A[8] = A[8] + x
A[16] = A[16] + x

Point update algorithm

To update the cell at index i in the a Fenwick tree of size N:

```
function add(i, x):
    while i < N:
        tree[i] = tree[i] + x
        i = i + LSB(i)
```

Where **LSB** returns the value of the least significant bit. For example:

LSB(12) = 4 because $12_{10} = 1100_2$ and the least significant bit of 1100_2 is 100_2, or 4 in base ten

Fenwick Tree construction follows in next video

Implementation source code and tests can all be found at the following link:

github.com/williamfiset/data-structures

Fenwick Tree Construction

William Fiset

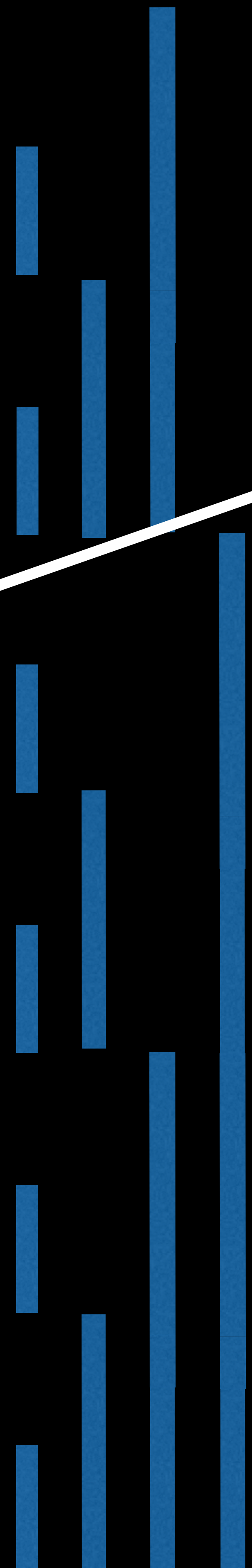
Naive Construction

Let A be an array of values. For each element in A at index *i* do a point update on the Fenwick tree with a value of A[i]. There are *n* elements and each point update takes O(log(n)) for a total of O(nlog(n)), can we do better?

Linear Construction

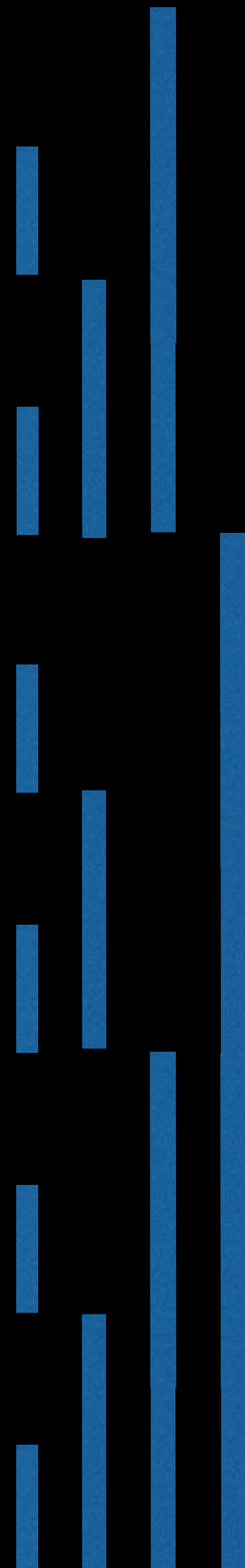
Input values we wish to turn into a legitimate Fenwick tree.

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	4
1	0001_2	3



Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	4
1	0001_2	3

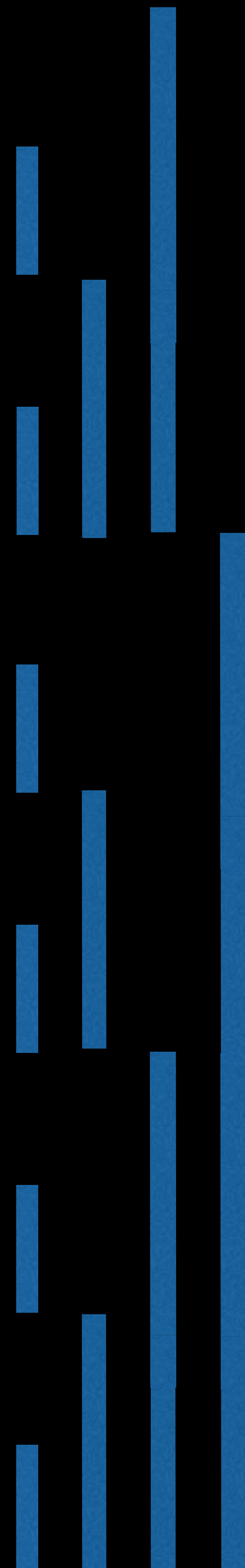


Idea: Add the value in the current cell to the **immediate cell** that is responsible for us. This resembles what we did for point updates but only one cell at a time.

This will make the ‘cascading’ effect in range queries possible by propagating the value in each cell throughout the tree.

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	4
1	0001_2	3



Let i be the current index

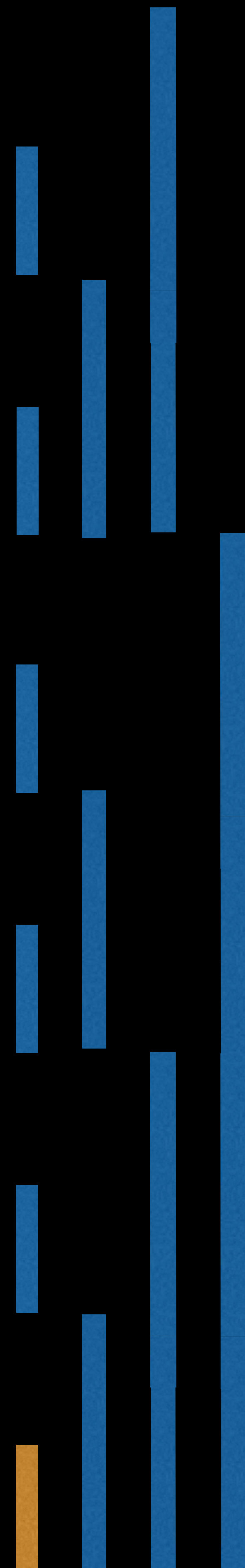
The immediate cell above us is at position j given by:

$$j := i + \mathrm{LSB}(i)$$

Where LSB is the **Least Significant Bit** of i

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	4
1	0001_2	3

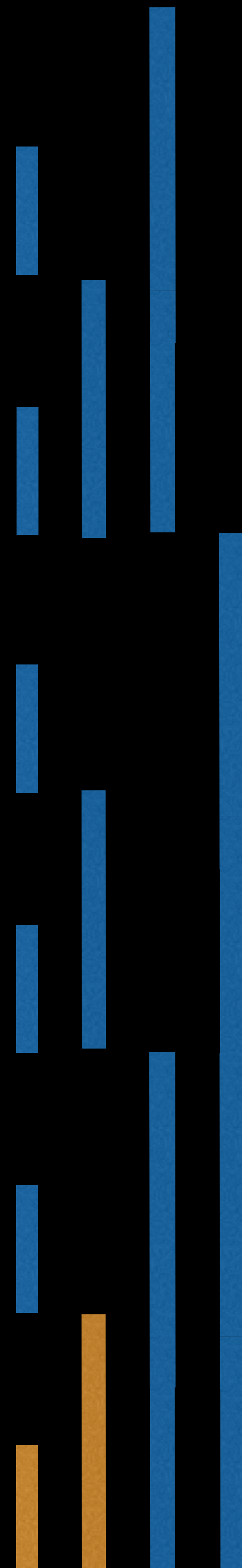


$i = 1 = 0001_2$

$j = 0001_2 + 0001_2 = 0010_2$
$= 2$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	4+3
1	0001_2	3

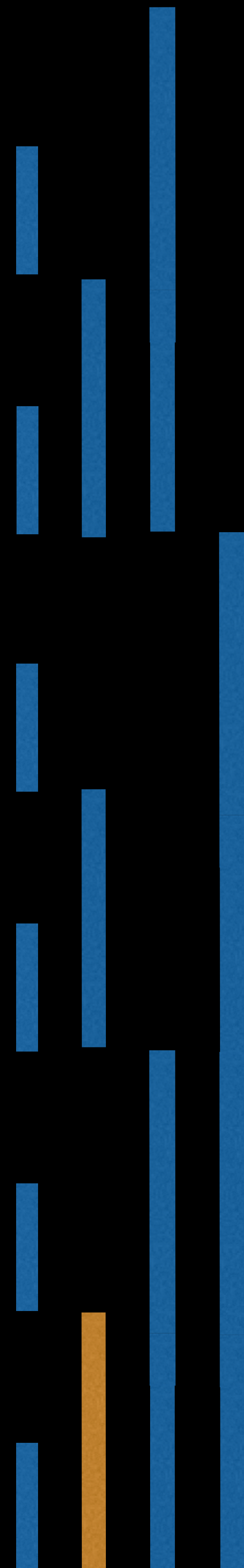


$i = 1 = 0001_2$

$j = 0001_2 + 0001_2 = 0010_2$
$= 2$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7
3	0011_2	-2
2	0010_2	7
1	0001_2	3

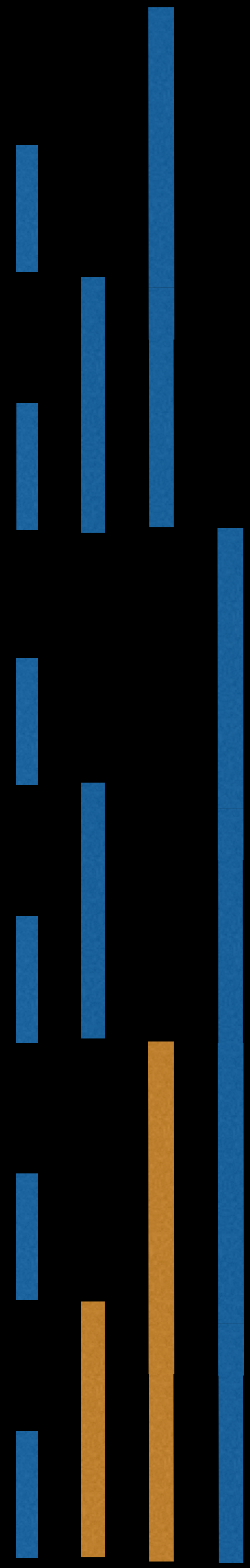


$i = 2 = 0010_2$

$j = 0010_2 + 0010_2 = 0100_2$
$= 4$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	7+7
3	0011_2	-2
2	0010_2	7
1	0001_2	3

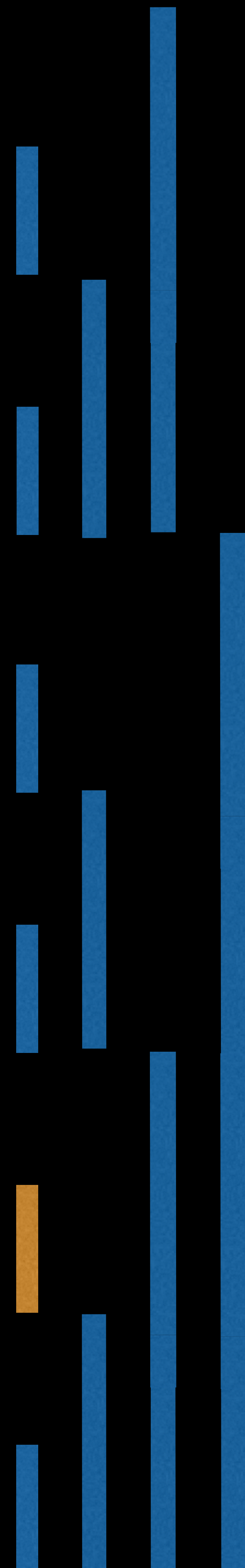


$i = 2 = 0010_2$

$j = 0010_2 + 0010_2 = 0100_2$
$= 4$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	14
3	0011_2	-2
2	0010_2	7
1	0001_2	3

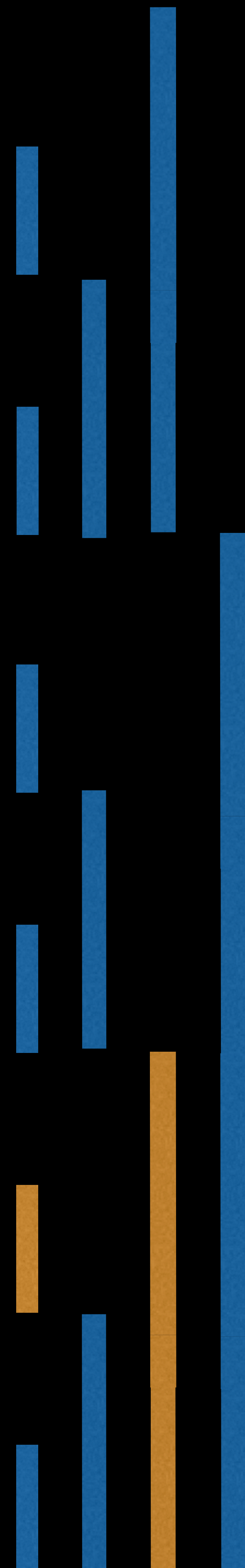


$$i = 3 = 0011_2$$

$$j = 0011_2 + 0001_2 = 0100_2 = 4$$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	14+-2
3	0011_2	-2
2	0010_2	7
1	0001_2	3

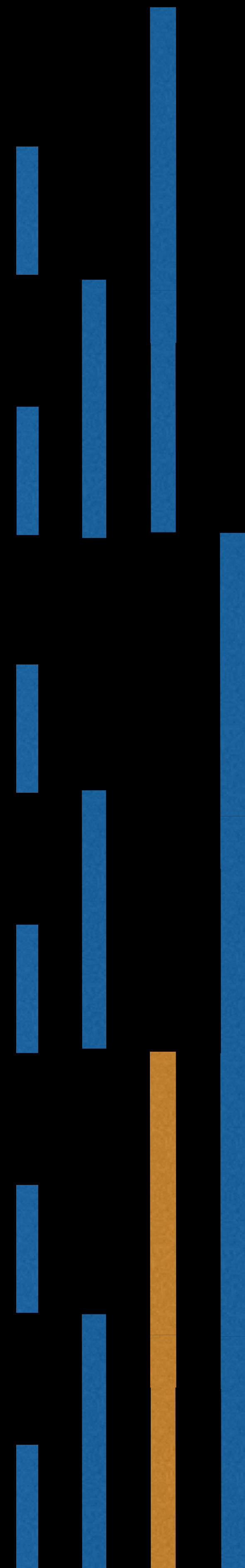


$$i = 3 = 0011_2$$

$$j = 0011_2 + 0001_2 = 0100_2$$
$$= 4$$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

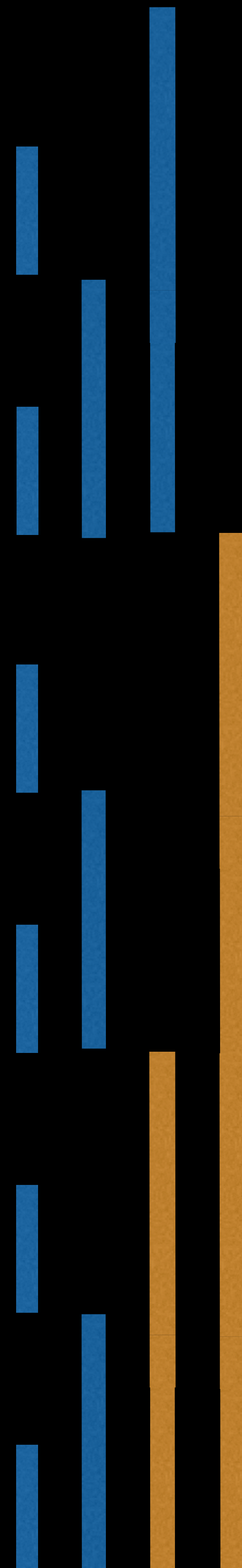


$i = 4 = 0100_2$

$j = 0100_2 + 0100_2 = 1000_2$
$= 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	-8+12
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

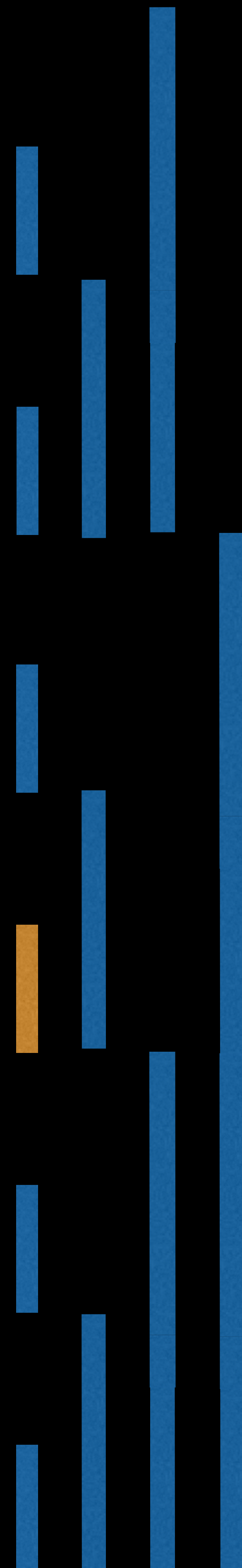


$i = 4 = 0100_2$

$j = 0100_2 + 0100_2 = 1000_2$
$= 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	4
7	0111_2	5
6	0110_2	11
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

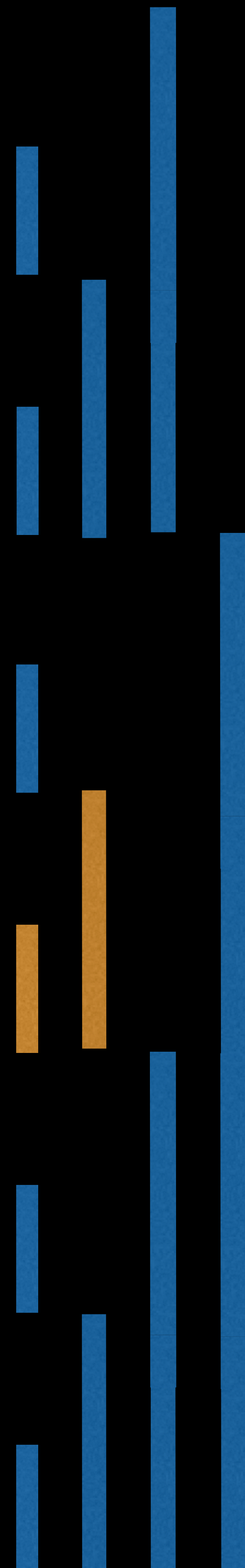


$i = 5 = 0101_2$

$j = 0101_2 + 0001_2 = 0110_2$
$= 6$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	4
7	0111_2	5
6	0110_2	11+3
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

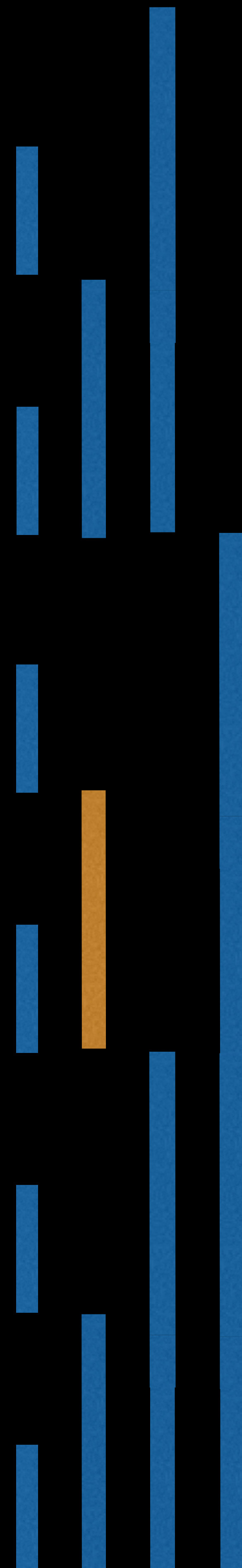


$$i = 5 = 0101_2$$

$$j = 0101_2 + 0001_2 = 0110_2$$
$$= 6$$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	4
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

