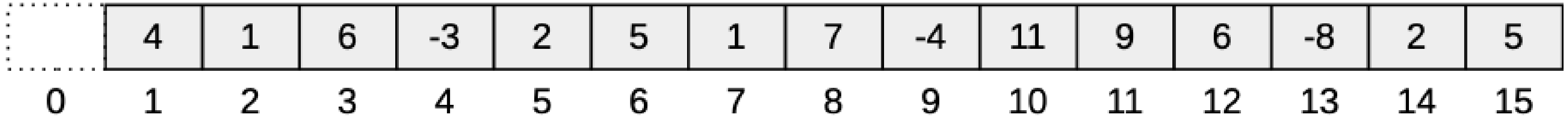
Competitive Programming

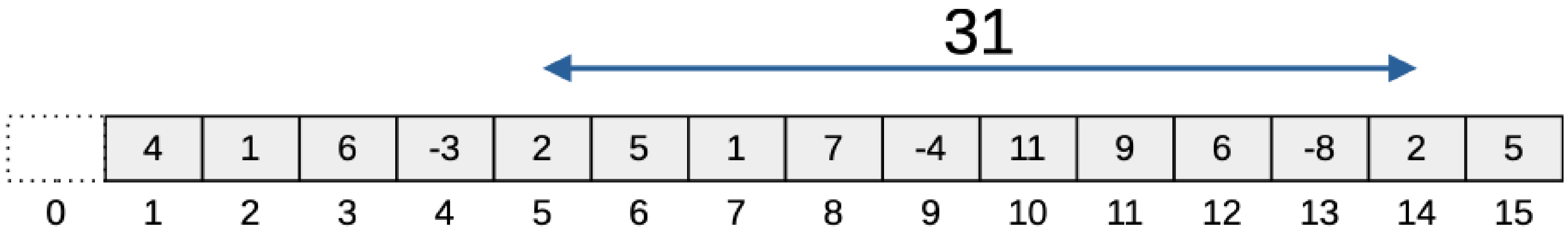
Fenwick Tree Data Structure

Range Sums

- Assume we have a sequence of values.



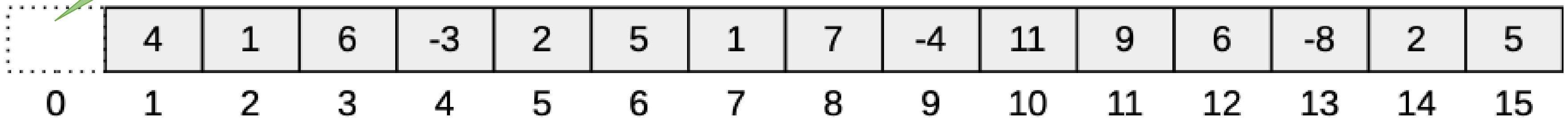
- We'd like a quick way to compute the sum of any contiguous range.



Range Sums

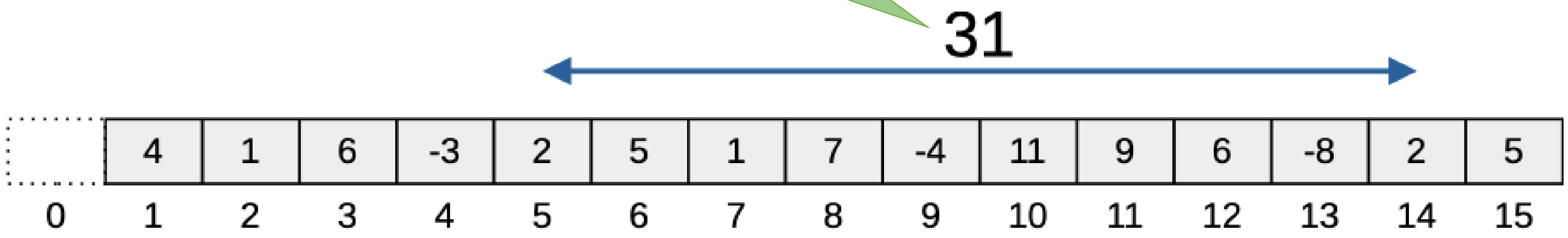
I'm skipping element 0. Not generally needed but it will simplify the Fenwick Tree later.

- Assume we have … ues.



Obviously, we could do this in linear time.

- We'd like a quick way to com … f any contiguous range.



Range Sums via Prefix Sums

- We could compute a *prefix sum* for every element.
- For each i > 0, sum all elements up to index i

Original sequence.

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

Prefix sums

	4	5	11	8	10	15	16	23	19	30	39	45	37	39	44
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

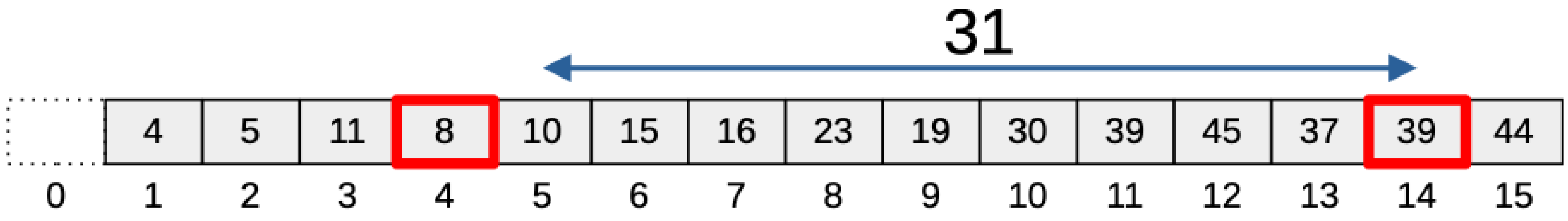
- We can compute this in linear time.

Range Sums via Prefix Sums

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

	4	5	11	8	10	15	16	23	19	30	39	45	37	39	44
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- We can compute any range as the difference of two prefix sums.
    - In constant time!



Range Sums via Prefix Sums

- This trick is worth a lot.
- Generalizes to 2 or more dimensions.

5	-2	0	10
6	9	13	4
1	7	-5	8
4	3	-3	2

5	3	3	13
11	18	31	45
12	26	34	56
16	33	38	62

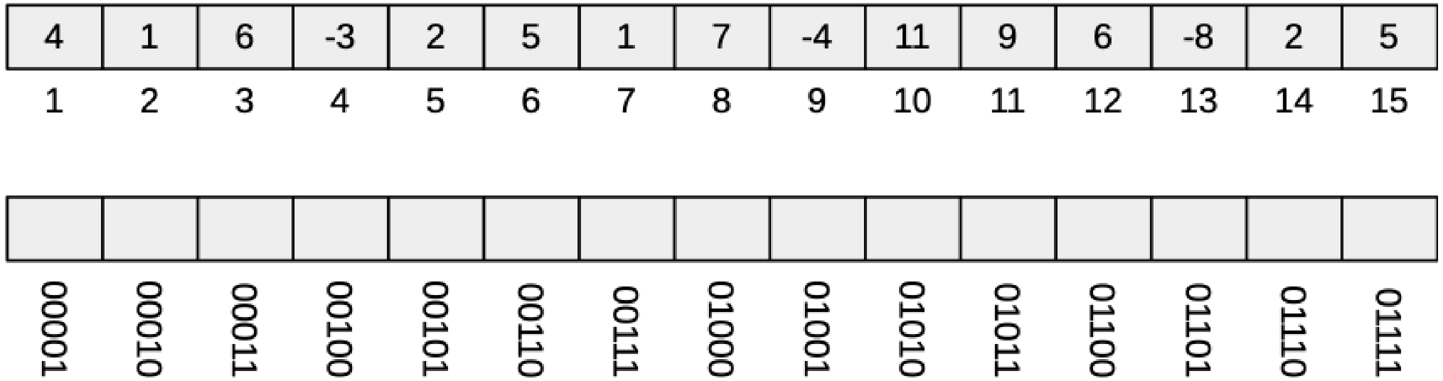
Range Sums via Prefix Sums

- Doesn't efficiently handle changes to the sequence.

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

	4	5	11	8	10	15	16	23	20	31	40	46	38	40	45
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

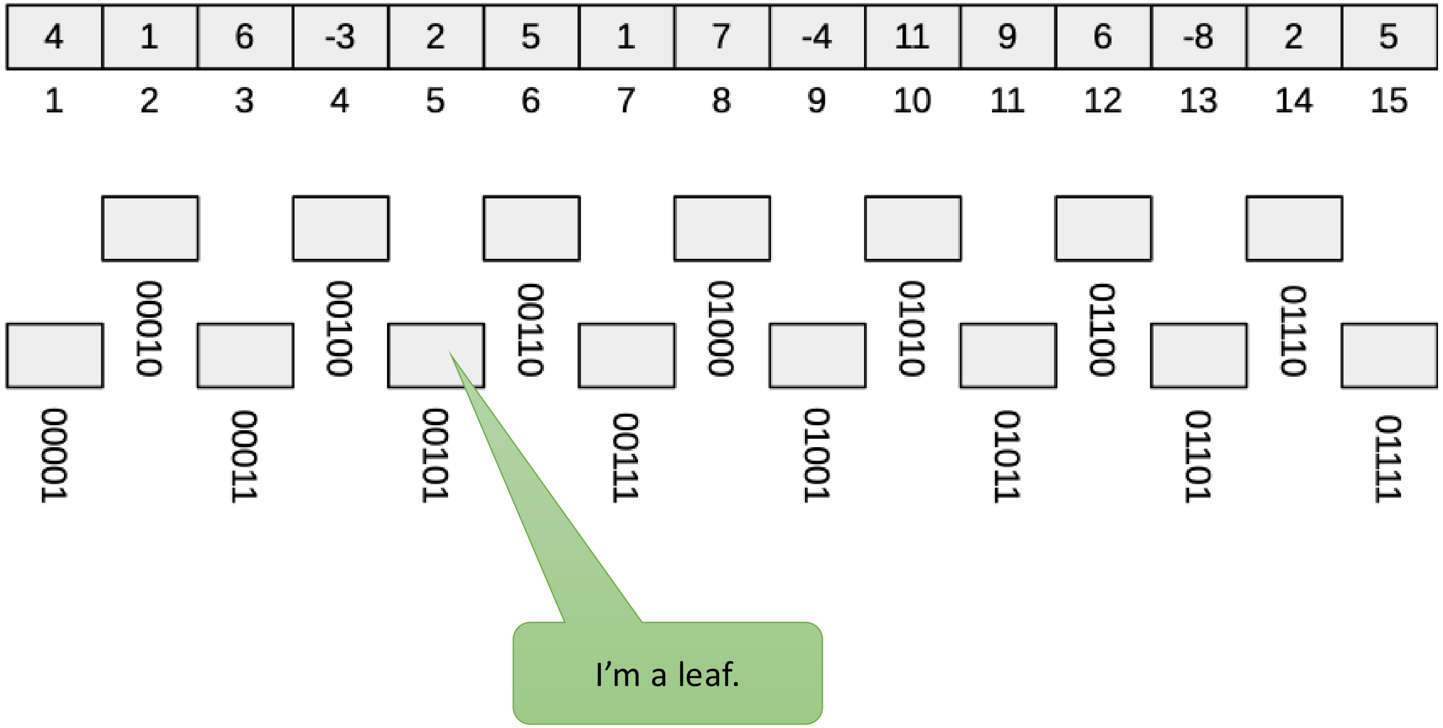
Fenwick Tree



- Cover the sequence with sums of different ranges of values.
- Structure reflects the index of each cell in binary
- Also called a *Binary Indexed Tree*

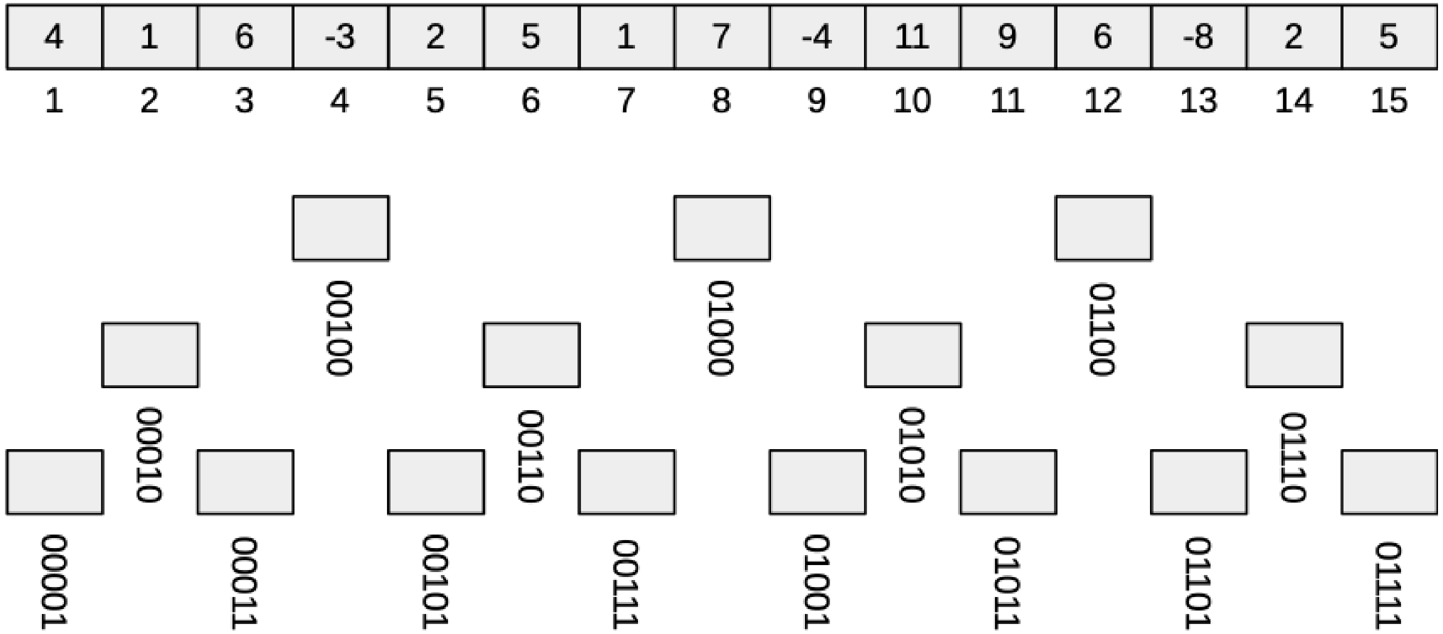
Fenwick Tree

- Indices with a 1 in the least-significant bit are the leaves.



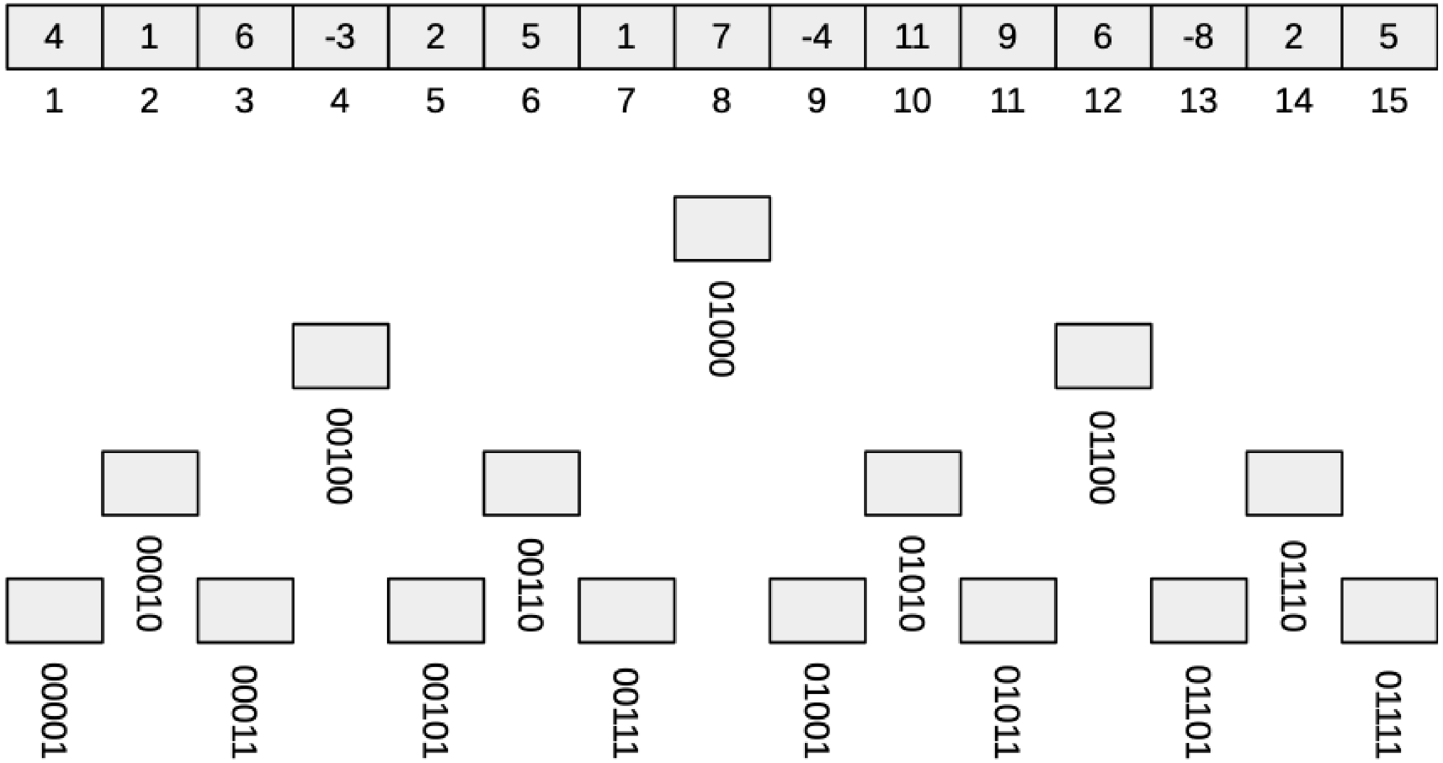
Fenwick Tree

- Indices with a 1 in the next bit are their parents.