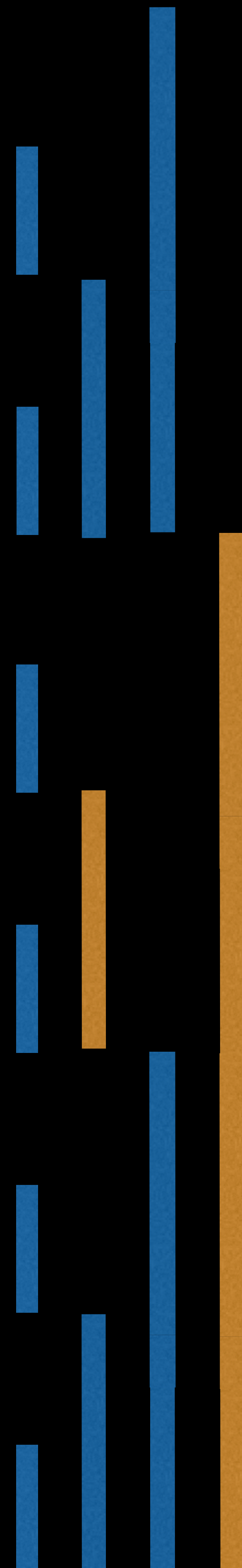


$i = 6 = 0110_2$

$j = 0110_2 + 0010_2 = 1000_2$
$= 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	4+14
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

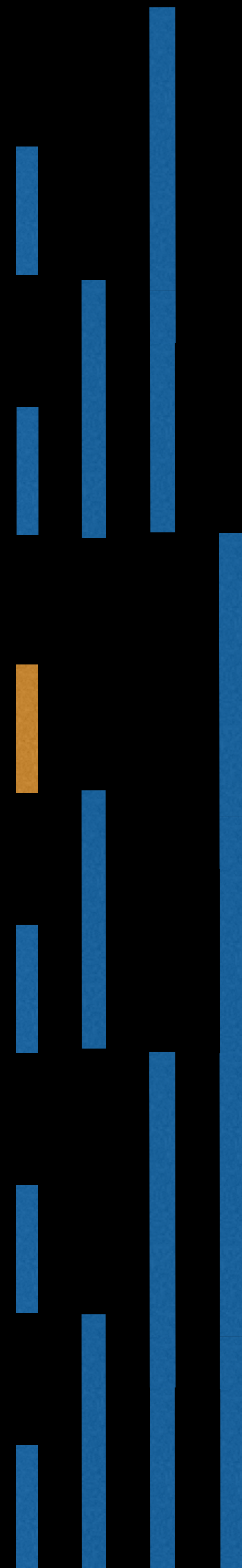


$$i = 6 = 0110_2$$

$$j = 0110_2 + 0010_2 = 1000_2 = 8$$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	18
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

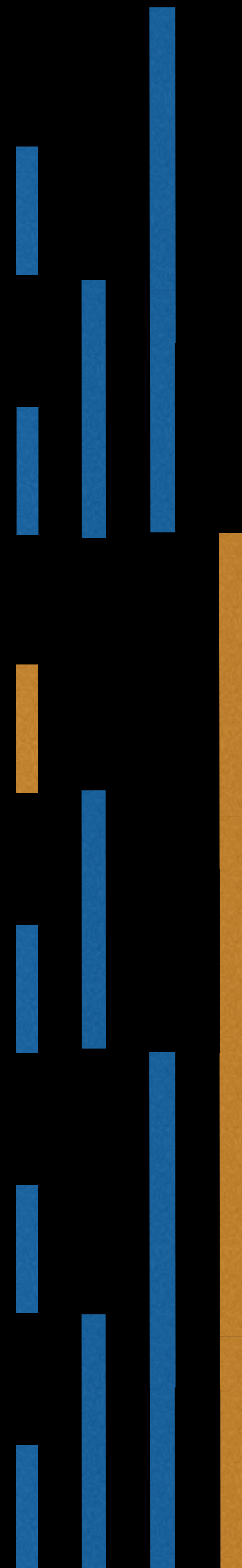


$i = 7 = 0111_2$

$j = 0111_2 + 0001_2 = 1000_2$
$= 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	18+5
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

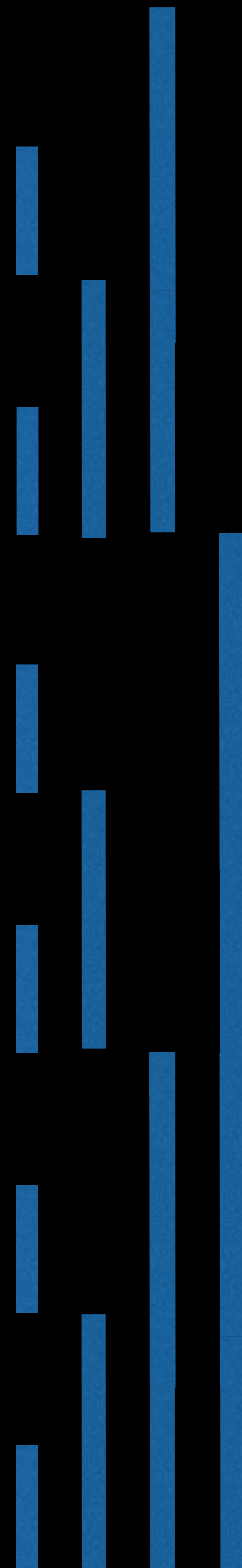


$i = 7 = 0111_2$

$j = 0111_2 + 0001_2 = 1000_2 = 8$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3



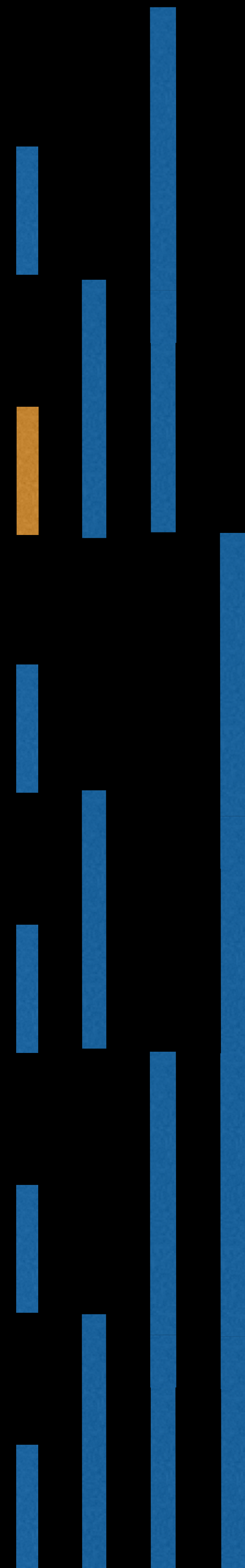
$i = 8 = 1000_2$

$j = 1000_2 + 1000_2 = 10000_2$
$= \mathbf{16}$

Ignore updating j if
index is out of bounds

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

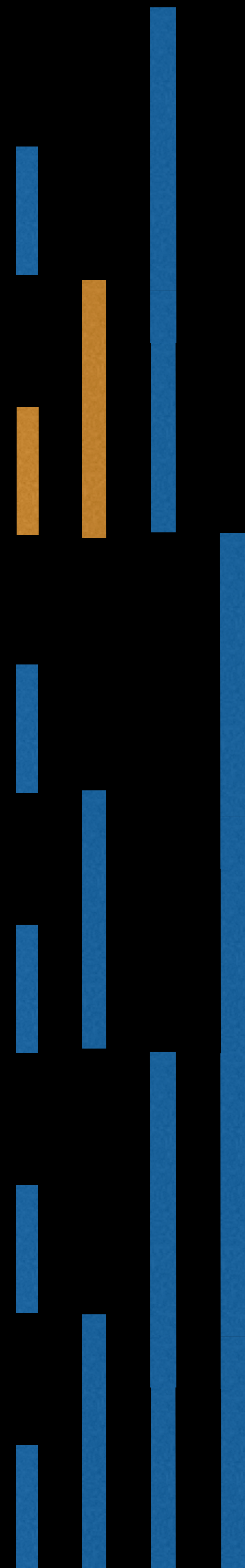


$$i = 9 = 1001_2$$

$$j = 1001_2 + 0001_2 = 1010_2 = 10$$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	2+-9
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

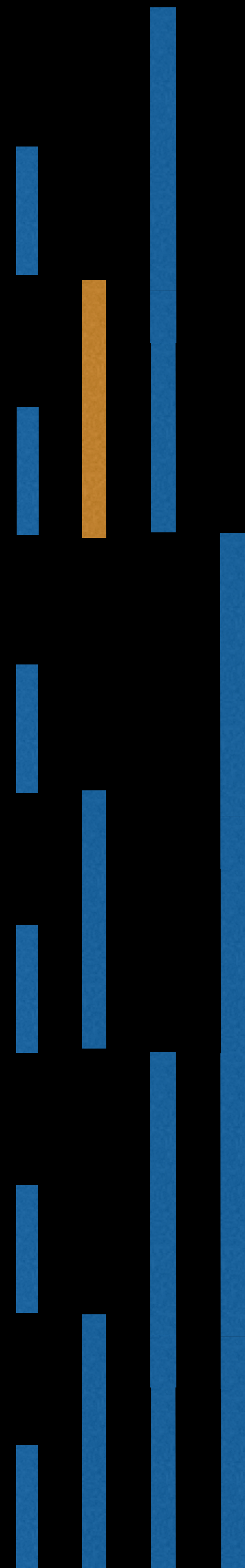


$i = 9 = 1001_2$

$j = 1001_2 + 0001_2 = 1010_2$
$= 10$

Linear Construction

12	1100_2	-8
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

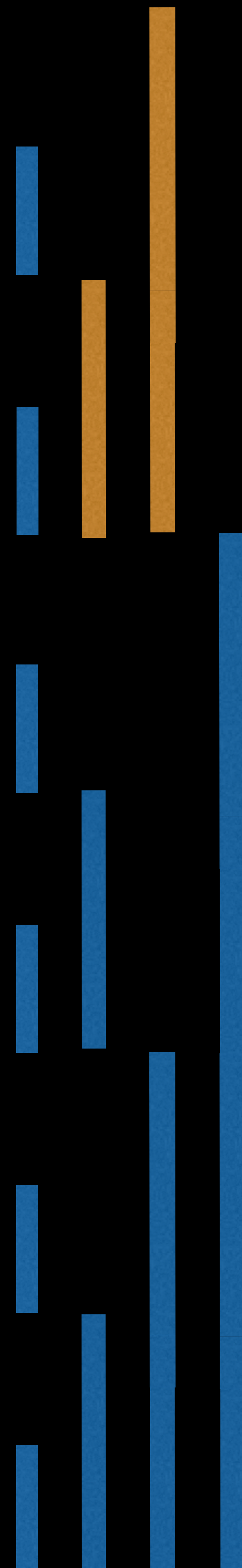


$i = 10 = 1010_2$

$j = 1010_2 + 0010_2 = 1100_2$
$= 12$

Linear Construction

12	1100_2	-8+-7
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

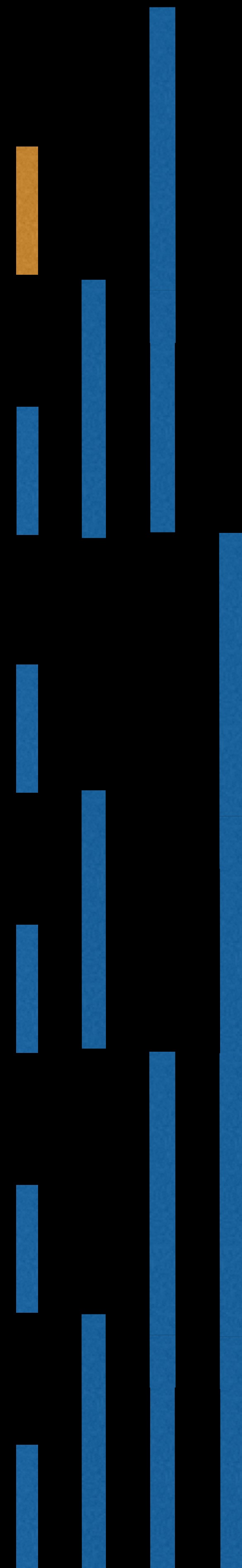


$i = 10 = 1010_2$

$j = 1010_2 + 0010_2 = 1100_2$
$= 12$

Linear Construction

12	1100_2	-15
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

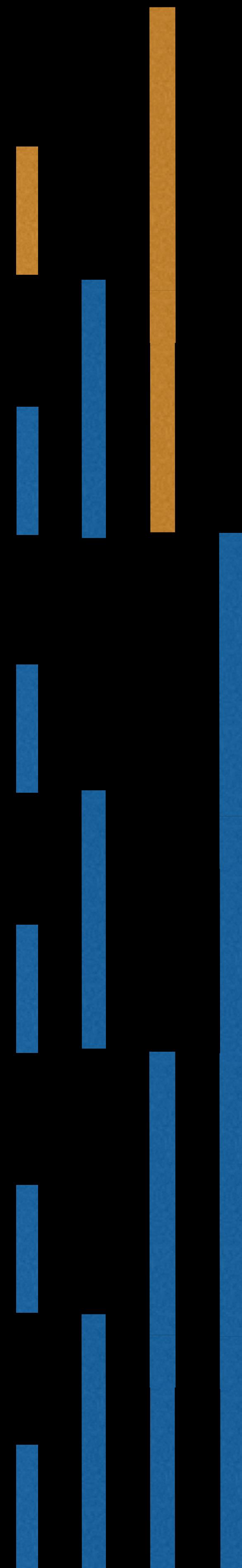


$i = 11 = 1011_2$

$j = 1011_2 + 0001_2 = 1100_2$
$= 12$

Linear Construction

12	1100_2	-15+4
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3

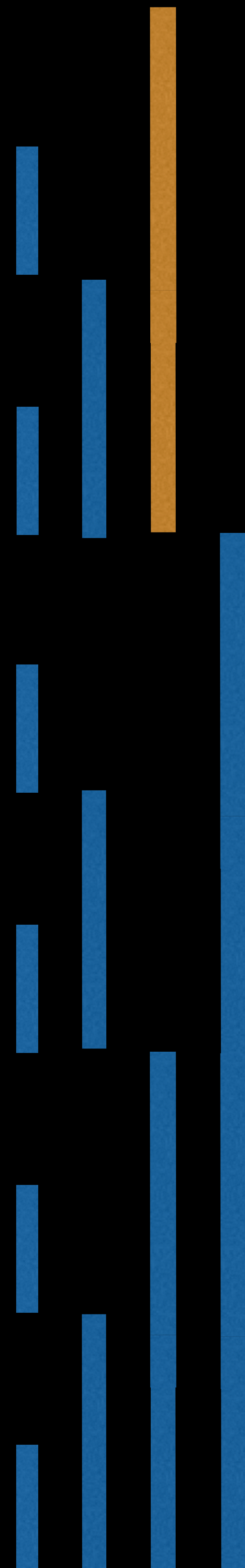


$i = 11 = 1011_2$

$j = 1011_2 + 0001_2 = 1100_2$
$= 12$

Linear Construction

12	1100_2	-11
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3



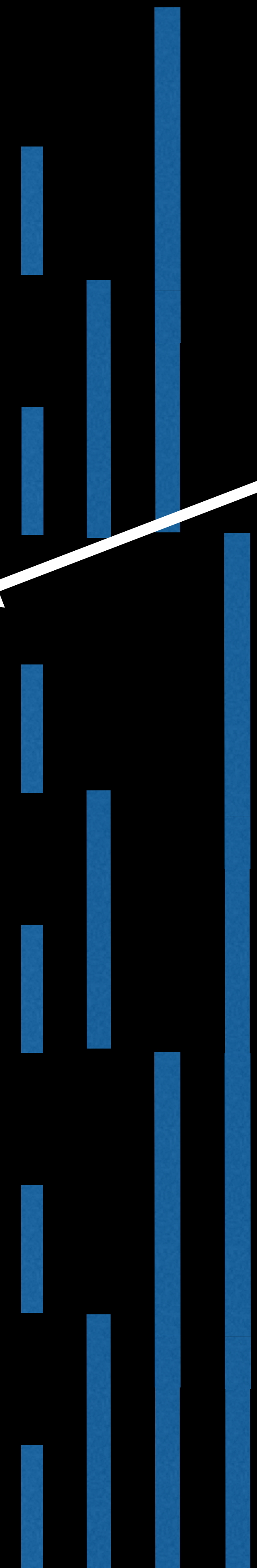
$i = 12 = 1100_2$

$j = 1100_2 + 0100_2 = 10000_2$
$= \mathbf{16}$

Ignore updating j if
index is out of bounds

Linear Construction

12	1100_2	-11
11	1011_2	4
10	1010_2	-7
9	1001_2	-9
8	1000_2	23
7	0111_2	5
6	0110_2	14
5	0101_2	3
4	0100_2	12
3	0011_2	-2
2	0010_2	7
1	0001_2	3



Constructed Fenwick tree! We can now perform point and range query updates as required.

Construction Algorithm

```
# Make sure values is 1-based!
function construct(values):

    N := length(values)

    # Clone the values array since we’re
    # doing in place operations
    tree = deepCopy(values)

    for i = 1,2,3, … N:
        j := i + LSB(i)
        if j < N:
            tree[j] = tree[j] + tree[i]

    return tree
```

Fenwick Tree source code follows in next video!

Implementation source code and tests can all be found at the following link:

github.com/williamfiset/data-structures

Fenwick Tree Source Code

William Fiset

Source Code Link

Implementation source code
and tests can all be found
at the following link:

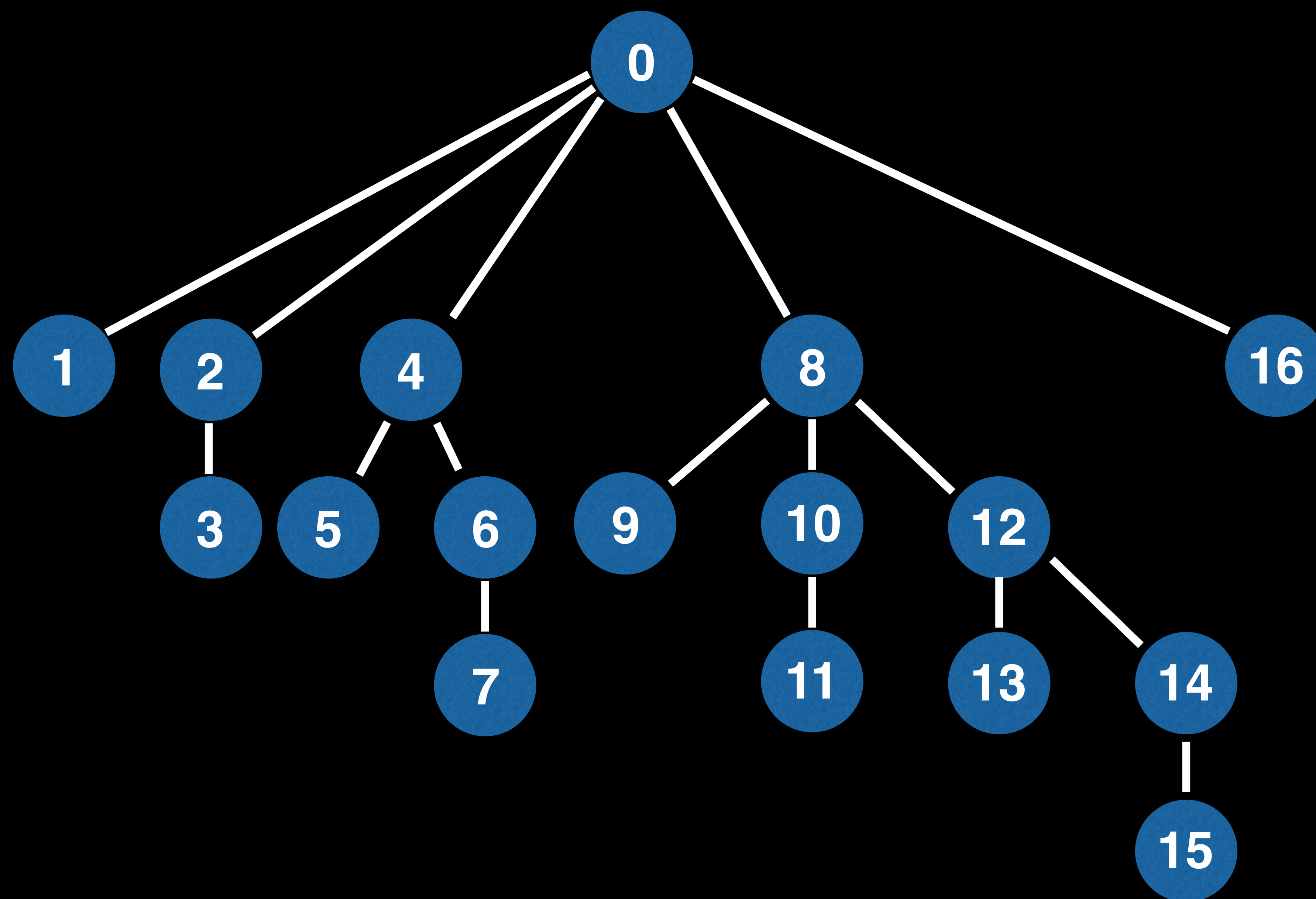
github.com/williamfiset/data-structures

NOTE: Make sure you have understood the previous video sections explaining how a Fenwick Tree works before continuing!