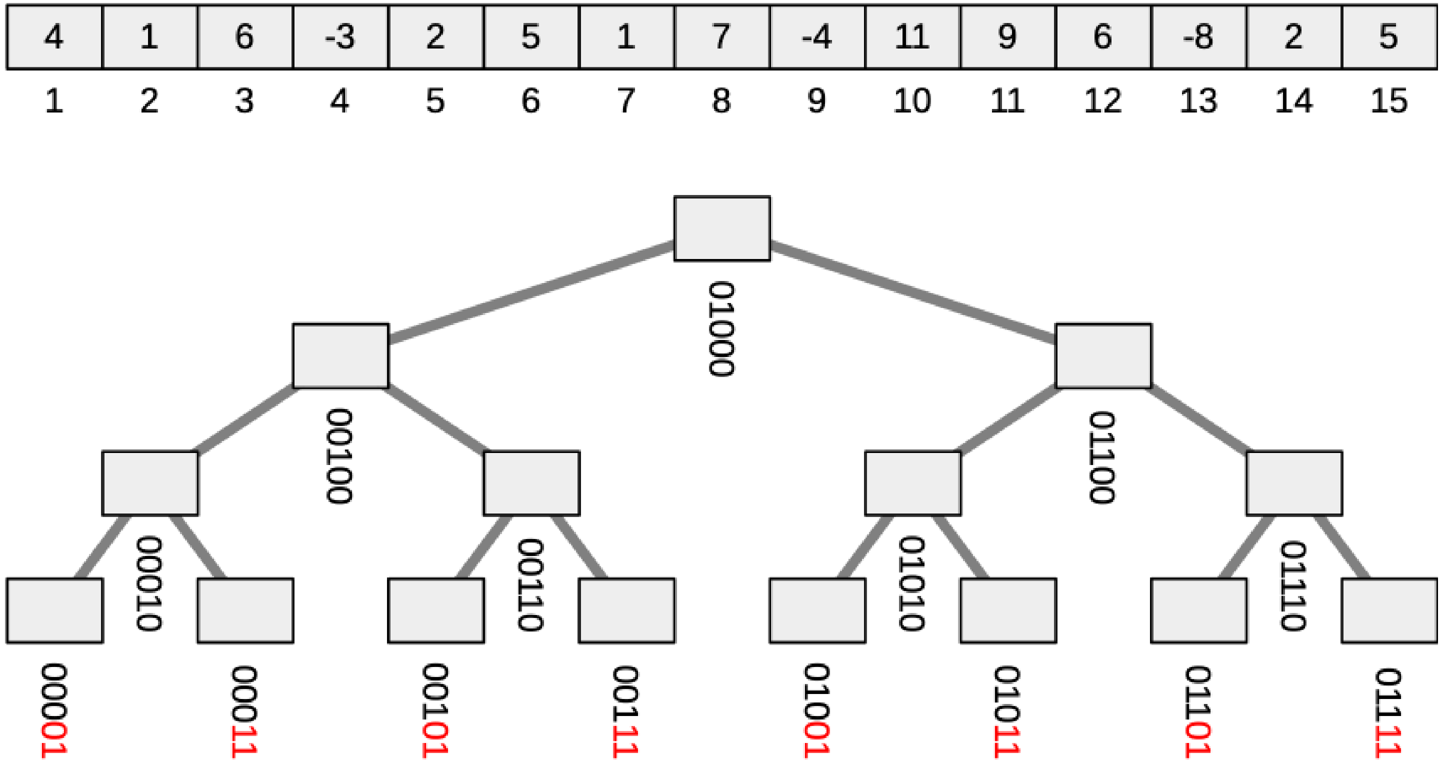
Fenwick Tree

- Values with a 1 in the next bit are their parents.
- All the way up the tree.



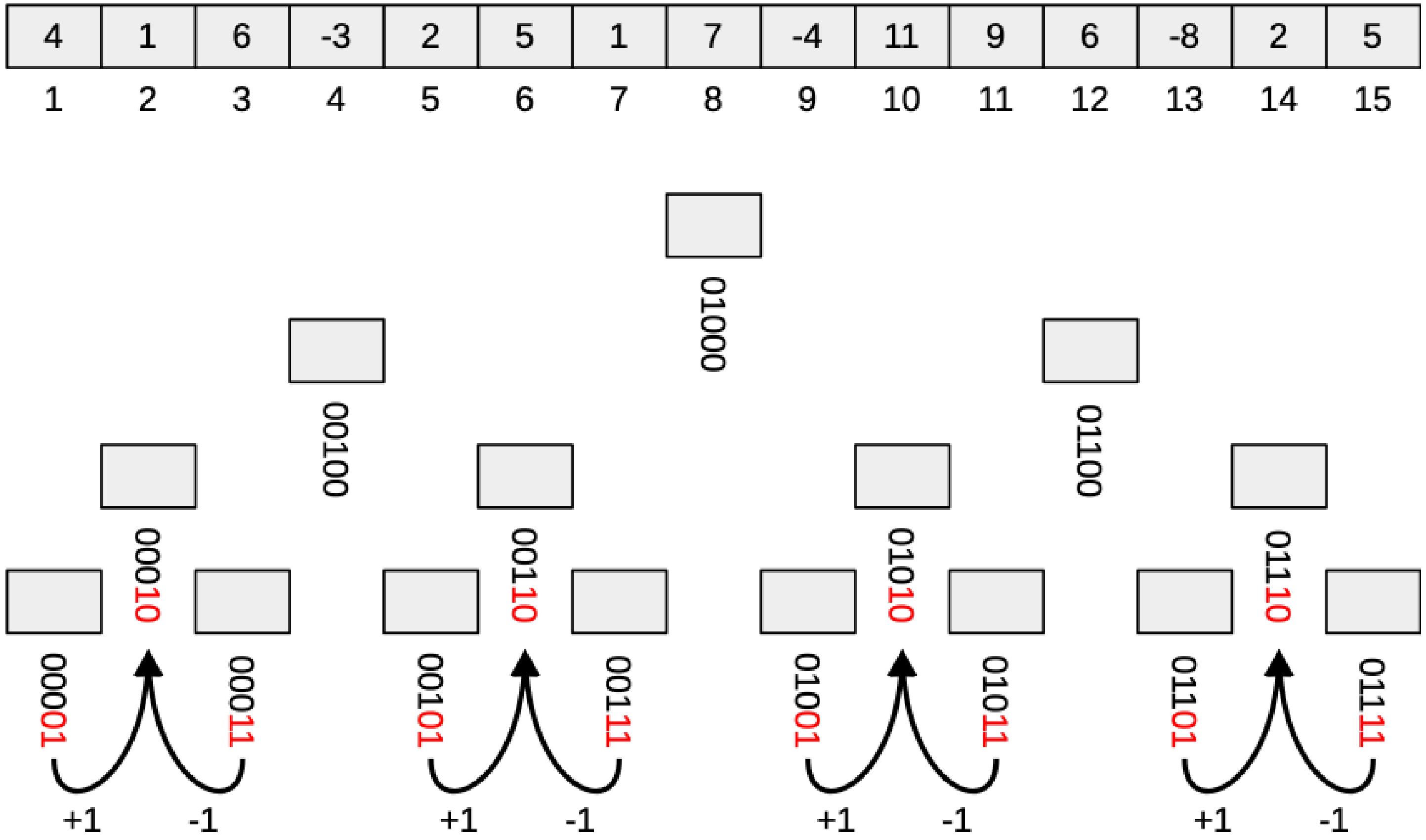
Fenwick Tree

- This is an implicit tree, like a binary heap.
- The low-order bits of the leaves alternate.
  - …01
  - …11
  - …01
  - …11



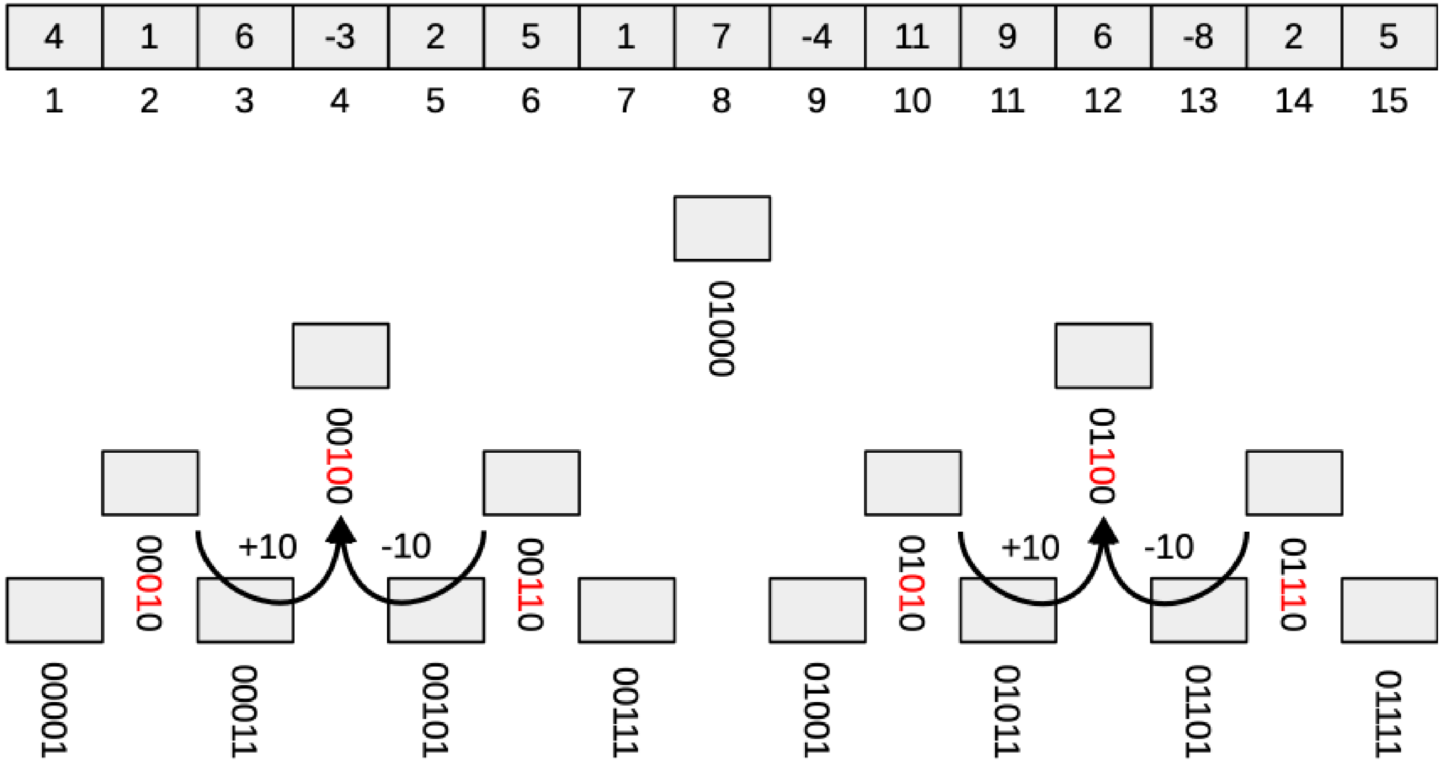
Fenwick Tree

- For a ….01 node, add 1 to get to the parent.
- For a …11 node, subtract 1 to get to the parent.



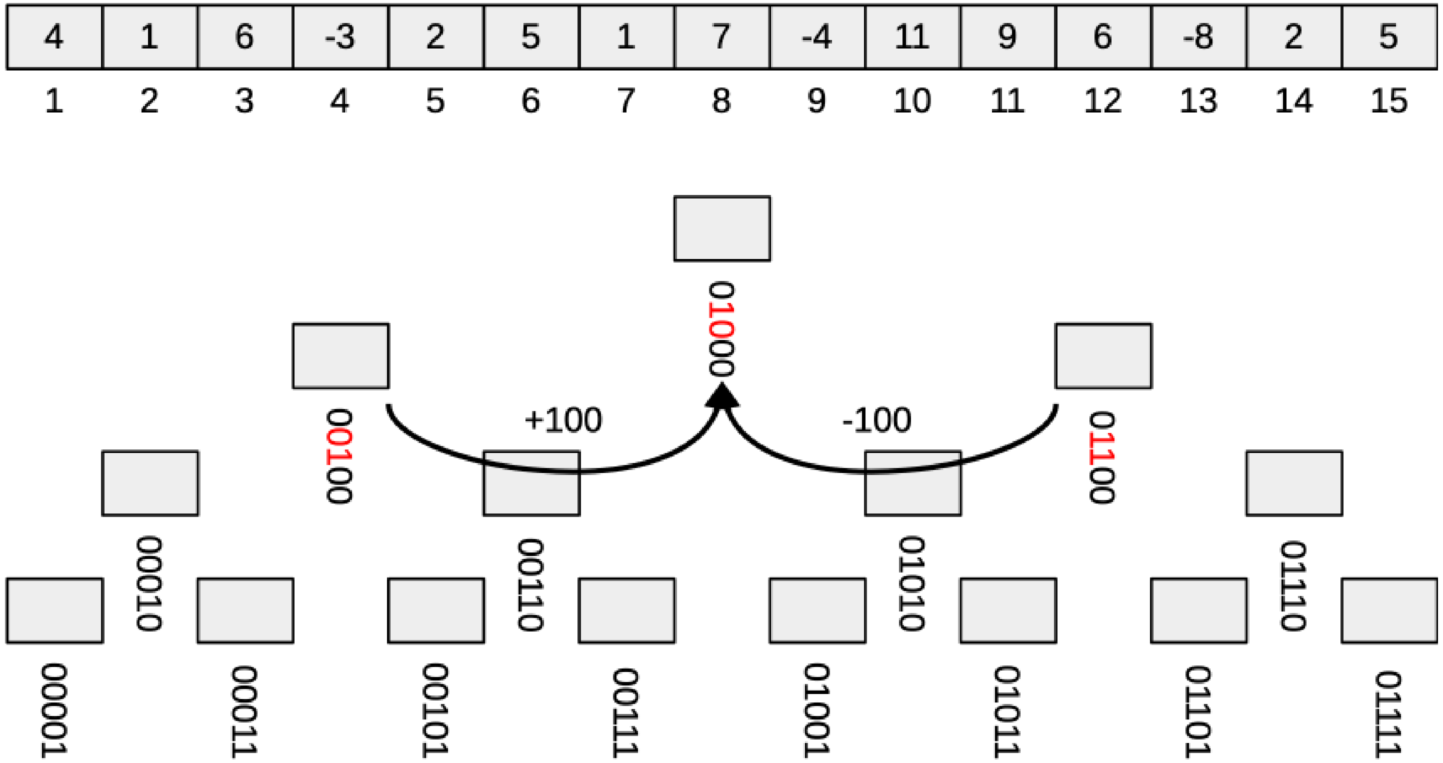
Fenwick Tree

- We have the same structure at the next level of the tree.
- But, its shifted one bit up.



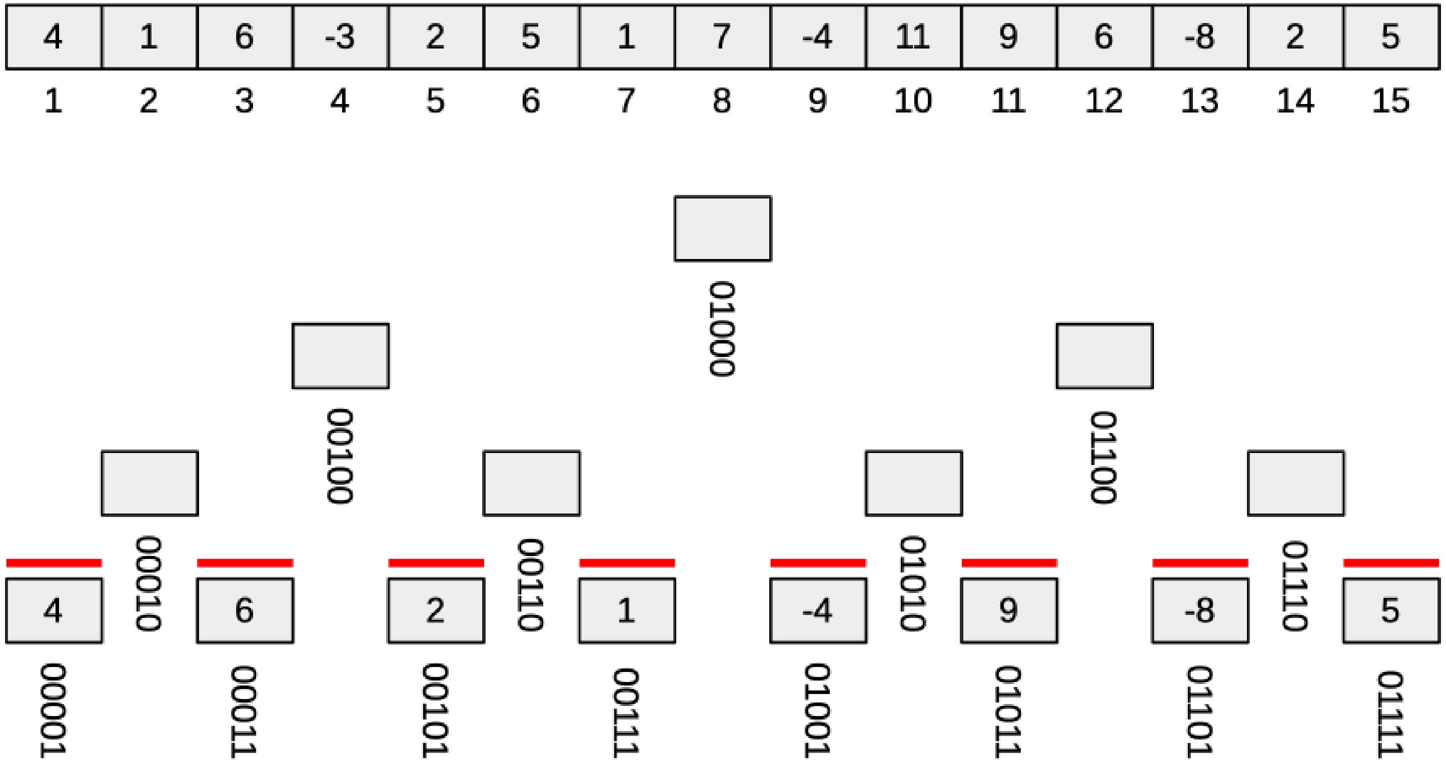
Fenwick Tree

- Same thing at every level of the tree.