Fenwick Tree Range Update Visualization

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

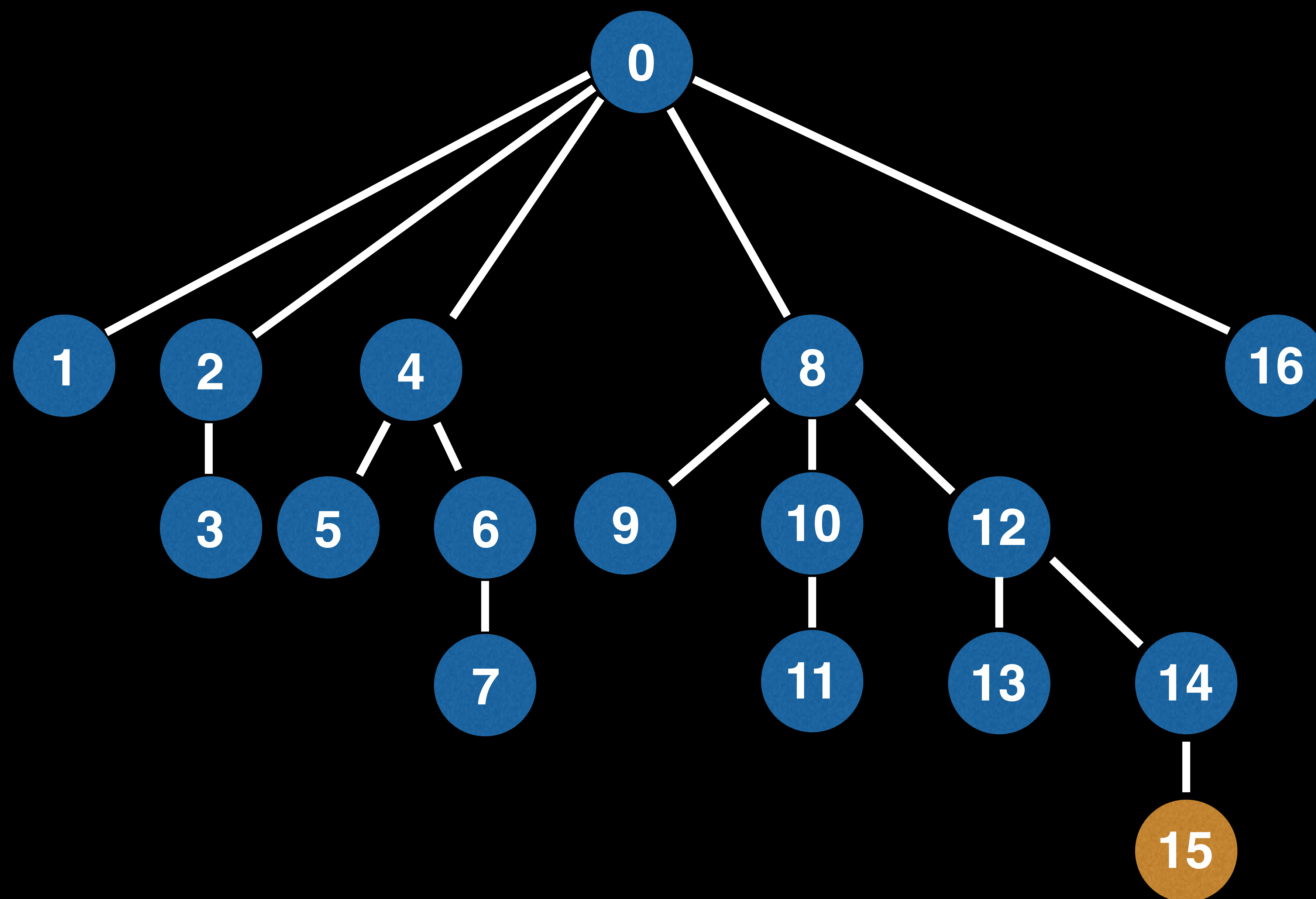
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7



Find the sum in the array between [11,15]

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

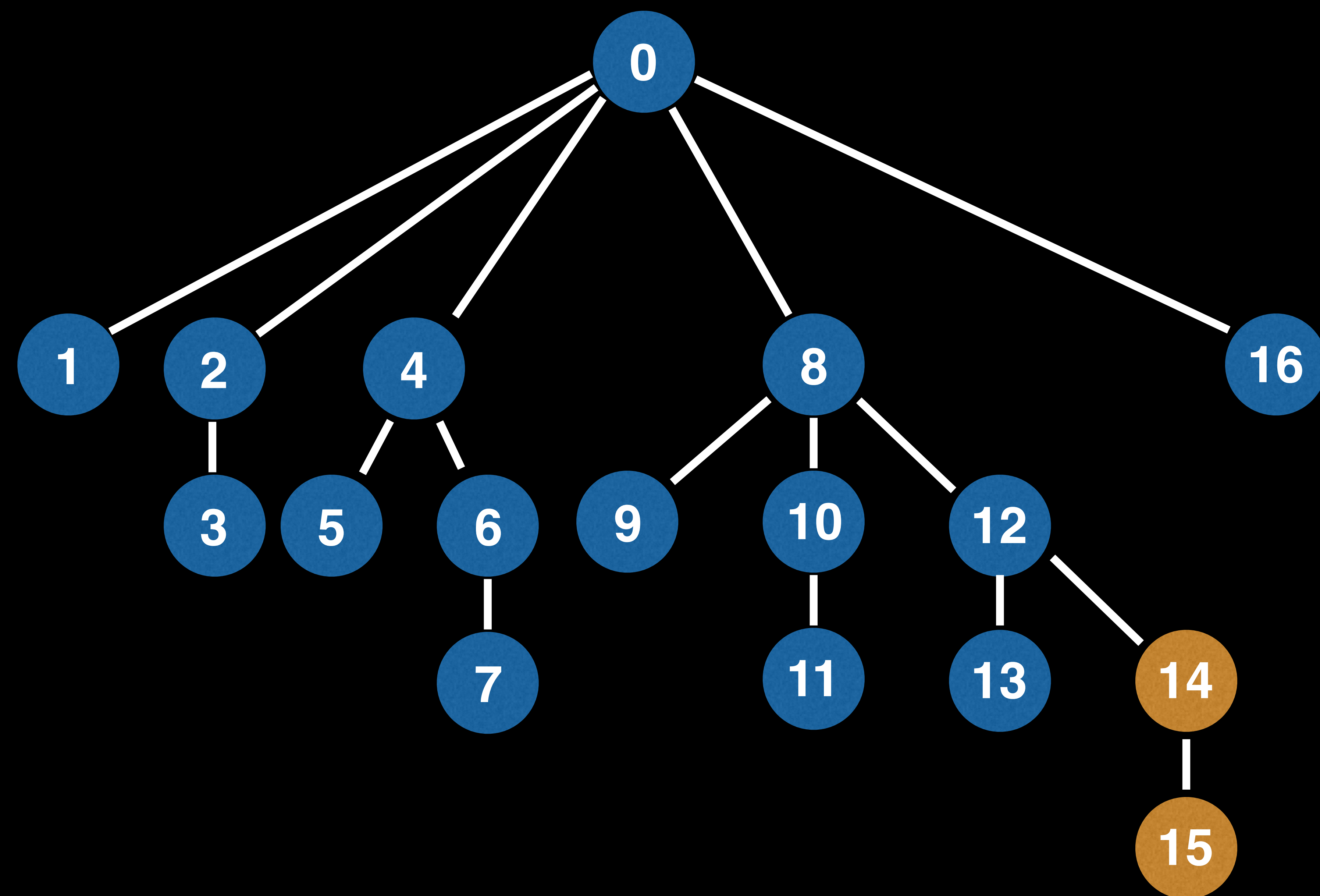


Find the sum in the array between [11,15]

6

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

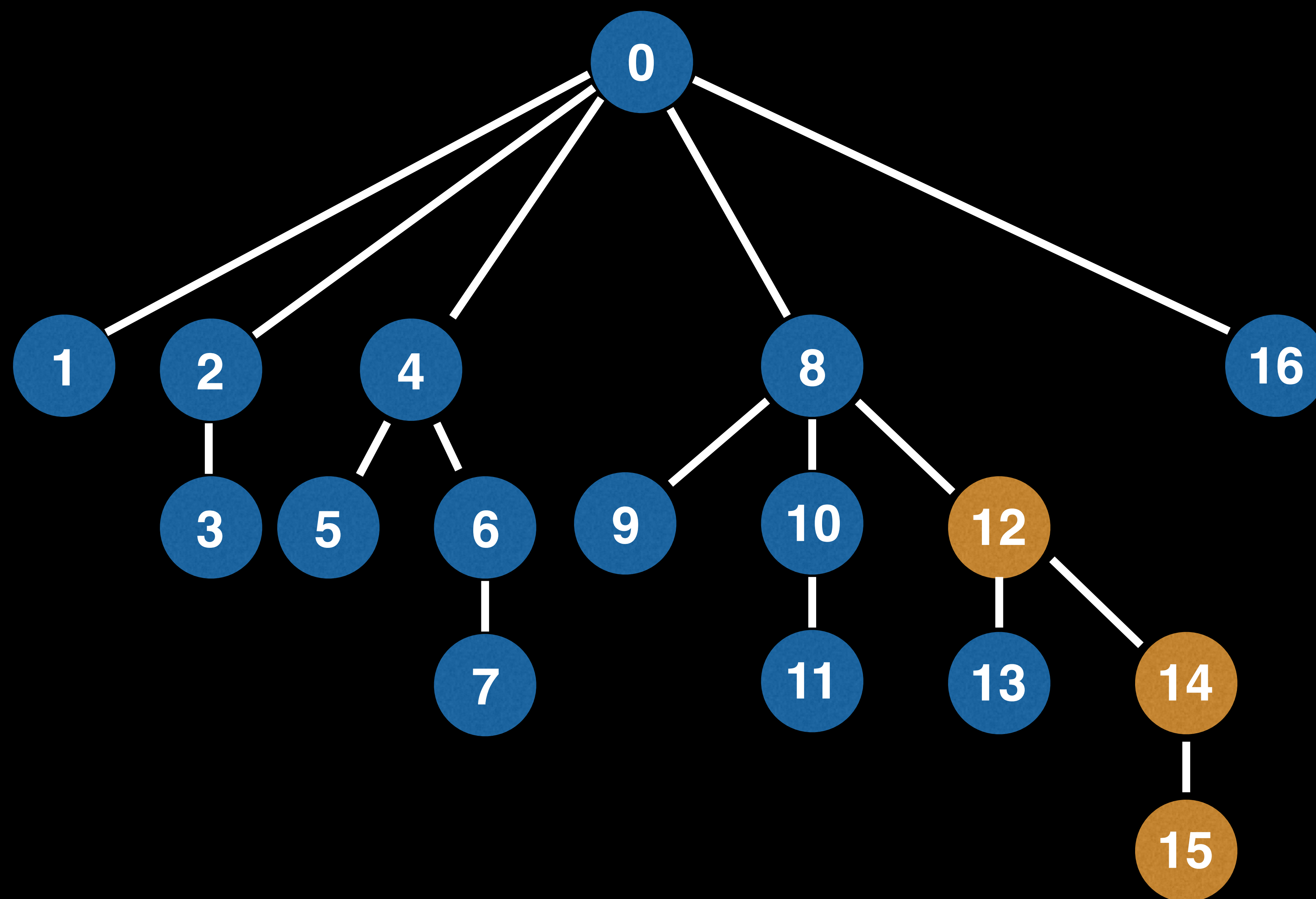


Find the sum in the array between [11,15]

$6 + -1$

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

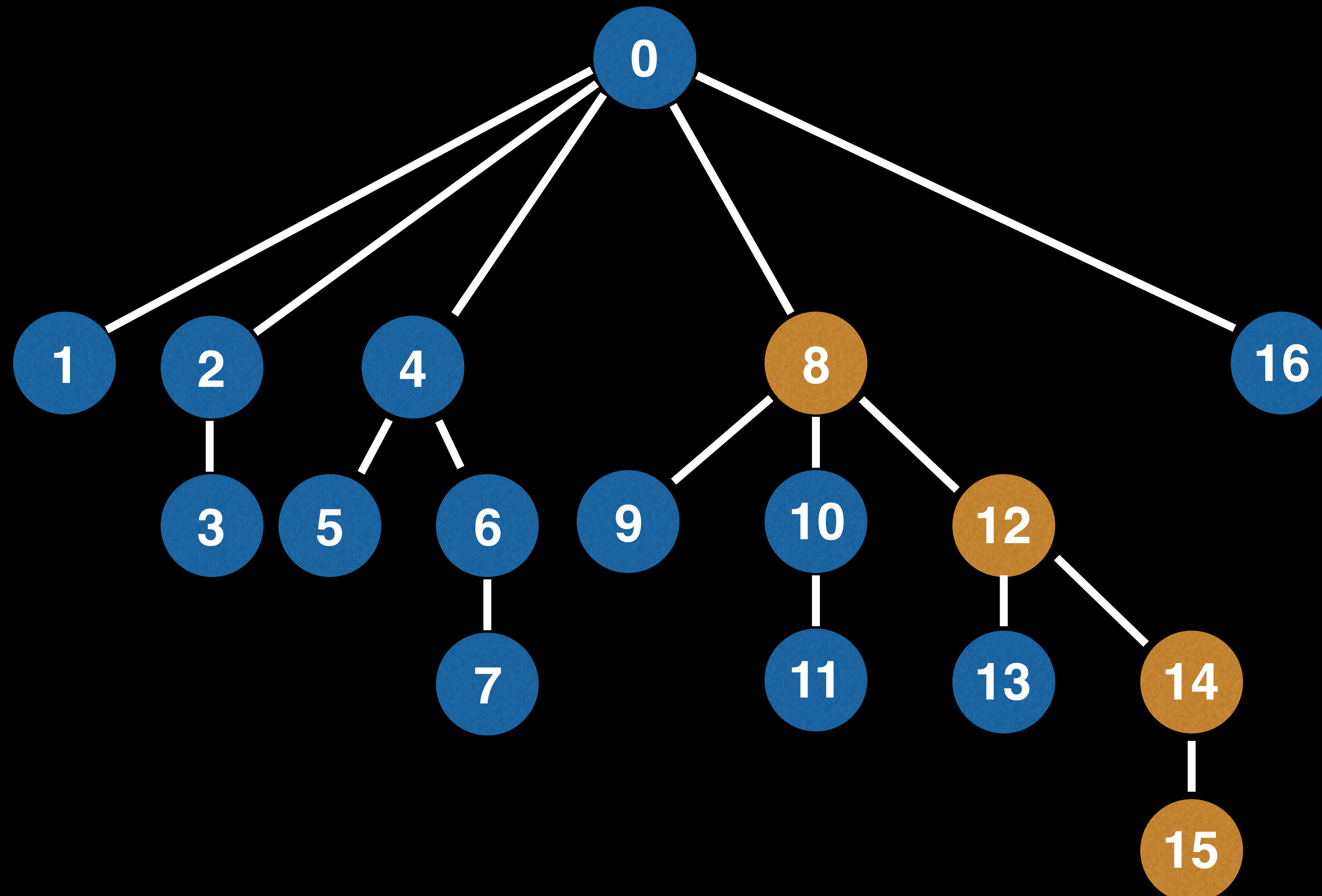


Find the sum in the array between [11,15]

$6 + -1 + -10$

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

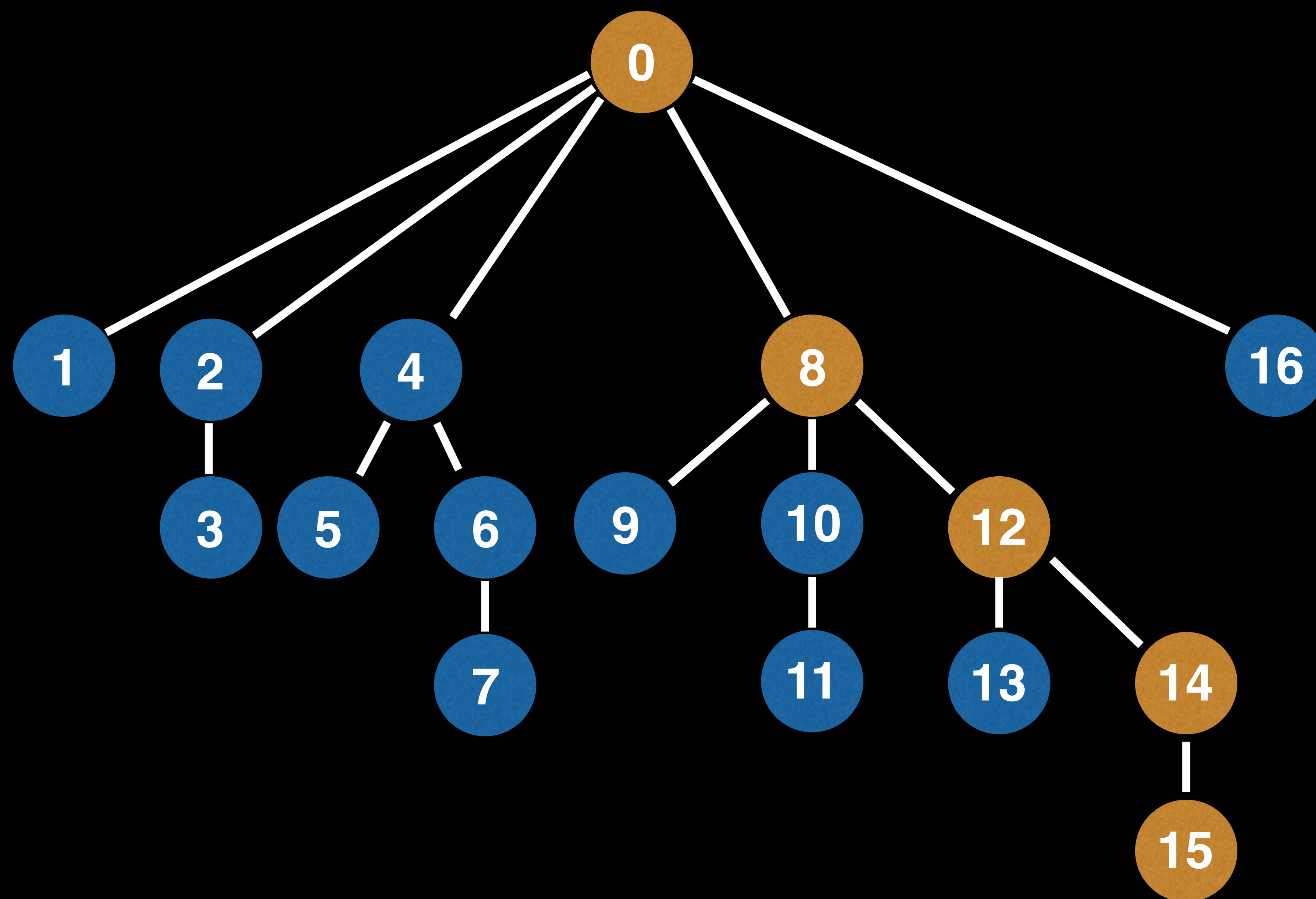


Find the sum in the array between [11,15]

$$6 + -1 + -10 + 26$$

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

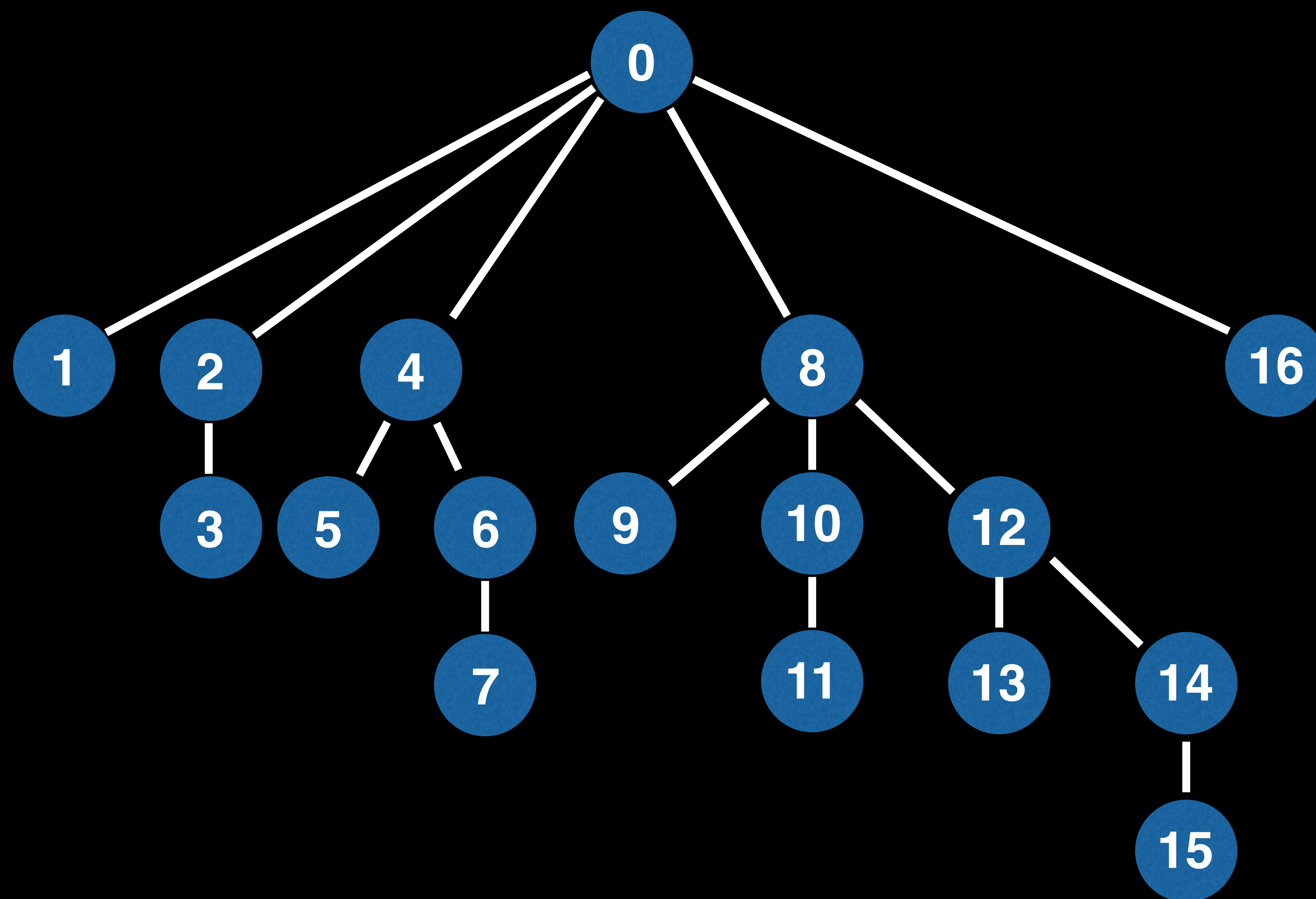


Find the sum in the array between [11,15]

$$6 + -1 + -10 + 26$$

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

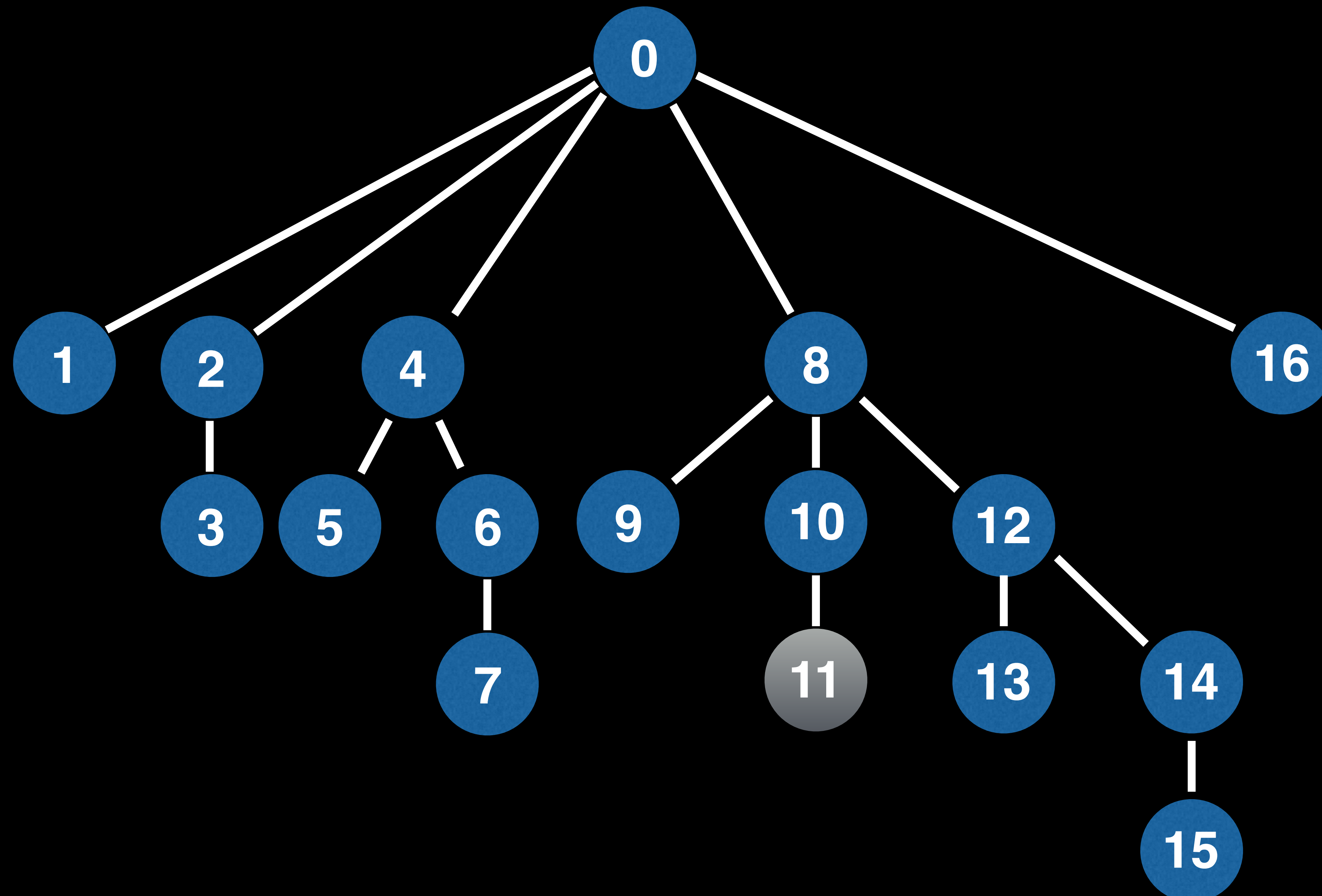


Find the sum in the array between [11,15]

(6 + −1 + −10 + 26) − (

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

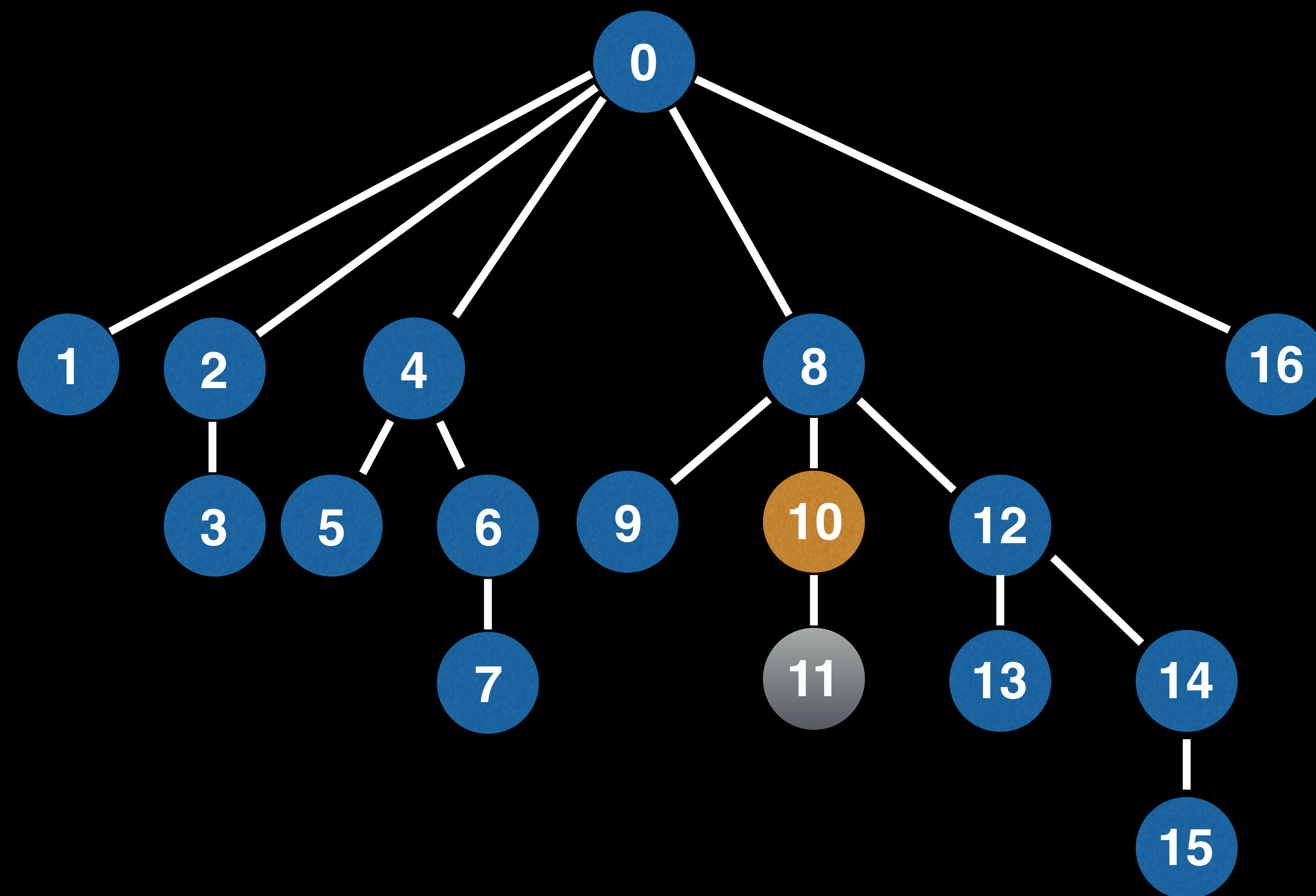


Find the sum in the array between [11,15]

(6 + −1 + −10 + 26) − (

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

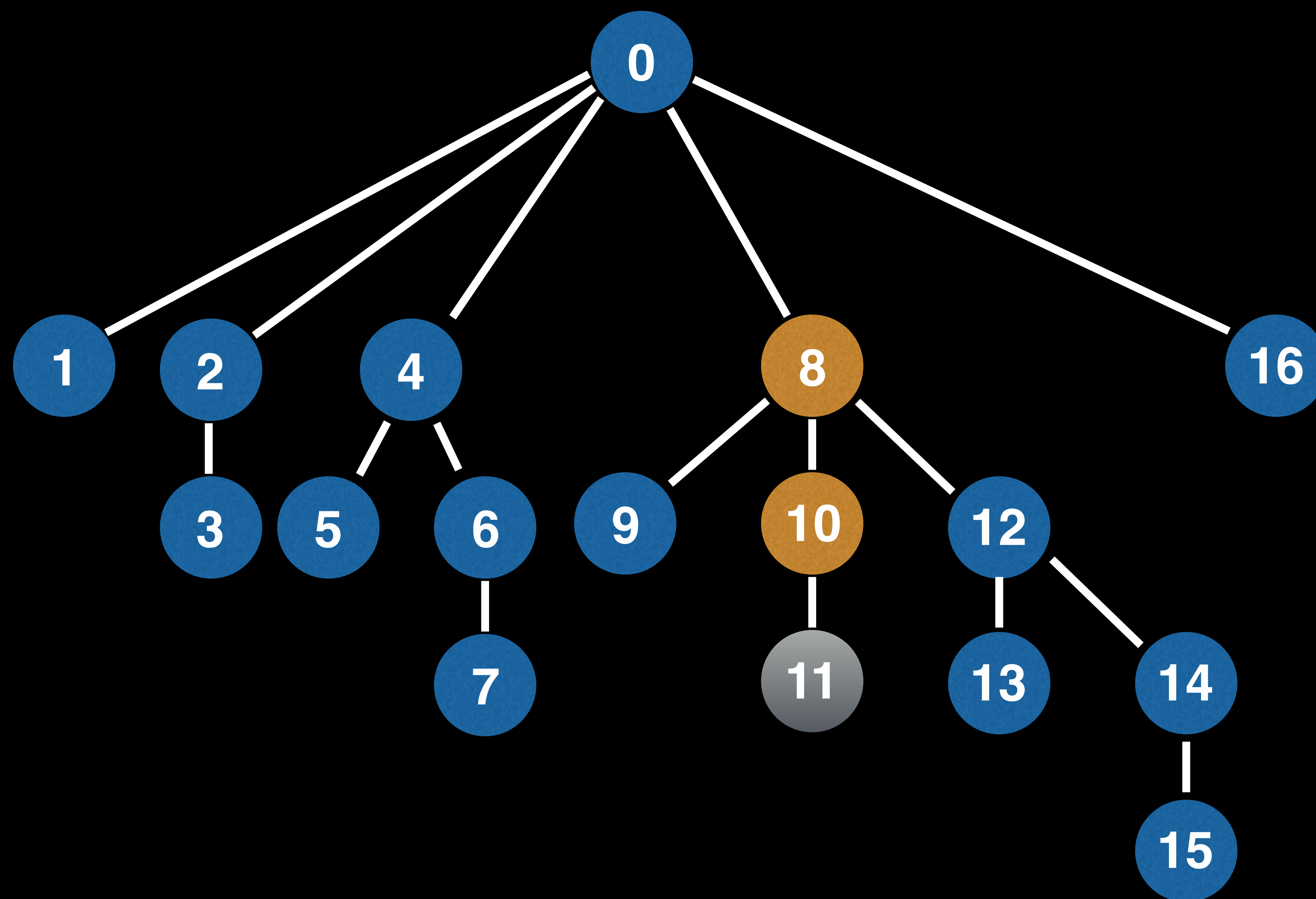


Find the sum in the array between [11,15]

(6 + −1 + −10 + 26) − (−11

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

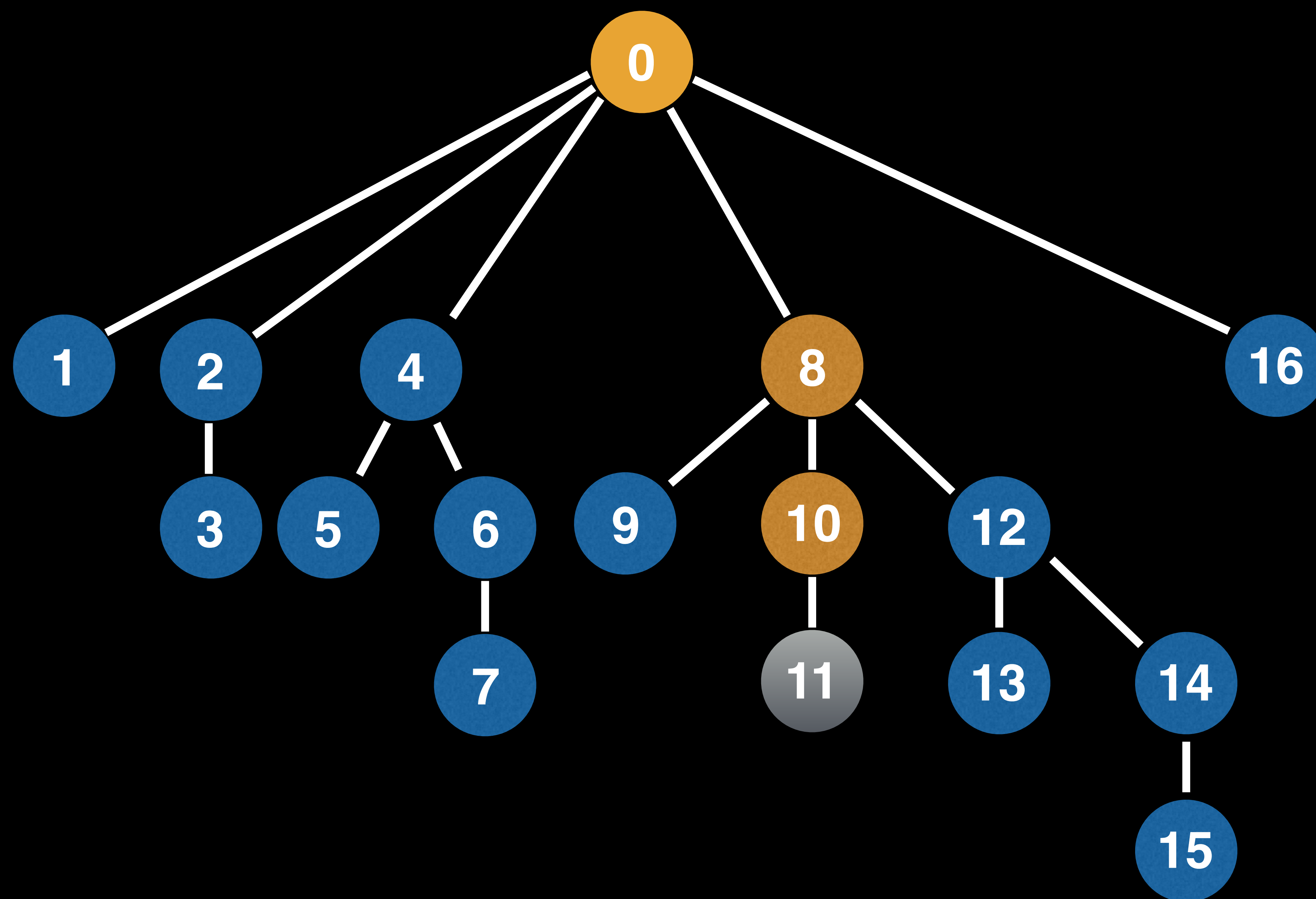


Find the sum in the array between [11,15]

$$(6 + -1 + -10 + 26) - (-11 + 26)$$

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

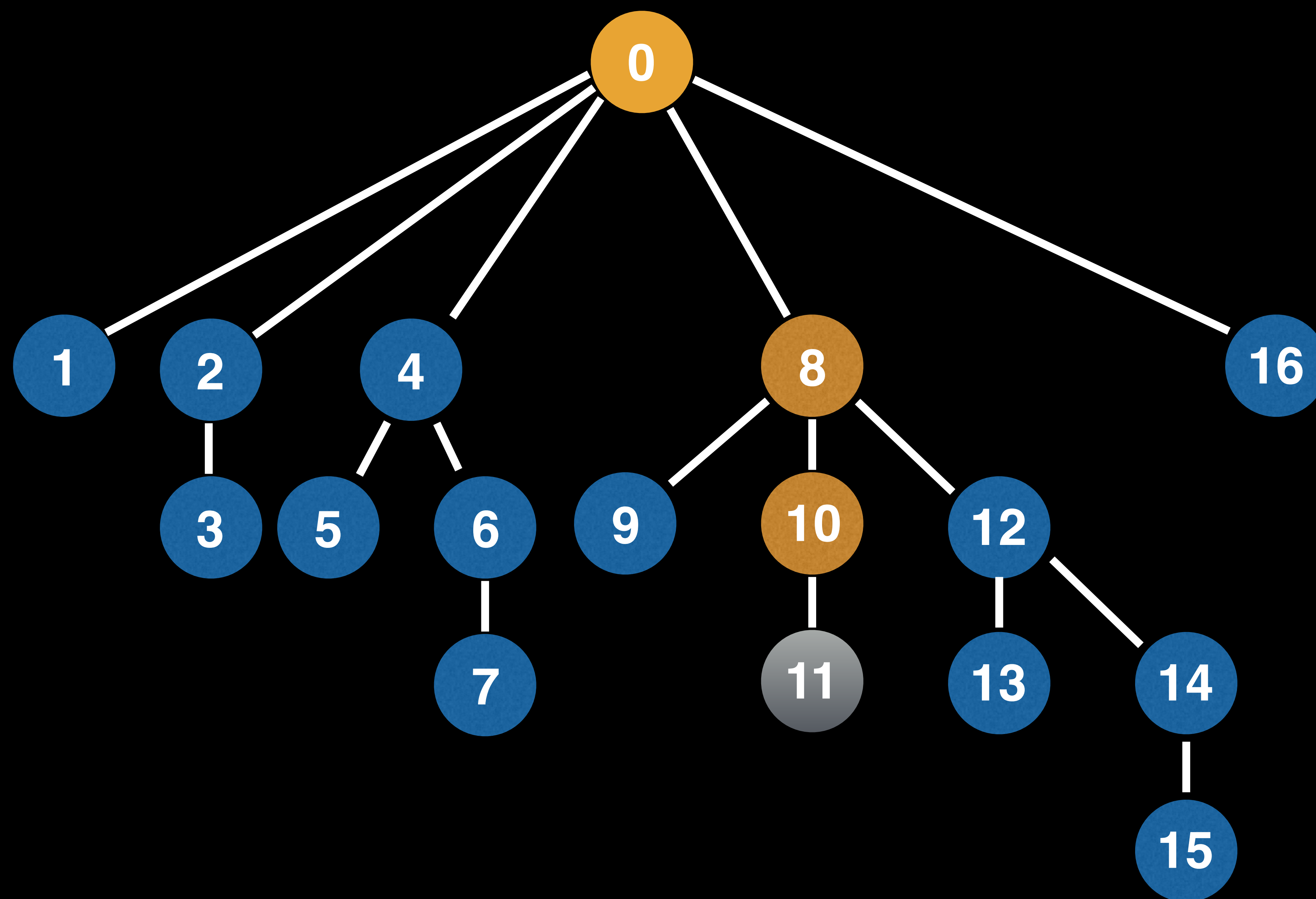


Find the sum in the array between [11,15]