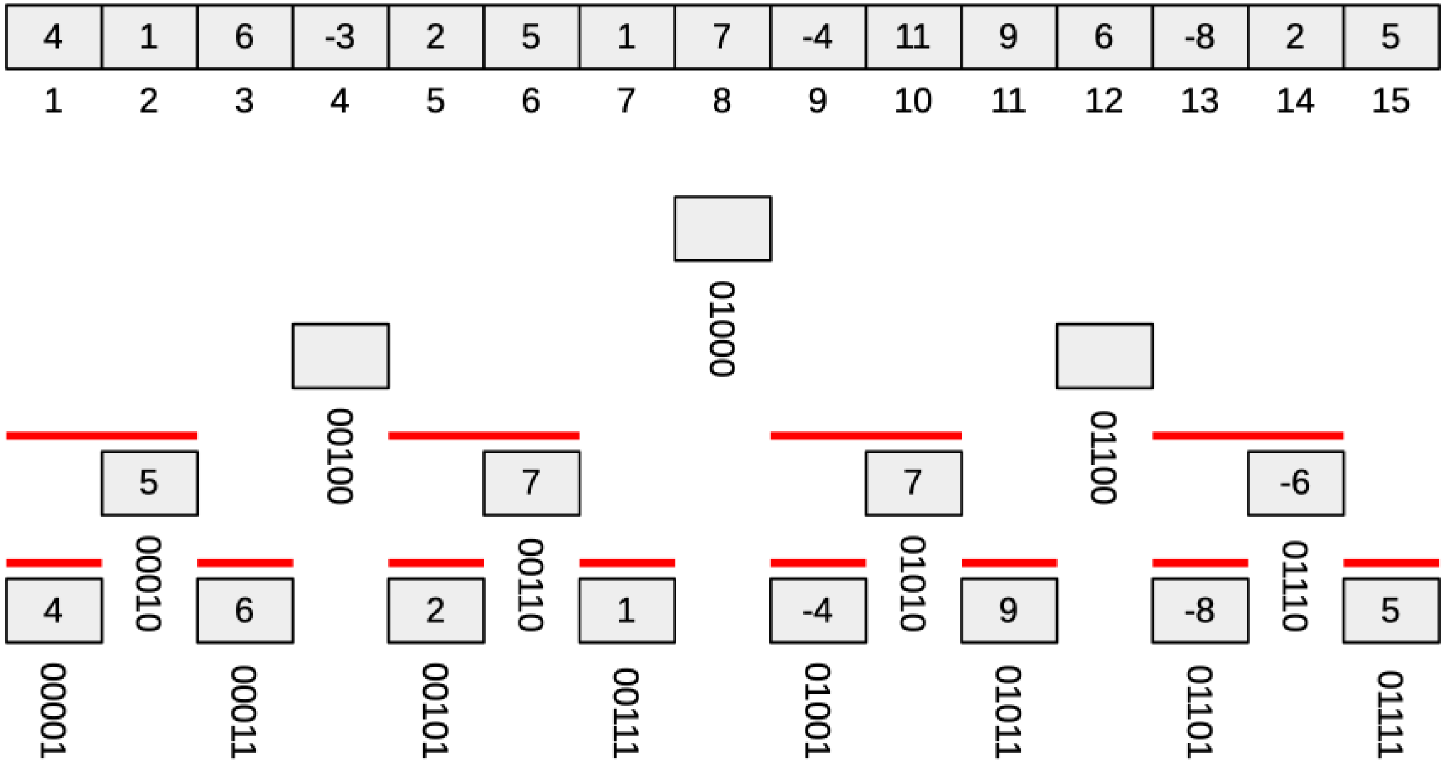
Storing Range Sums

- The leaves just store the value at their index.



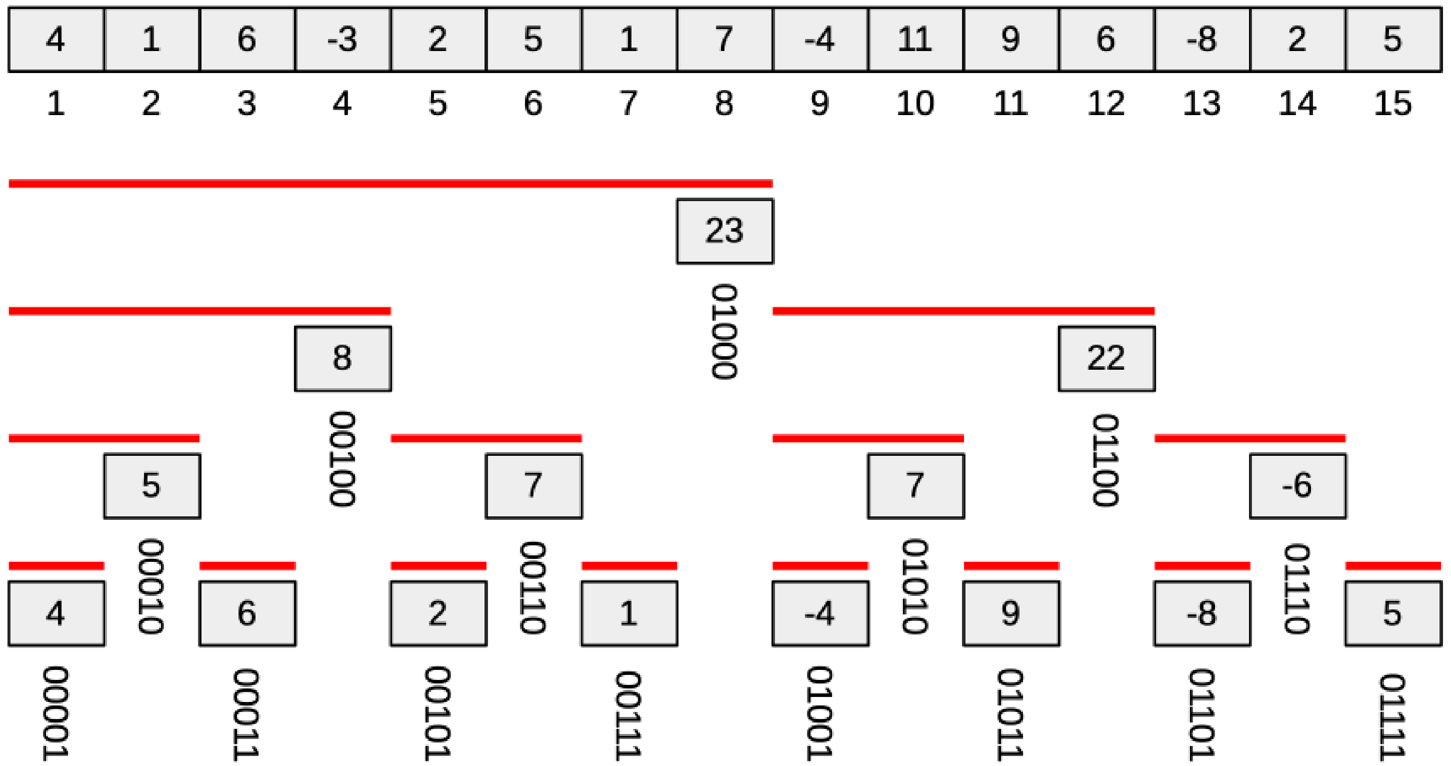
Storing Range Sums

- The next tree level stores two-element sums.