$$(6 + -1 + -10 + 26) - (-11 + 26)$$

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7

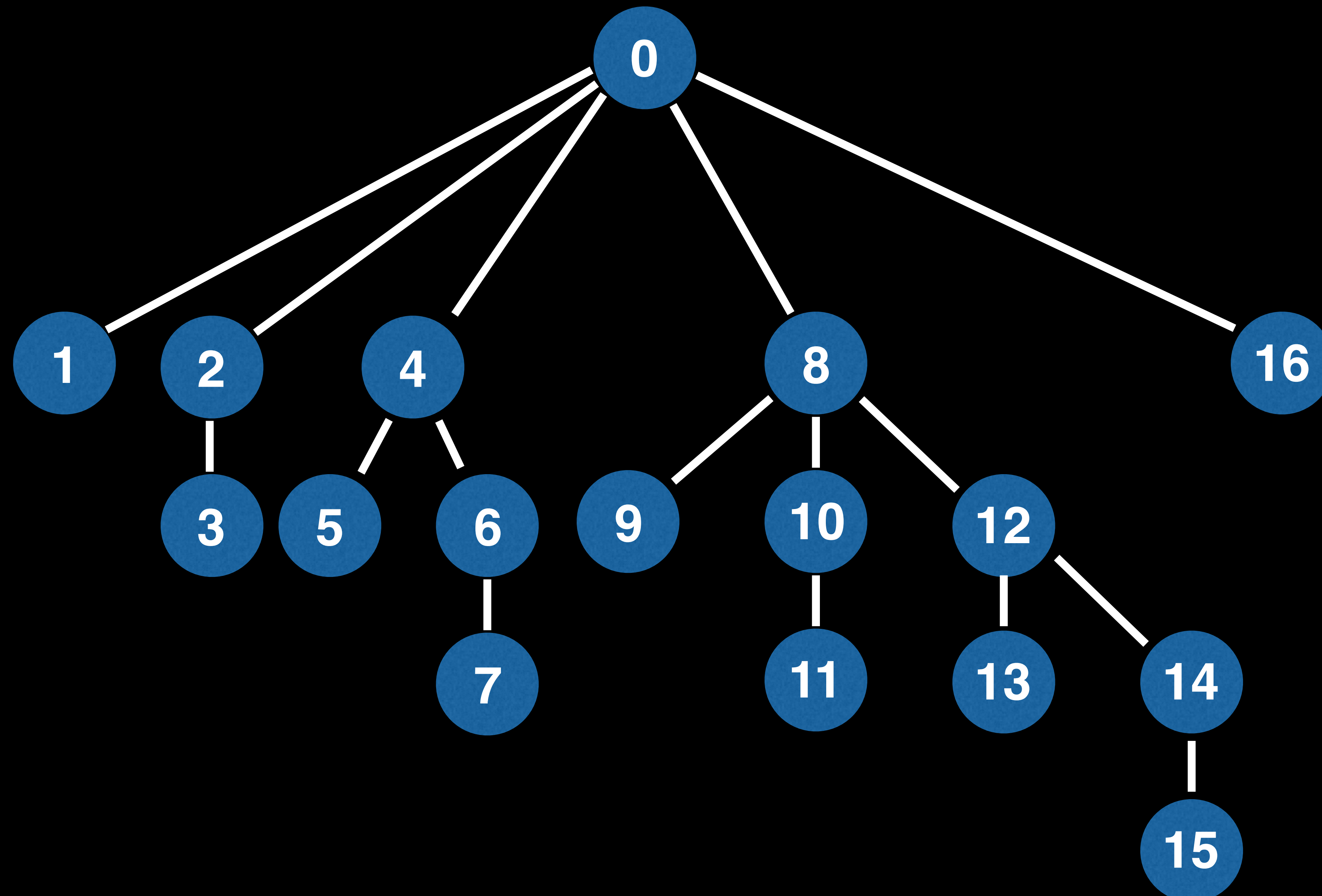


Find the sum in the array between [11,15]

$$(6 + -1 + -10 + 26) - (-11 + 26) = 6$$

16	10000	28
15	01111	6
14	01110	-1
13	01101	2
12	01100	-10
11	01011	9
10	01010	-11
9	01001	-5
8	01000	26
7	00111	11
6	00110	6
5	00101	2
4	00100	9
3	00011	-3
2	00010	5
1	00001	4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
N/A	4	1	-3	7	2	4	11	0	-5	-6	9	-8	2	-3	6	7



Find the sum in the array between [11,15]

$$(6 + -1 + -10 + 26) - (-11 + 26) = 6$$

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

Least Significant Bit

Idea: Place an edge between values who only differ by their **Least Significant Bit** (LSB) in binary.

Least Significant Bit

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

0

Idea: Place an edge between values who only differ by their **Least Significant Bit** (LSB) in binary.

Least significant bits

Least Significant Bit

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

0

Idea: Place an edge between values who only differ by their **Least Significant Bit** (LSB) in binary.

In other words: Place an edge between nodes i and i without its LSB.

Least Significant Bit

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

0

Idea: Place an edge between values who only differ by their **Least Significant Bit** (LSB) in binary.

In other words: Place an edge between nodes i and i without its LSB.

13

1101

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

Least Significant Bit

Idea: Place an edge between values who only differ by their **Least Significant Bit** (LSB) in binary.

In other words: Place an edge between nodes i and i without its LSB.

13 -> 12

1101 -> 1100

Least Significant Bit

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

0

Idea: Place an edge between values who only differ by their **Least Significant Bit** (LSB) in binary.

In other words: Place an edge between nodes i and i without its LSB.

13 -> 12 -> 8

1101 -> 1100 -> 1000

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

Least Significant Bit

0

Idea: Place an edge between values who only differ by their **Least Significant Bit** (LSB) in binary.

In other words: Place an edge between nodes i and i without its LSB.

13 -> 12 -> 8 -> 0

1101 -> 1100 -> 1000 -> 0000

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

Fenwick Tree Visualization

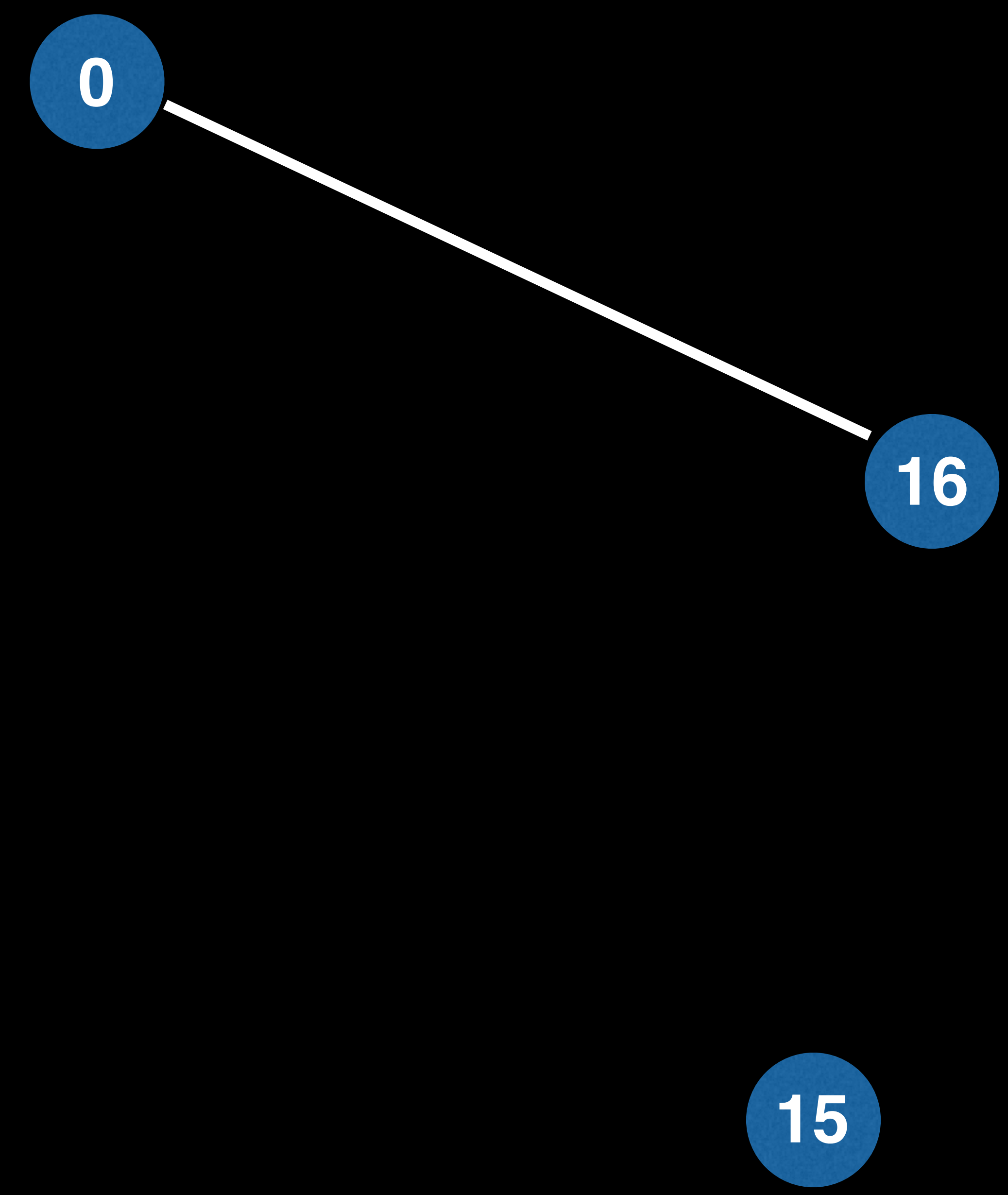
16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

16 -> 0

10000 -> 00000

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

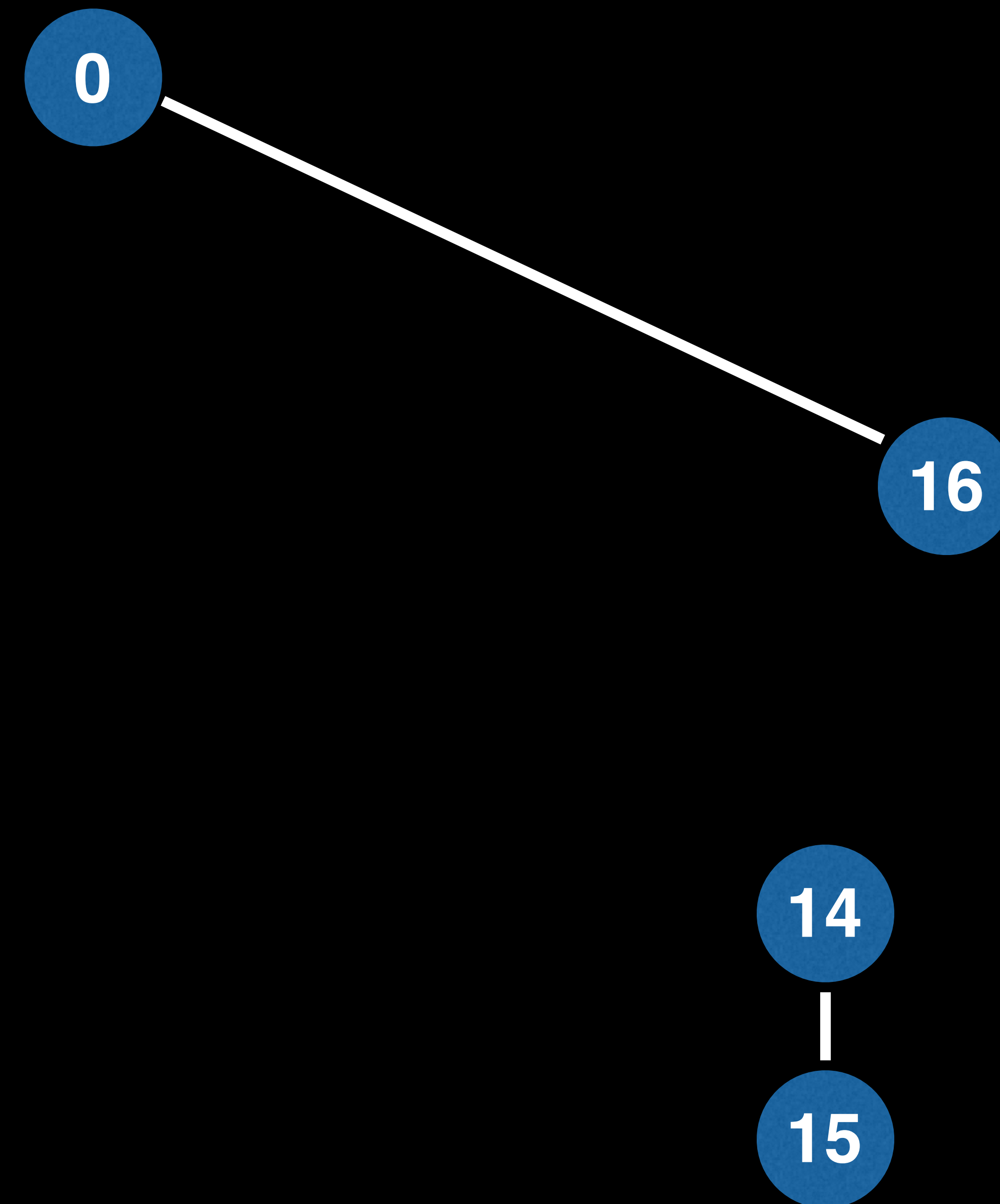


15

1111

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

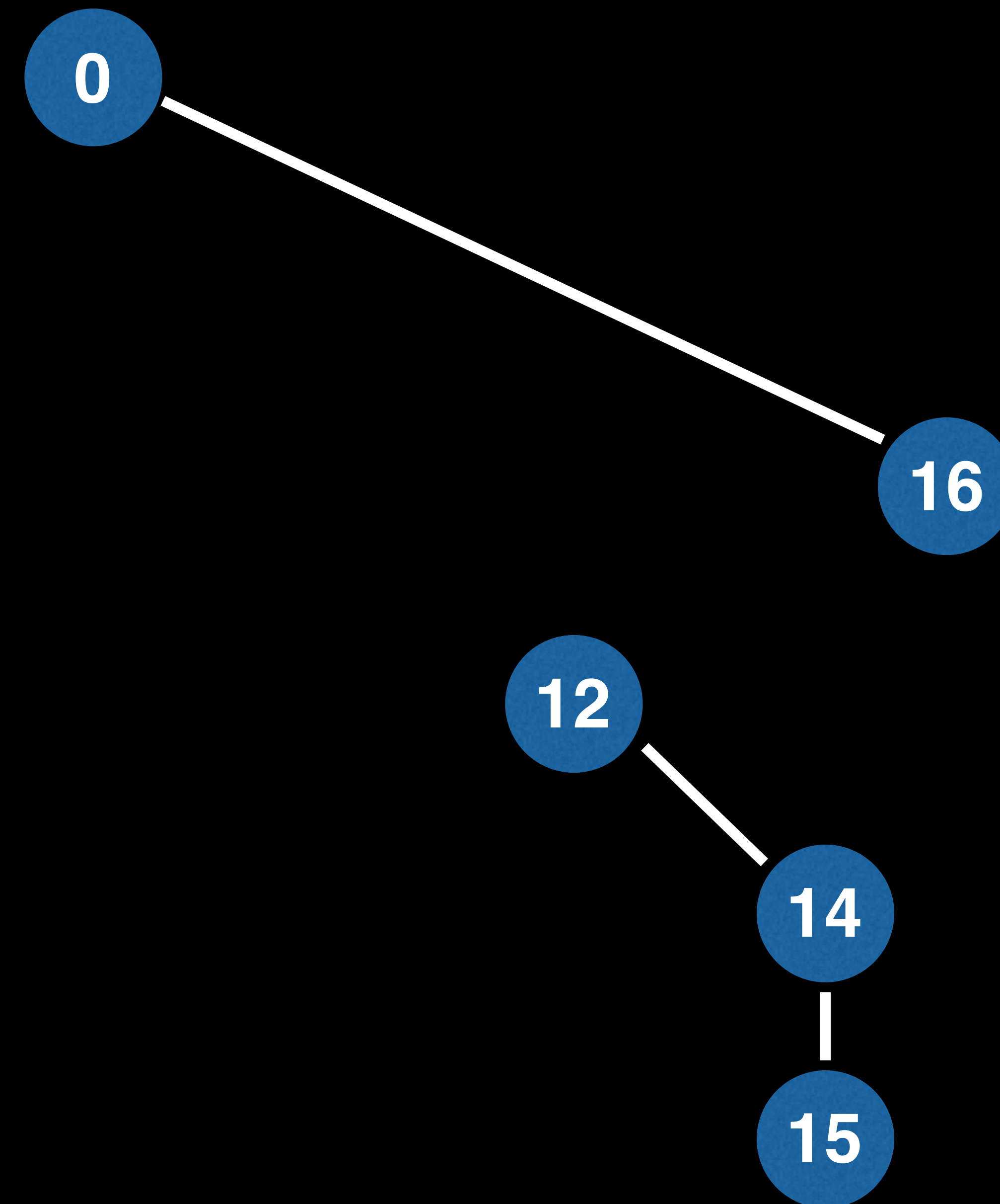


15 -> 14

1111 -> 1110

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

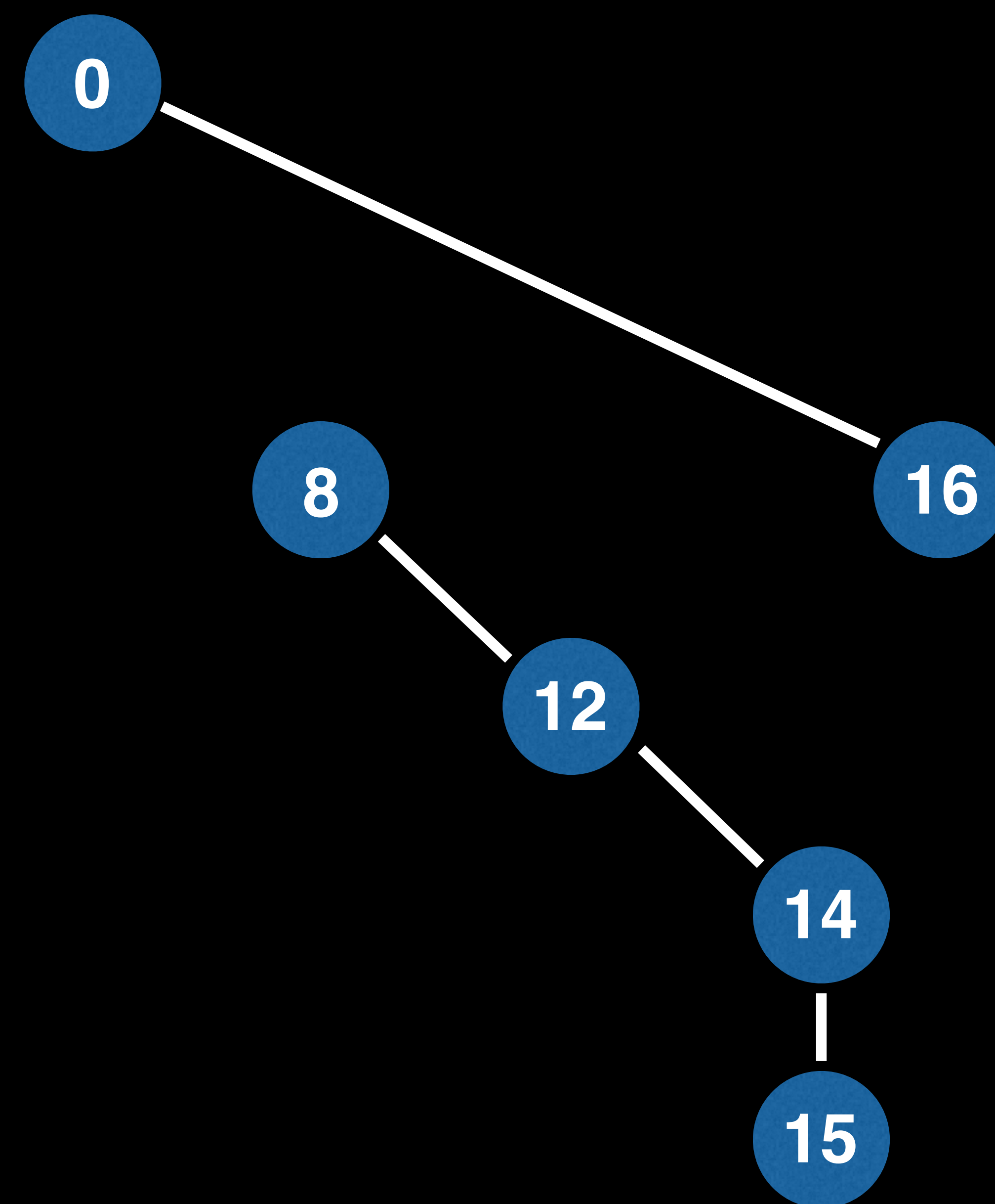


15 -> 14 -> 12

1111 -> 1110 -> 1100

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

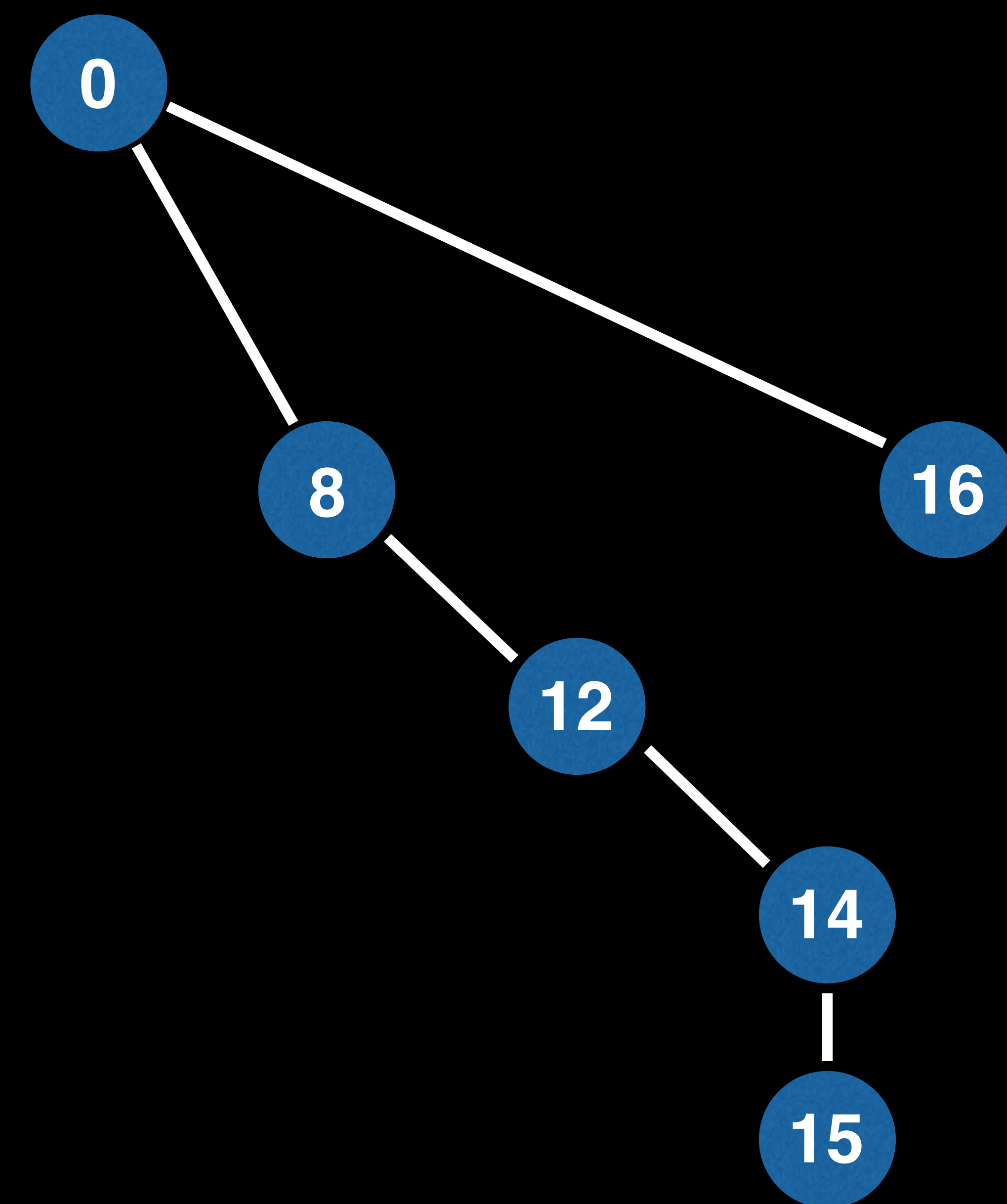


15 -> 14 -> 12 -> 8

1111 -> 1110 -> 1100 -> 1000

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

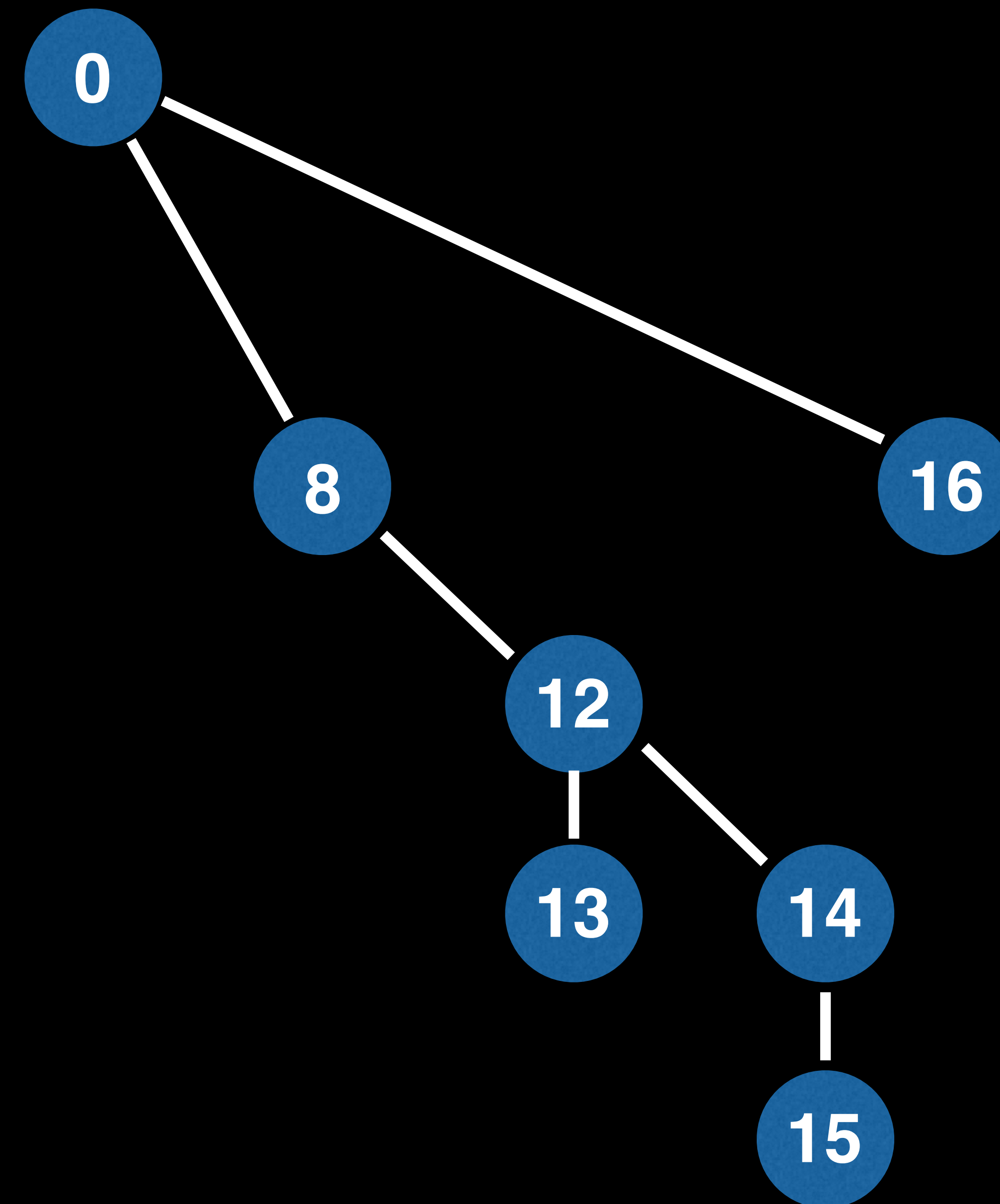


15 -> 14 -> 12 -> 8 -> 0

1111 -> 1110 -> 1100 -> 1000 -> 0000

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

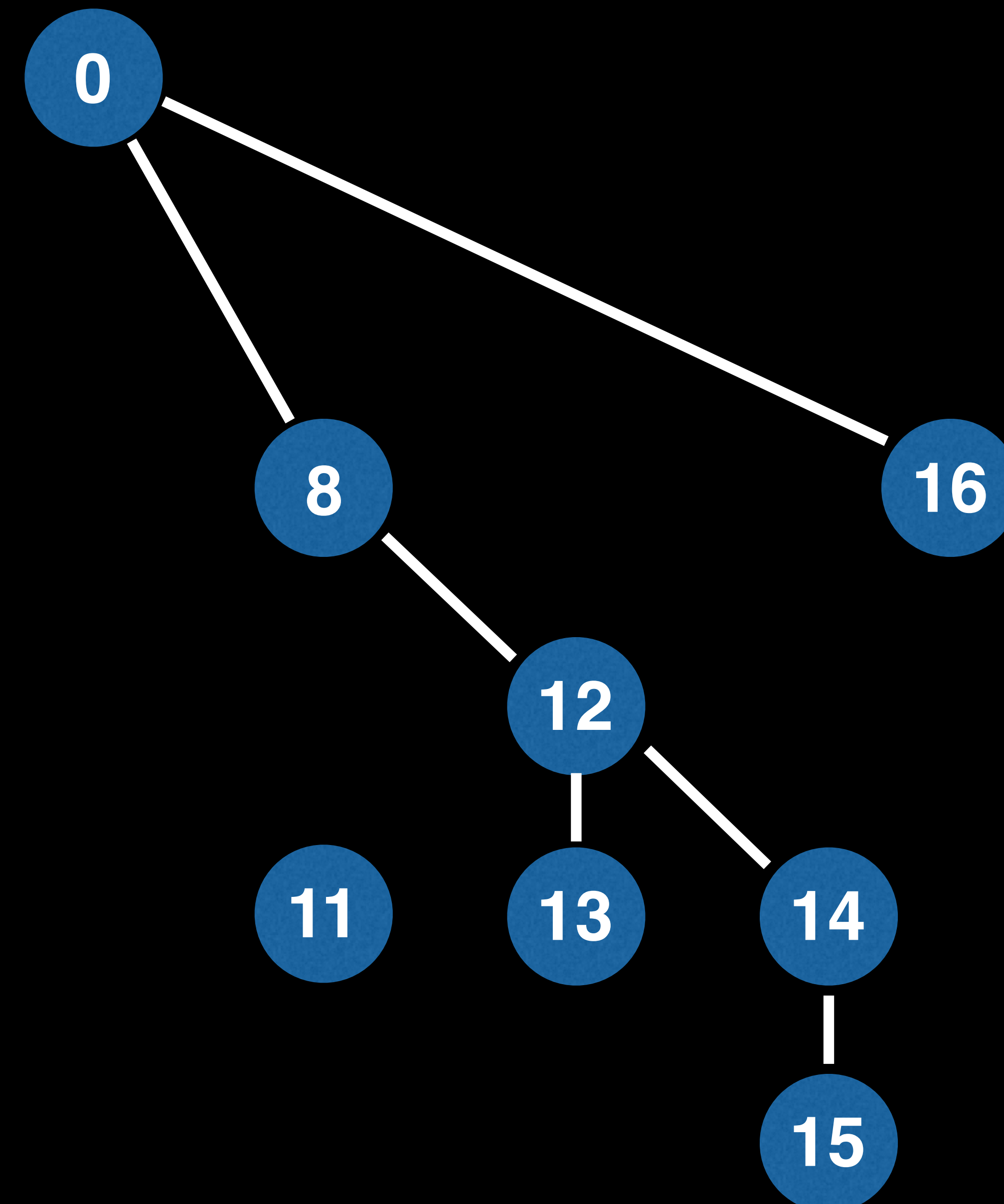


13 -> 12

1101 -> 1100

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