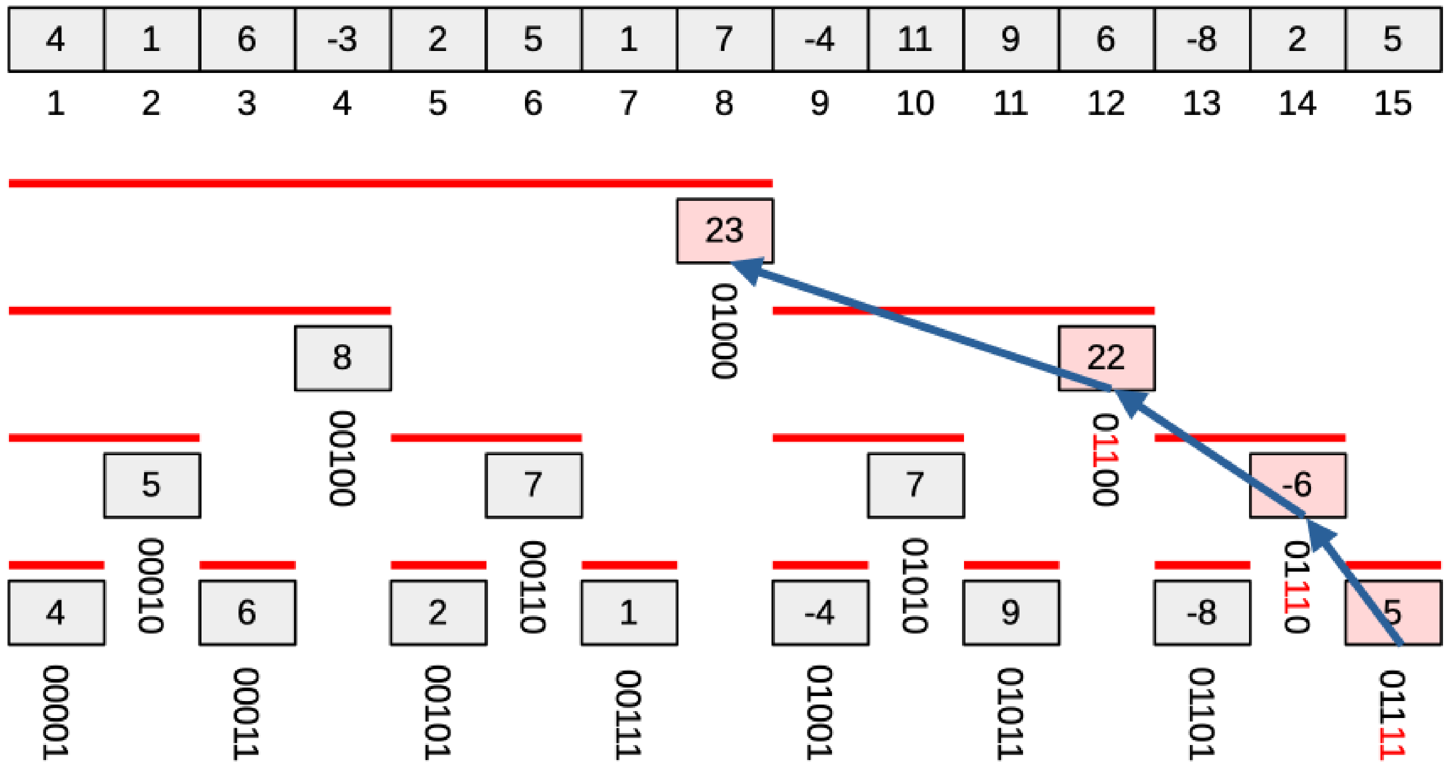
Storing Range Sums

- Four-element sums at the next level.
- Eight-element sums at the next.
- All the way up the tree.



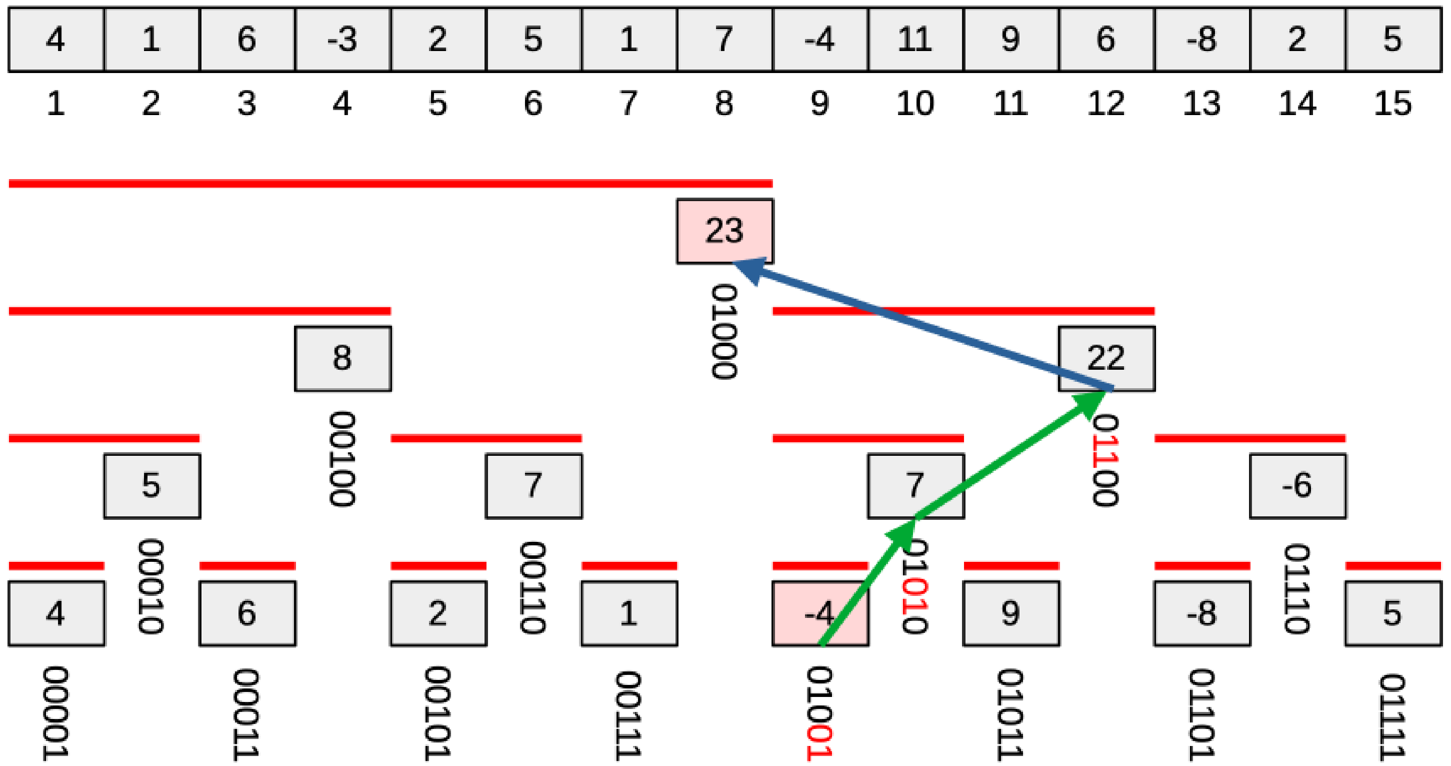
Computing a Prefix Sum

- We can compute a prefix sum by combining a few of these range sums
  - O($\log_2 n$)



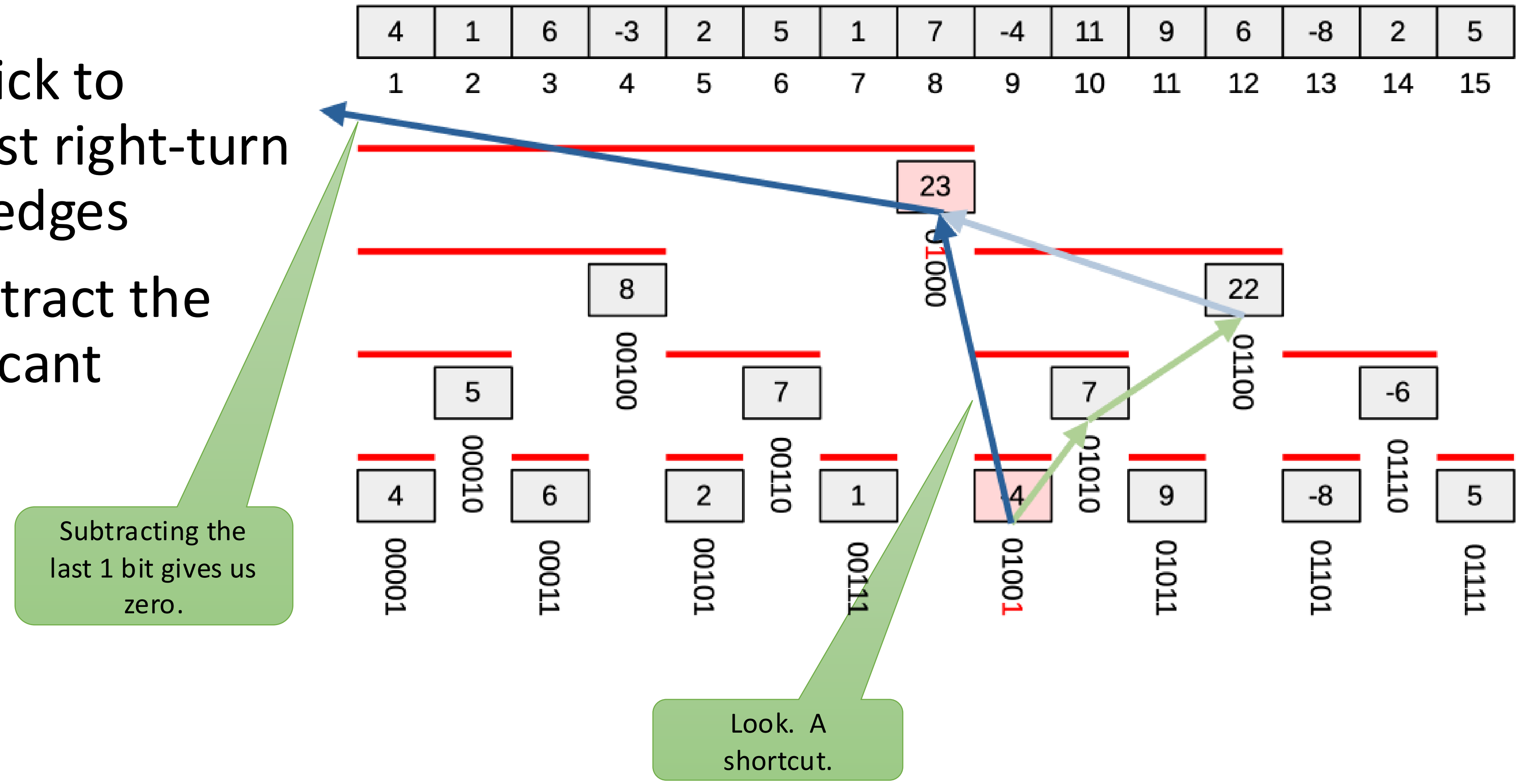
Computing a Prefix Sum

- We only want to add in sums that don't overlap.
- These are the ones we reach via a right child.