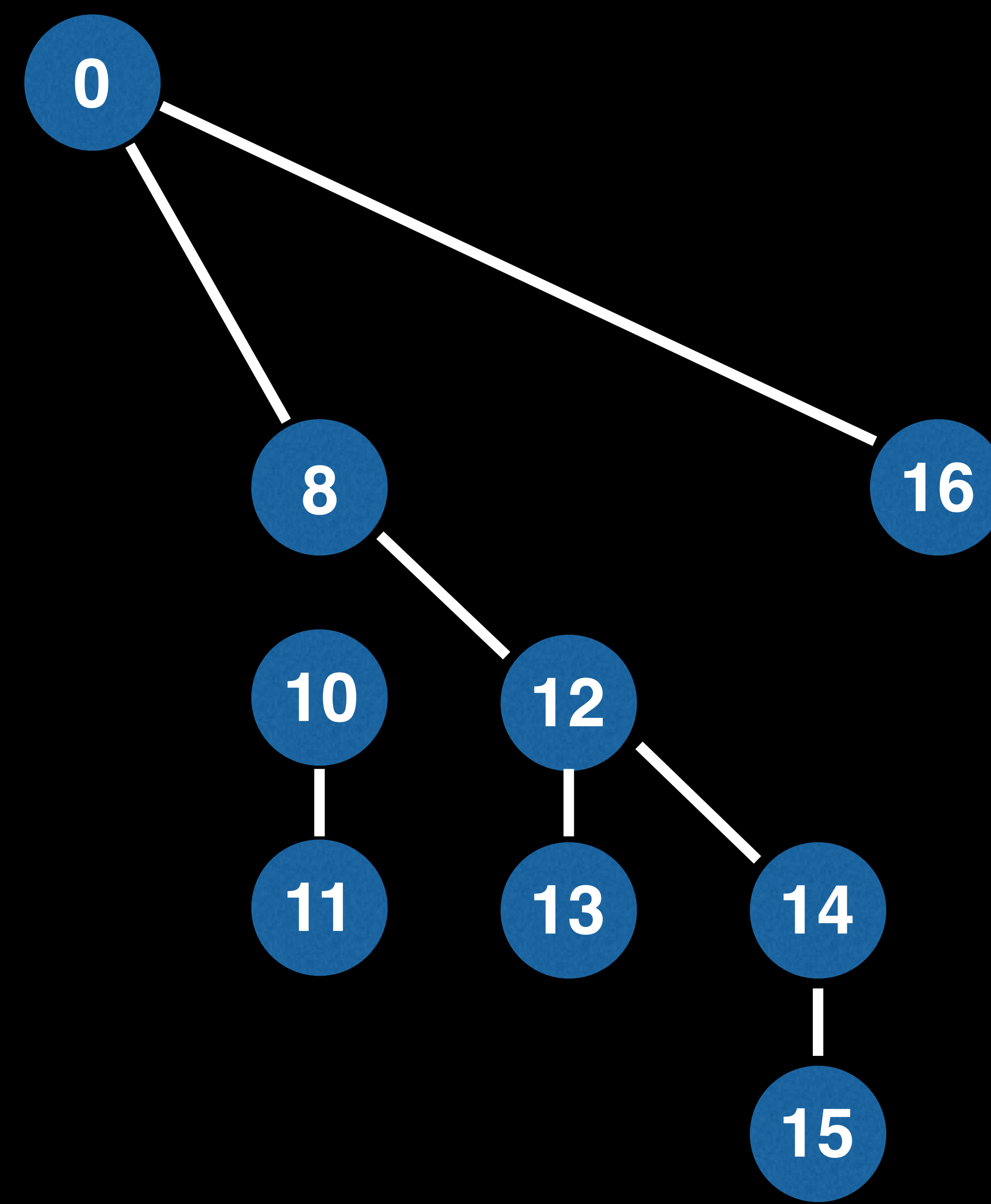


11

1011

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

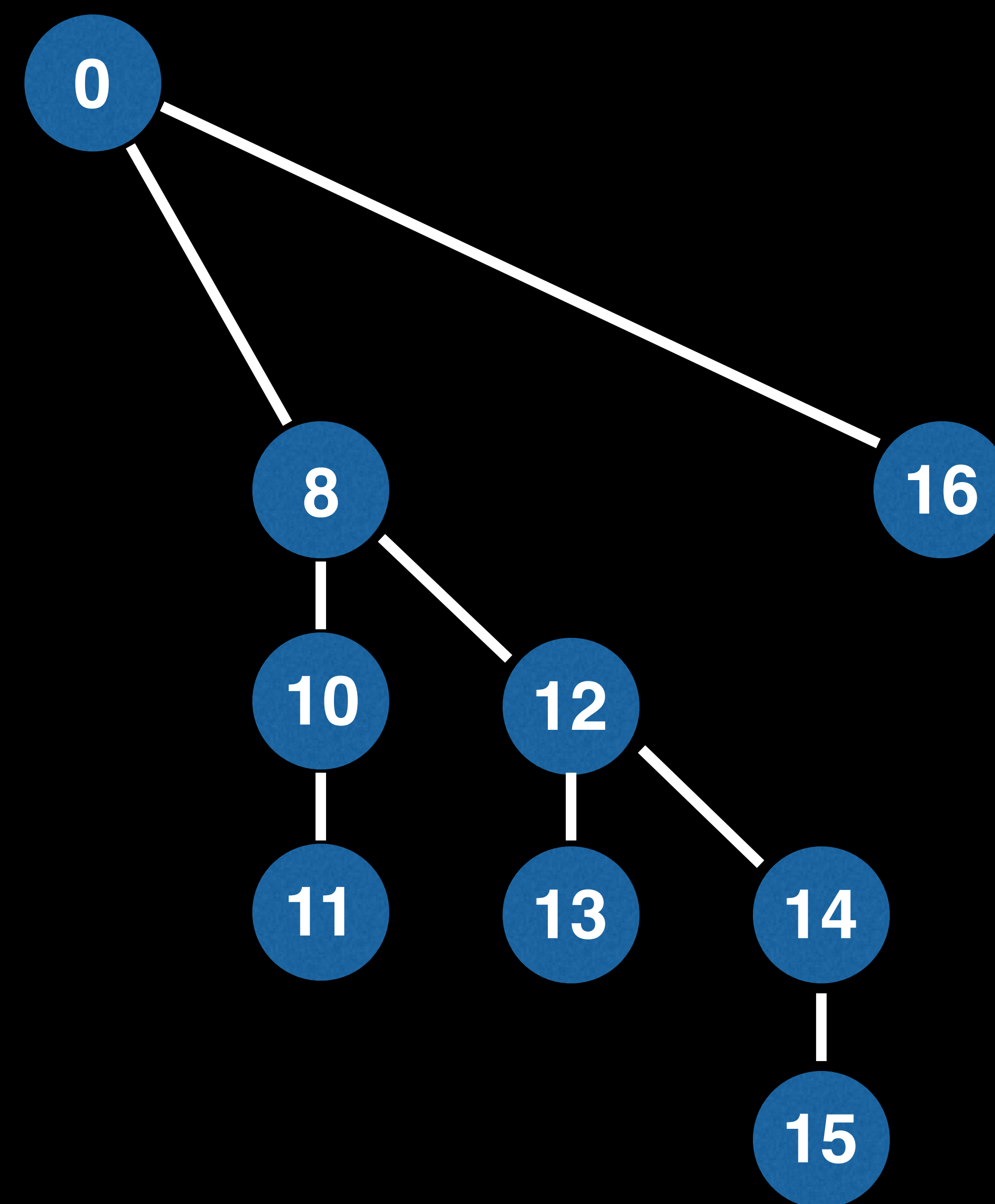


11 -> 10

1011 -> 1010

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

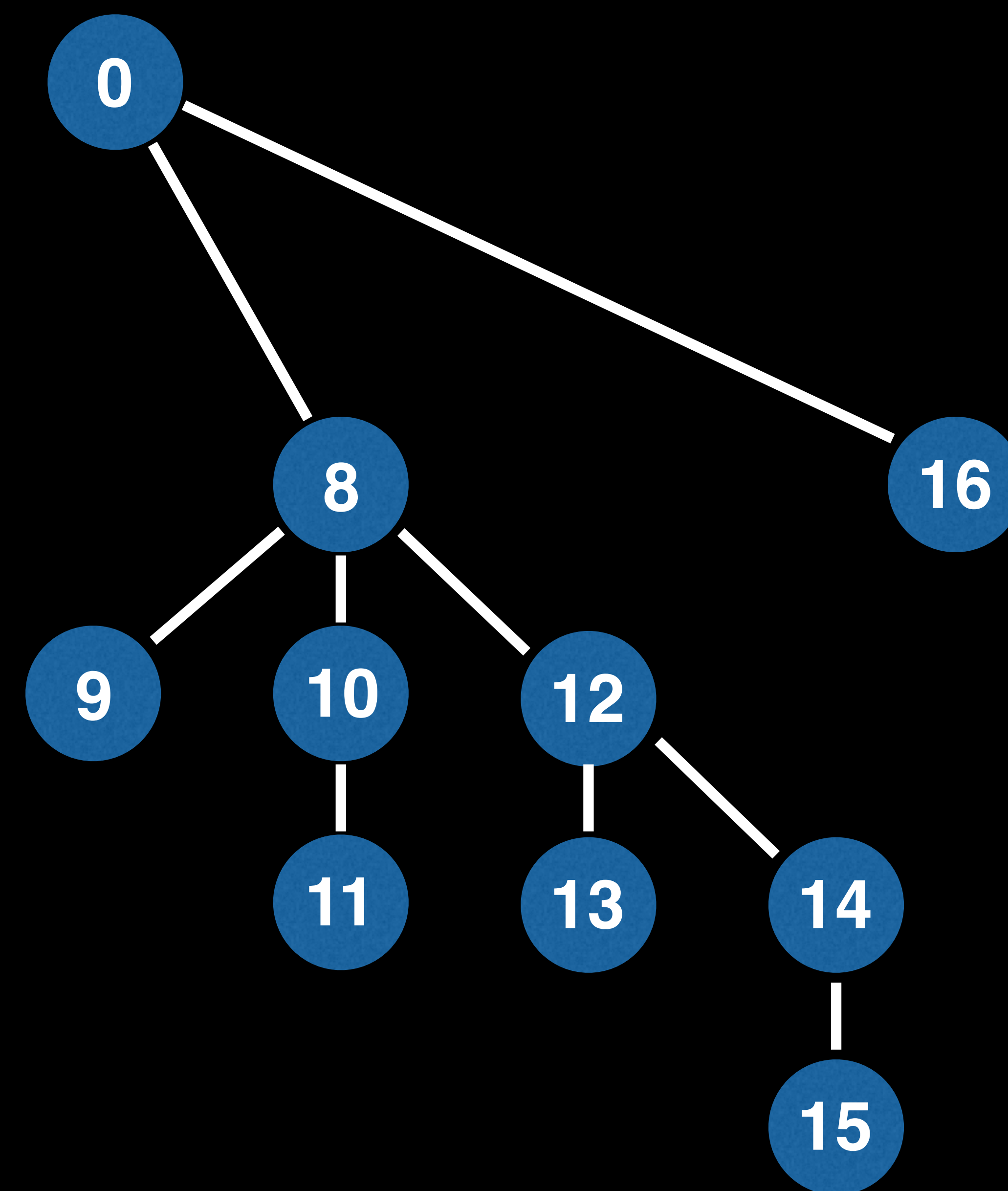


11 -> 10 -> 8

1011 -> 1010 -> 1000

Fenwick Tree Visualization

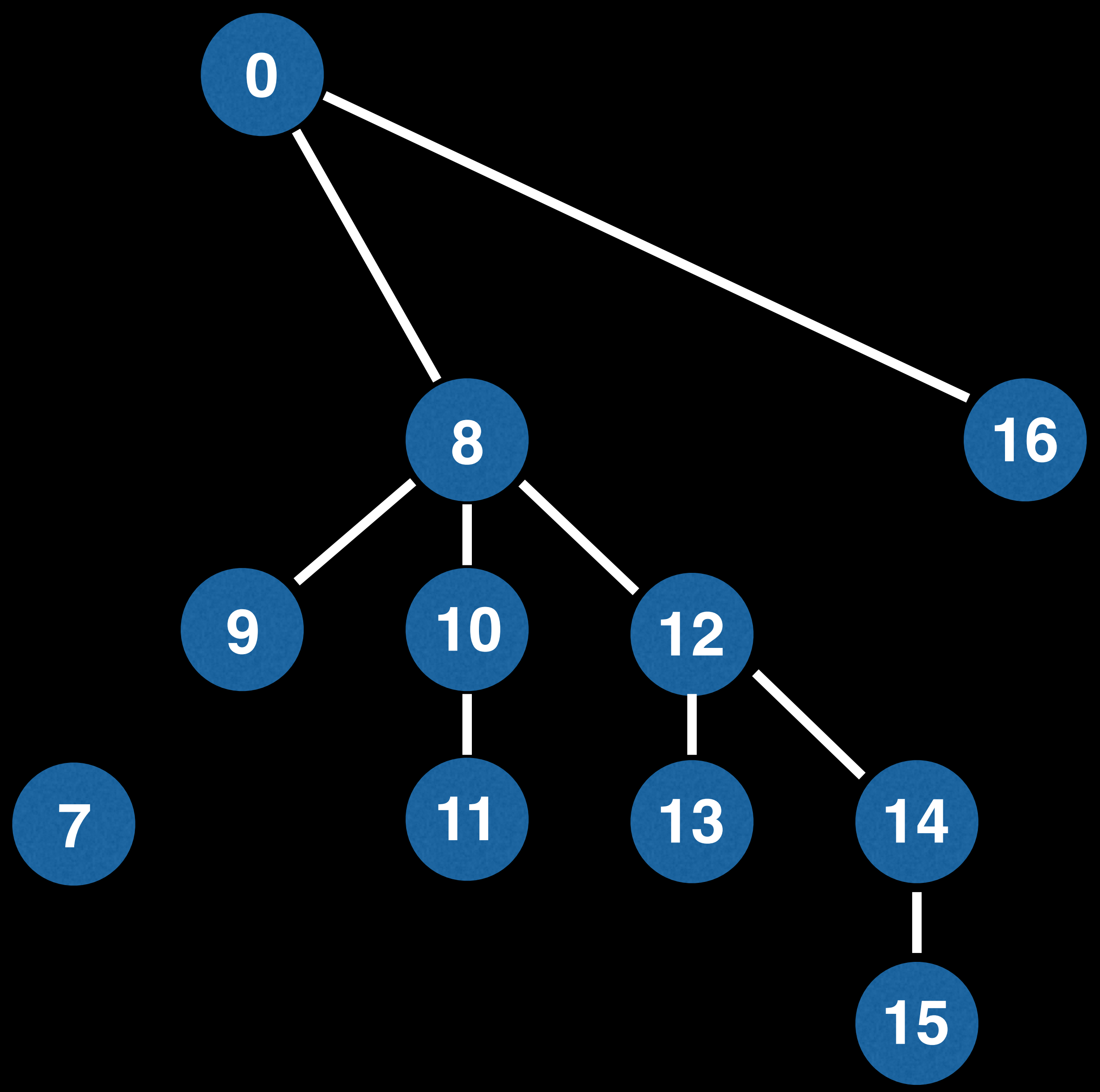
16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001




```
   9   ->   8
1001 -> 1000
```

Fenwick Tree Visualization

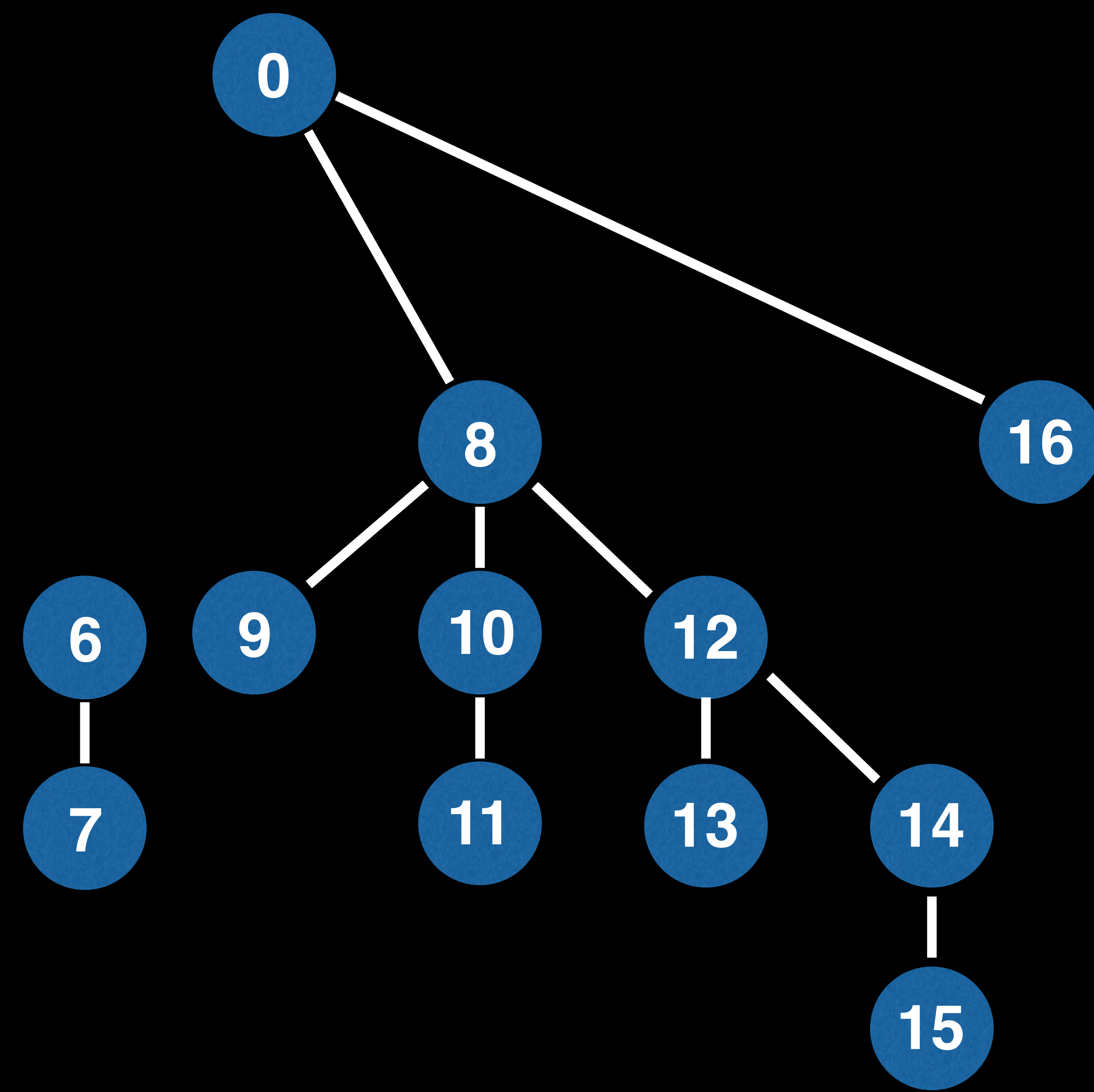
16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001



7
0111

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

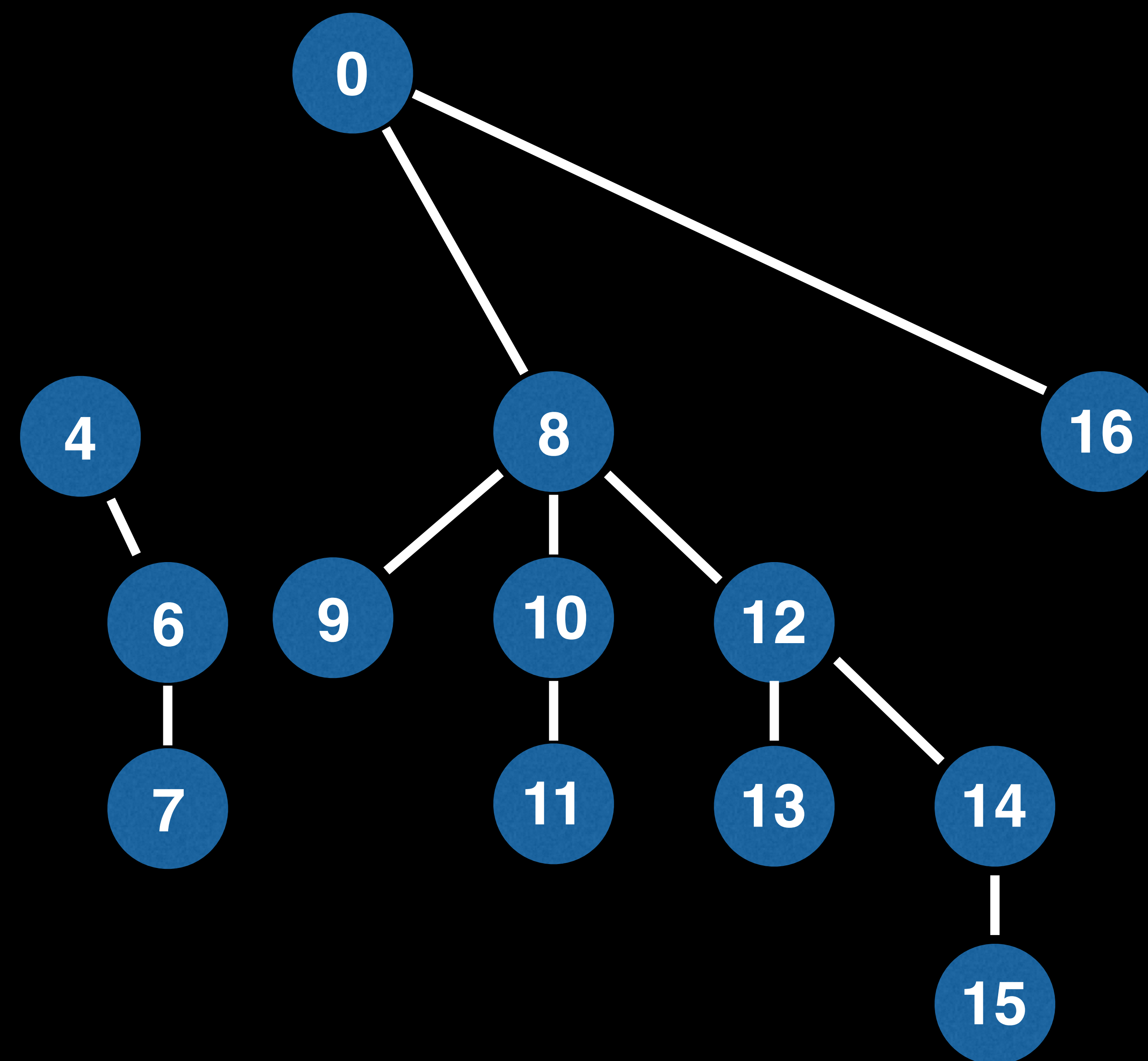


7 -> 6

0111 -> 0110

Fenwick Tree Visualization

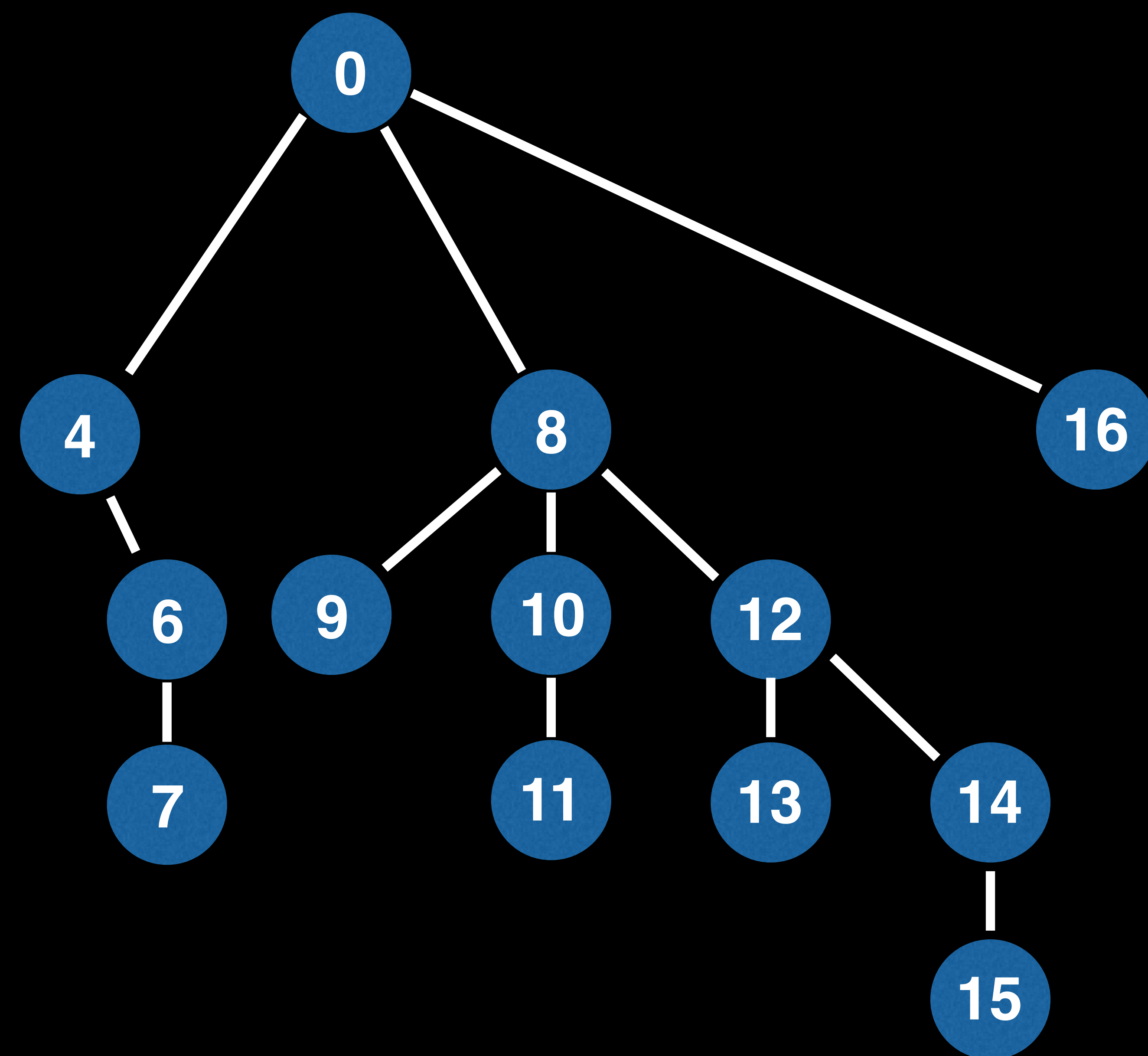
16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001




```
 7   ->   6  ->  4
0111 -> 0110 -> 0100
```

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

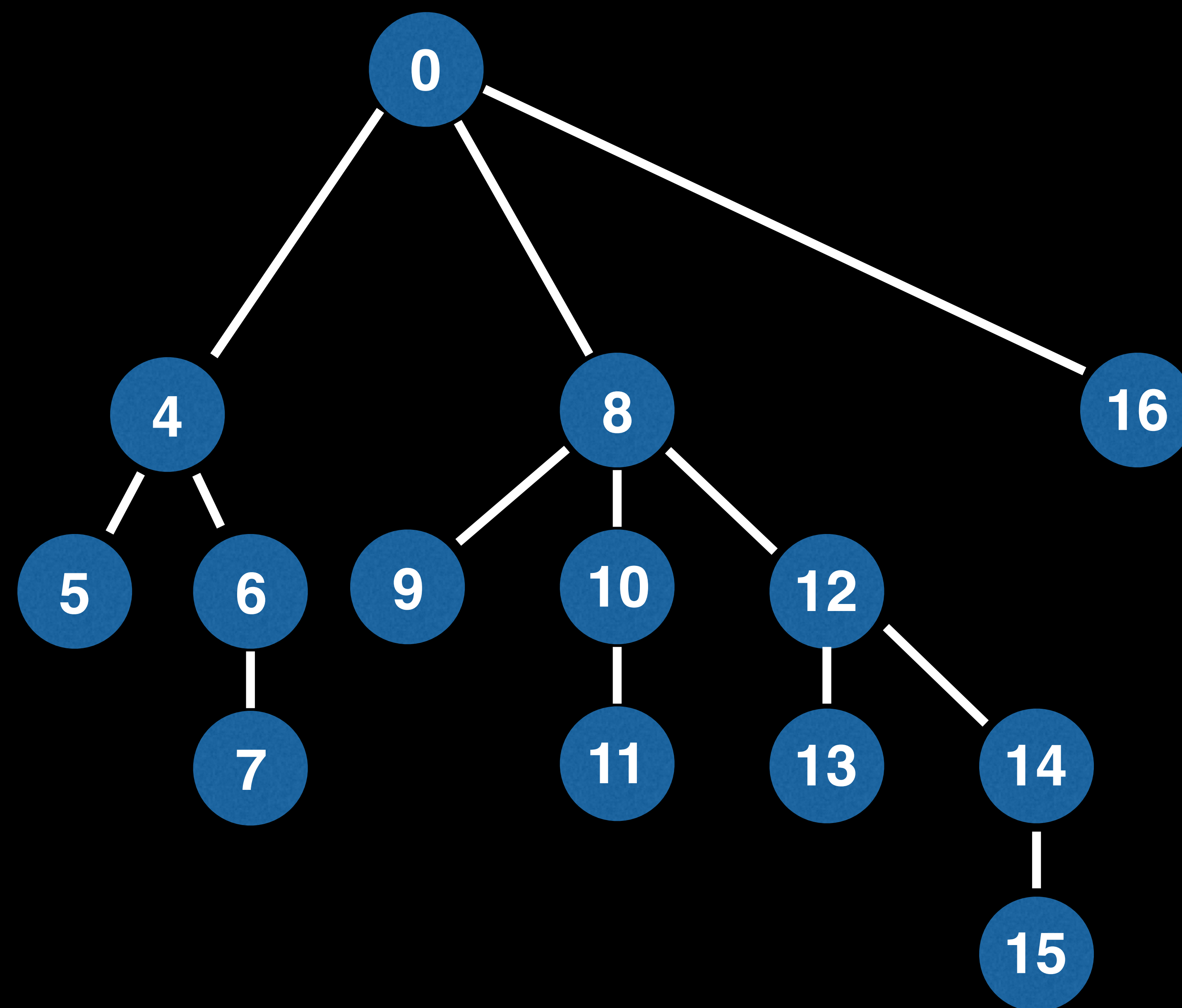


7 -> 6 -> 4 -> 0

0111 -> 0110 -> 0100 -> 0000

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

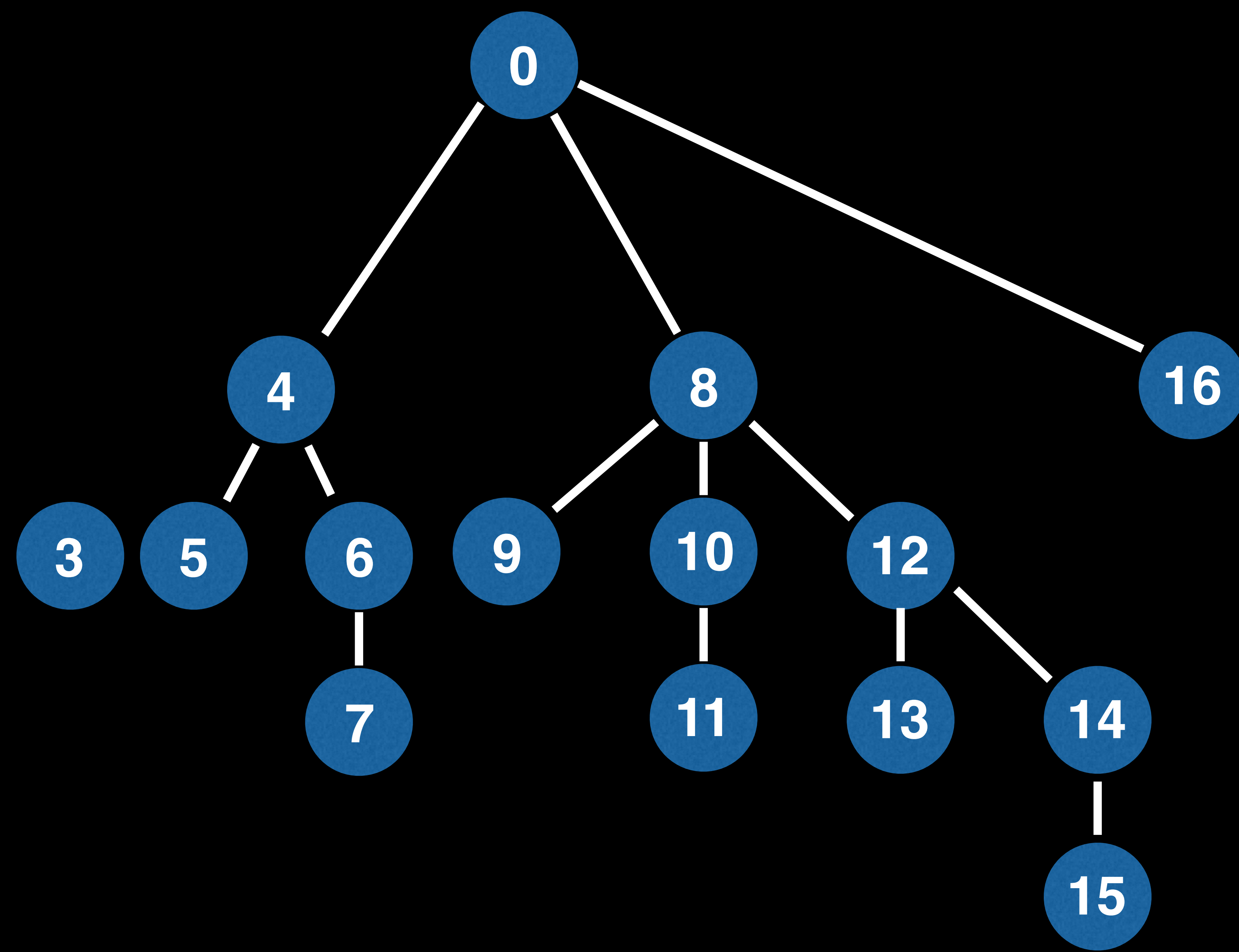


5 -> 4

0101 -> 0100

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

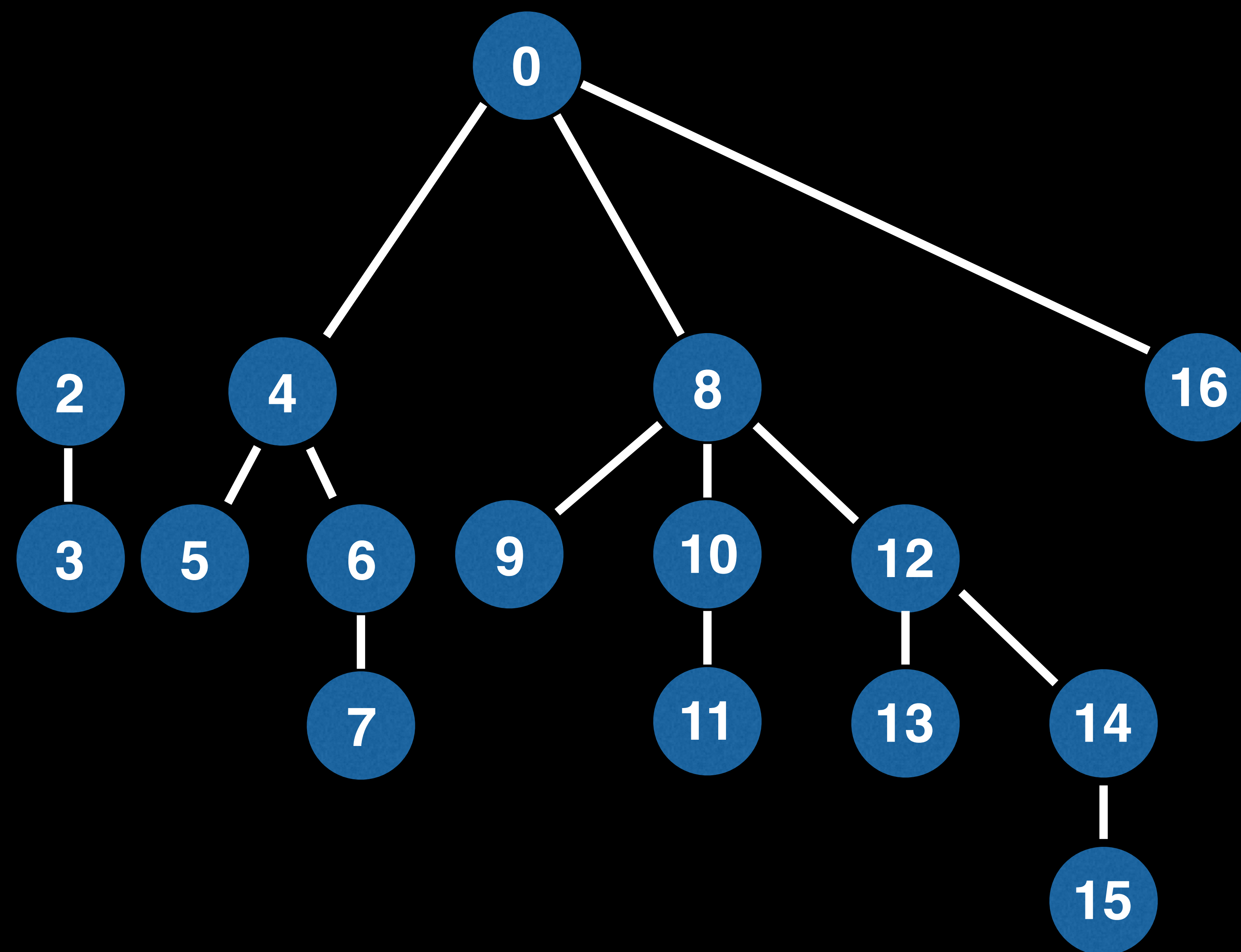


3

0011

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

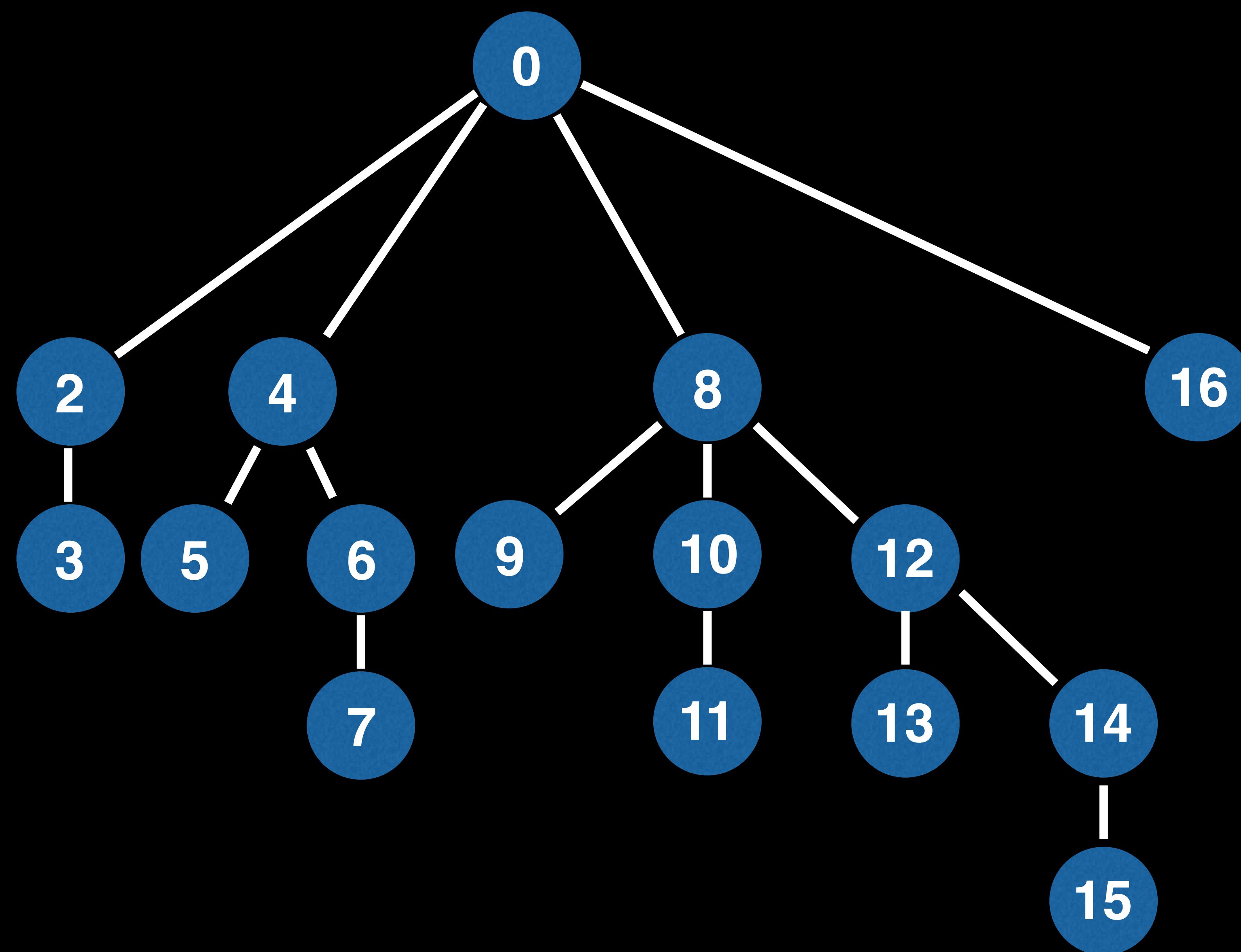


3 -> 2

0011 -> 0010

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

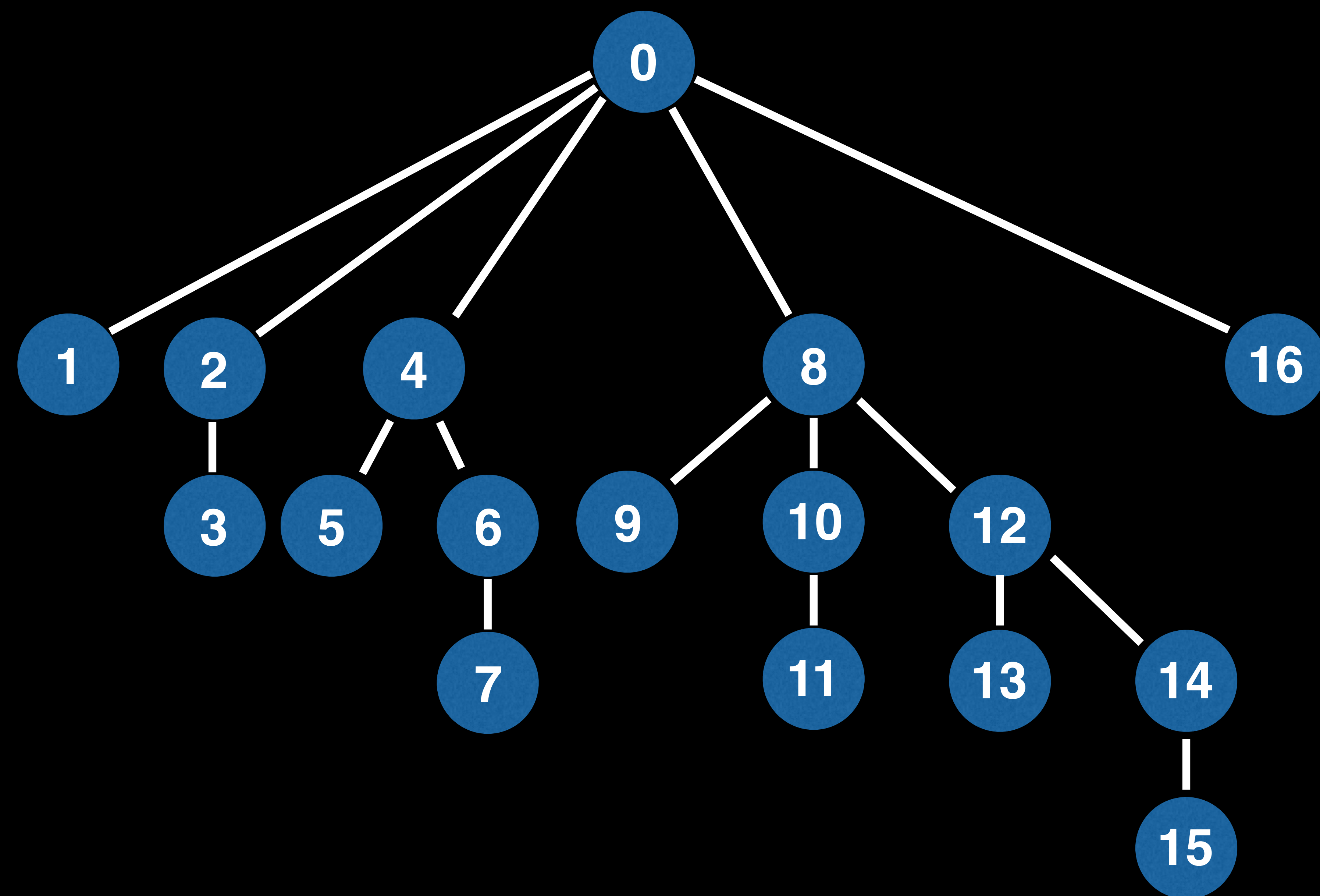


3 -> 2 -> 0

0011 -> 0010 -> 0000

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001

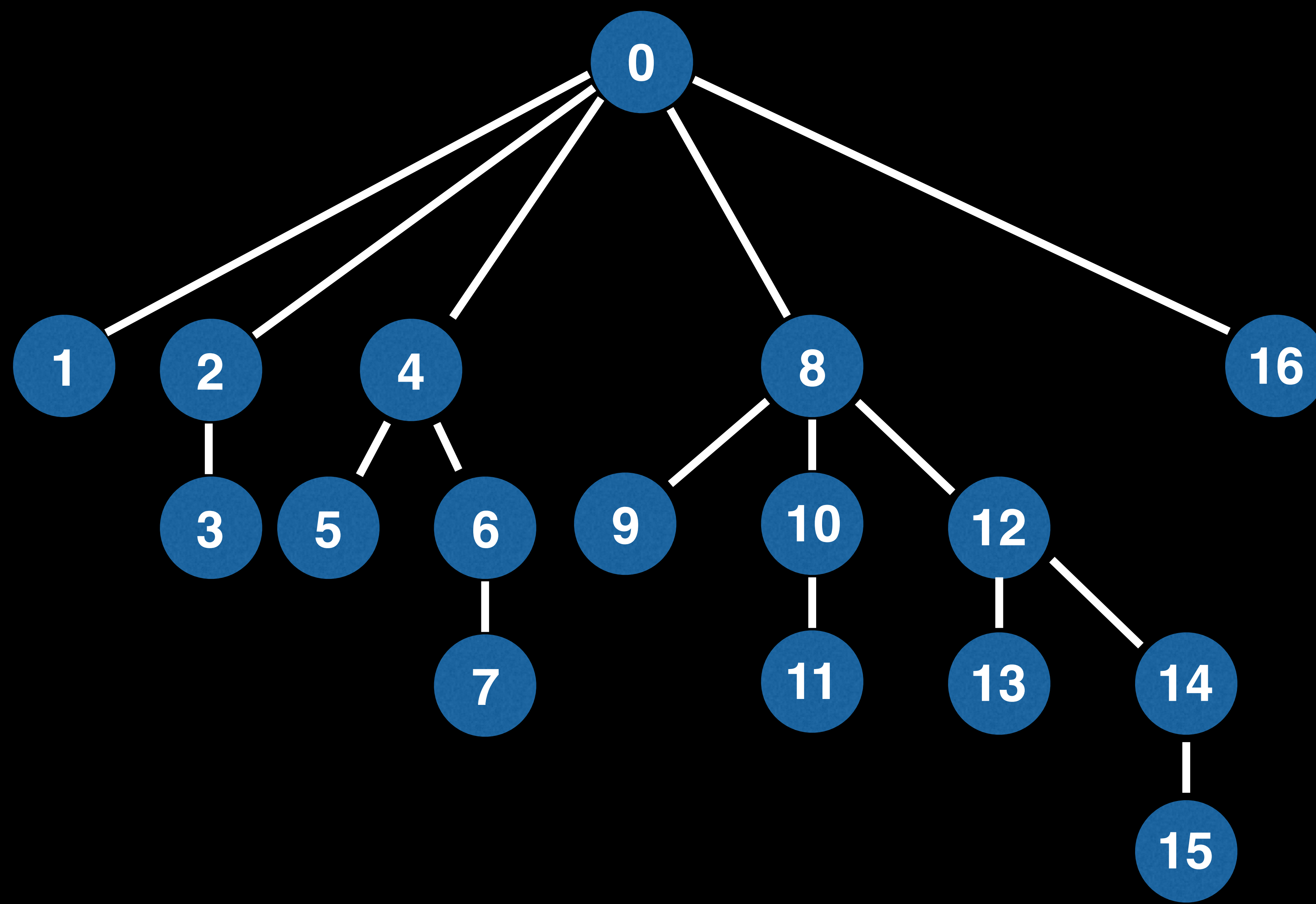


1 -> 0

0001 -> 0000

Fenwick Tree Visualization

16	10000
15	01111
14	01110
13	01101
12	01100
11	01011
10	01010
9	01001
8	01000
7	00111
6	00110
5	00101
4	00100
3	00011
2	00010
1	00001



Fenwick Tree Analysis

Although the values contained by different Fenwick trees may differ, the actual structure of the tree does not depend on the values it is holding.

Fenwick Tree Analysis

The furthest node from the root will always be at most $\log_2(n)$ nodes deep. This happens when all the trailing bits are 1’s. These numbers are of the form 2^n-1.

$2^1-1 = 1 = 0b000001$
$2^2-1 = 3 = 0b000011$
$2^3-1 = 7 = 0b000111$
$2^4-1 = 15 = 0b001111$
$2^5-1 = 31 = 0b011111$
$2^6-1 = 63 = 0b111111$

Fenwick Tree Analysis

Odd nodes are always leaves in our Fenwick Tree because their LSB is always a 1.