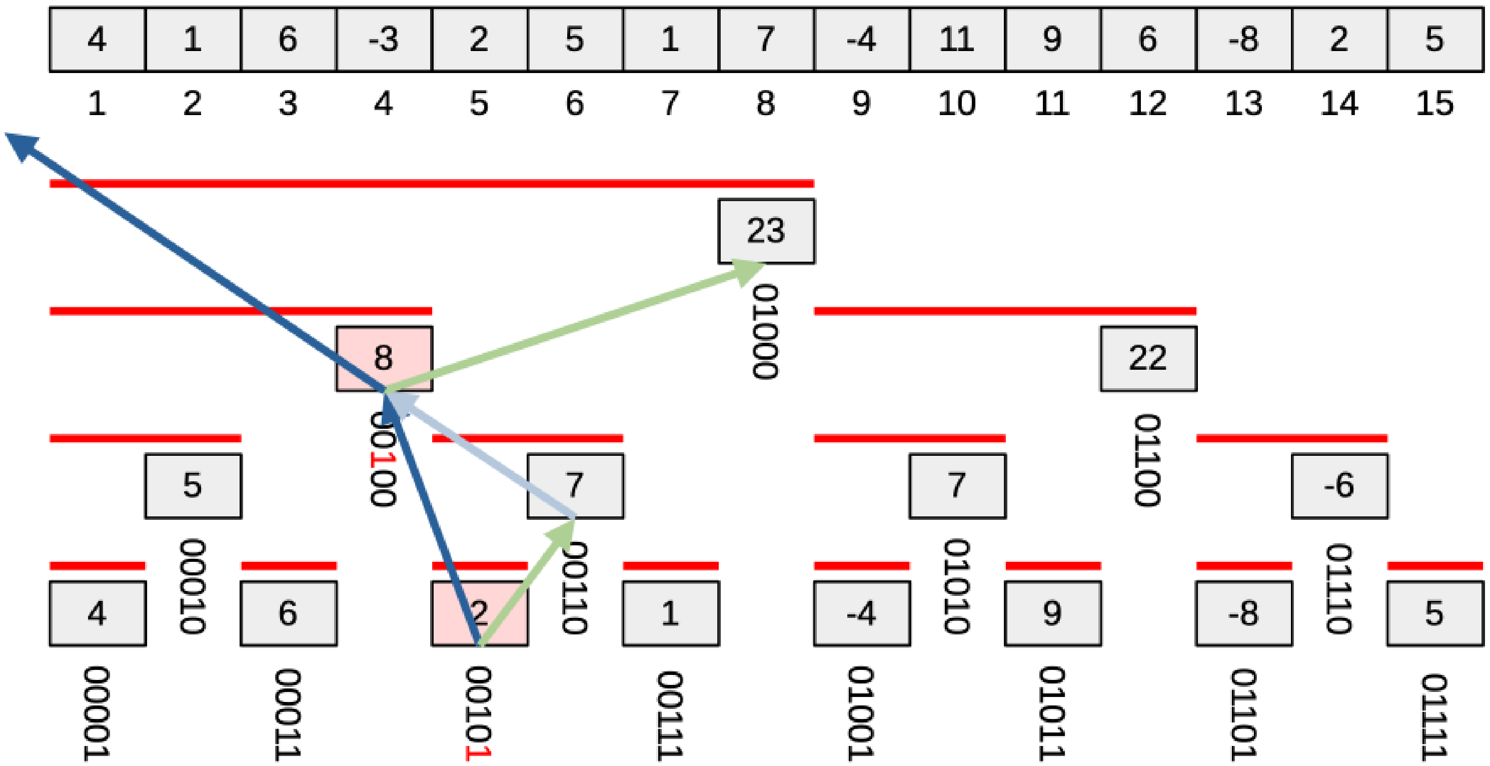
Computing a Prefix Sum … a little faster

- There's a trick to skipping past right-turn (left-child) edges
- We can subtract the least-significant 1 bit.



Computing a Prefix Sum … a little faster

- You can climb the tree, just visiting the nodes you need to add.
- When you've subtracted off all the 1 bits, you're done.
- That's why we skipped index zero.



Computing a Prefix Sum

Start at a given index in the tree.

```
int fenwickSum( int idx ) {
  int sum = 0;
  while ( idx > 0 ) {
    sum += tree[ idx ];
    idx -= LSB( idx );
  }
  return sum;
}
```

Computing a Prefix Sum

Start at a given index in the tree.

```
int fenwickSum( int idx ) {
  int sum = 0;
  while ( idx > 0 ) {
    sum += tree[ idx ];
    idx -= LSB( idx );
  }
  return sum;
}
```

Move up until you fall off the top.

Computing a Prefix Sum

```
int fenwickSum( int idx ) {
  int sum = 0;
  while ( idx > 0 ) {
    sum += tree[ idx ];
    idx -= LSB( idx );
  }
  return sum;
}
```

Start at a given index in the tree.

Move up until you fall off the top.

Add in the value at the current node.

Computing a Prefix Sum

```
int fenwickSum( int idx ) {
  int sum = 0;
  while ( idx > 0 ) {
    sum += tree[ idx ];
    idx -= LSB( idx );
  }
  return sum;
}
```

Start at a given index in the tree.

Move up until you fall off the top.

Add in the value at the current node.

Subtract off the least-significant 1 bit.

Computing a Prefix Sum

```
int fenwickSum( int idx ) {
  int sum = 0;
  while ( idx > 0 ) {
    sum += tree[ idx ];
    idx -= LSB( idx );
  }
  return sum;
}
```

Start at a given index in the tree.

Move up until you fall off the top.

Add in the value at the current node.

Subtract off the least-significant 1 bit.

How do we do this?

Finding the Least-significant 1 Bit

- We can get help from the processor's ALU

```
int LSB( int idx ) {
  return idx & -idx;
}
```

Finding the Least-significant 1 Bit

- We can get help from the processor's ALU

```
int LSB( int idx ) {
  return idx & -idx;
}
```

```
idx = 01001100100000
```

Finding the Least-significant 1 Bit

- We can get help from the processor's ALU

```
int LSB( int idx ) {
  return idx & -idx;
}
```

```
 idx = 01001100100000
~idx = 10110011011111
```

Finding the Least-significant 1 Bit

- We can get help from the processor's ALU

```
int LSB( int idx ) {
  return idx & -idx;
}
```

```
 idx = 01001100100000
-idx = 10110011100000
```

Finding the Least-significant 1 Bit

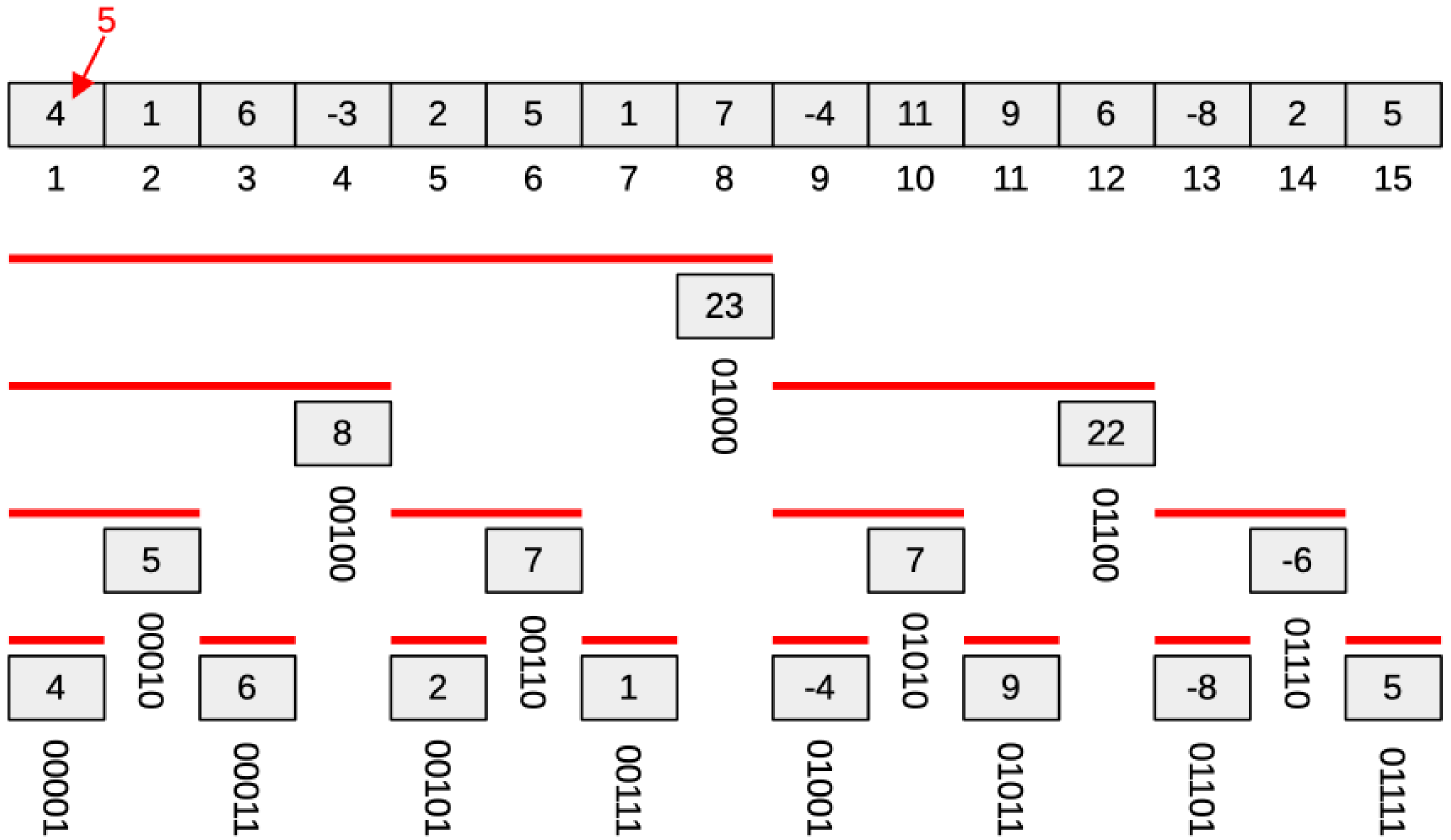
- We can get help from the processor's ALU

```
int LSB( int idx ) {
  return idx & -idx;
}
```

```
  01001100100000
& 10110011100000
  --------------
  00000000100000
```

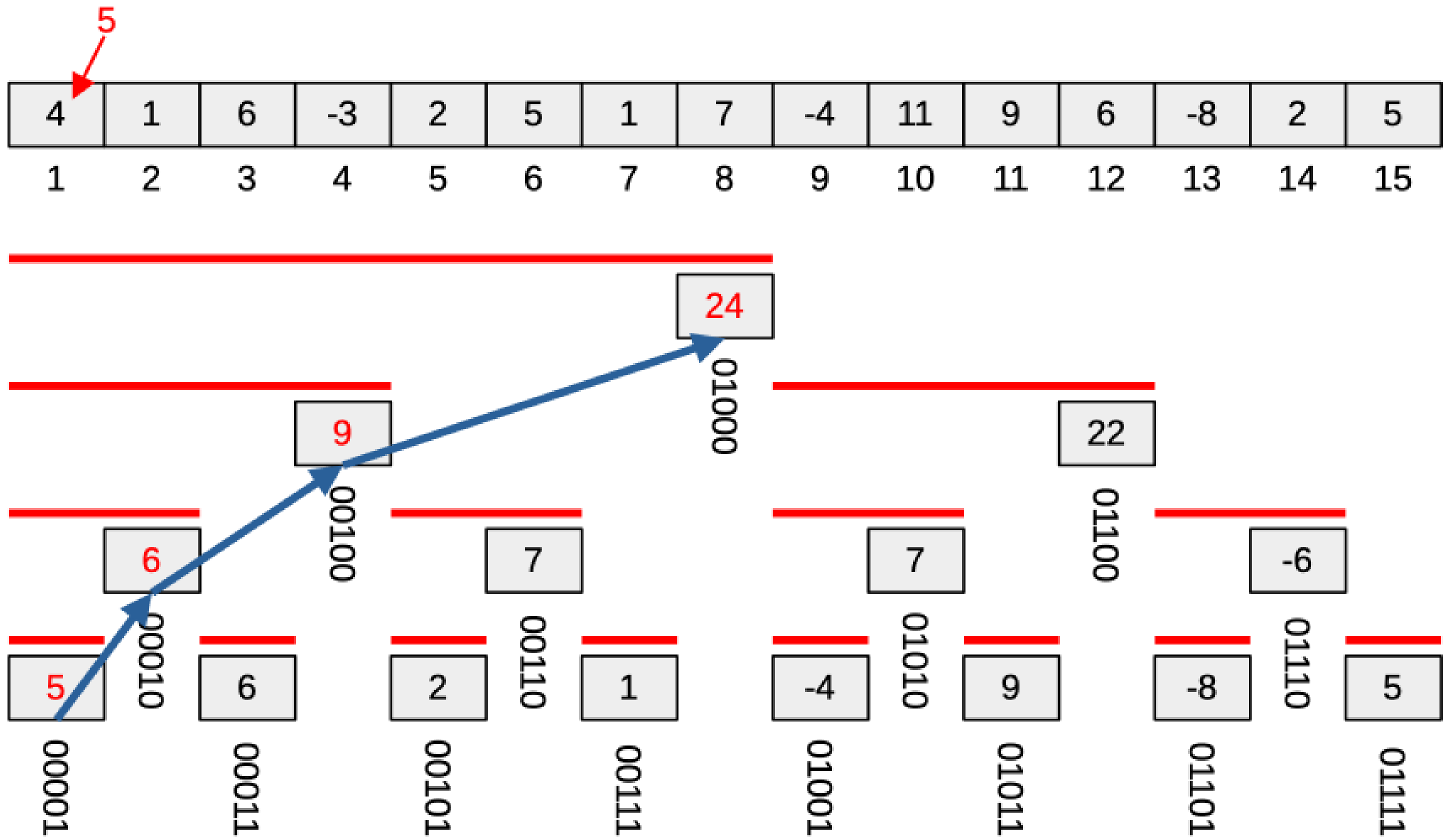
Changing the Sequence

- If we change the sequence, we will have to update multiple range sums



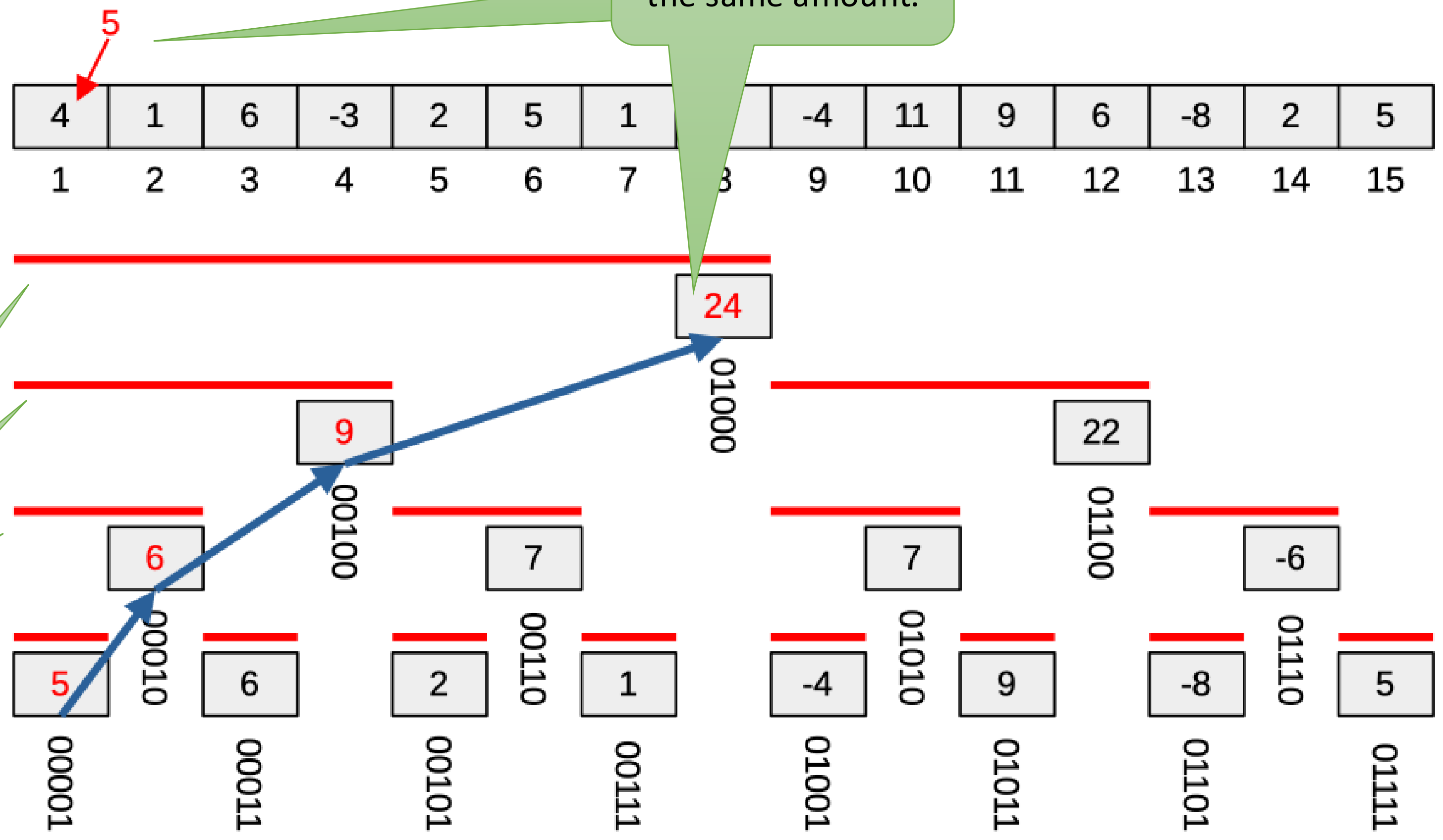
Changing the Sequence

- If we change the sequence, we will have to update multiple range sums
- … but not too many.



Changing the Sequence

- If we change the sequence, we will have to update multiple range sums
- … but not too many

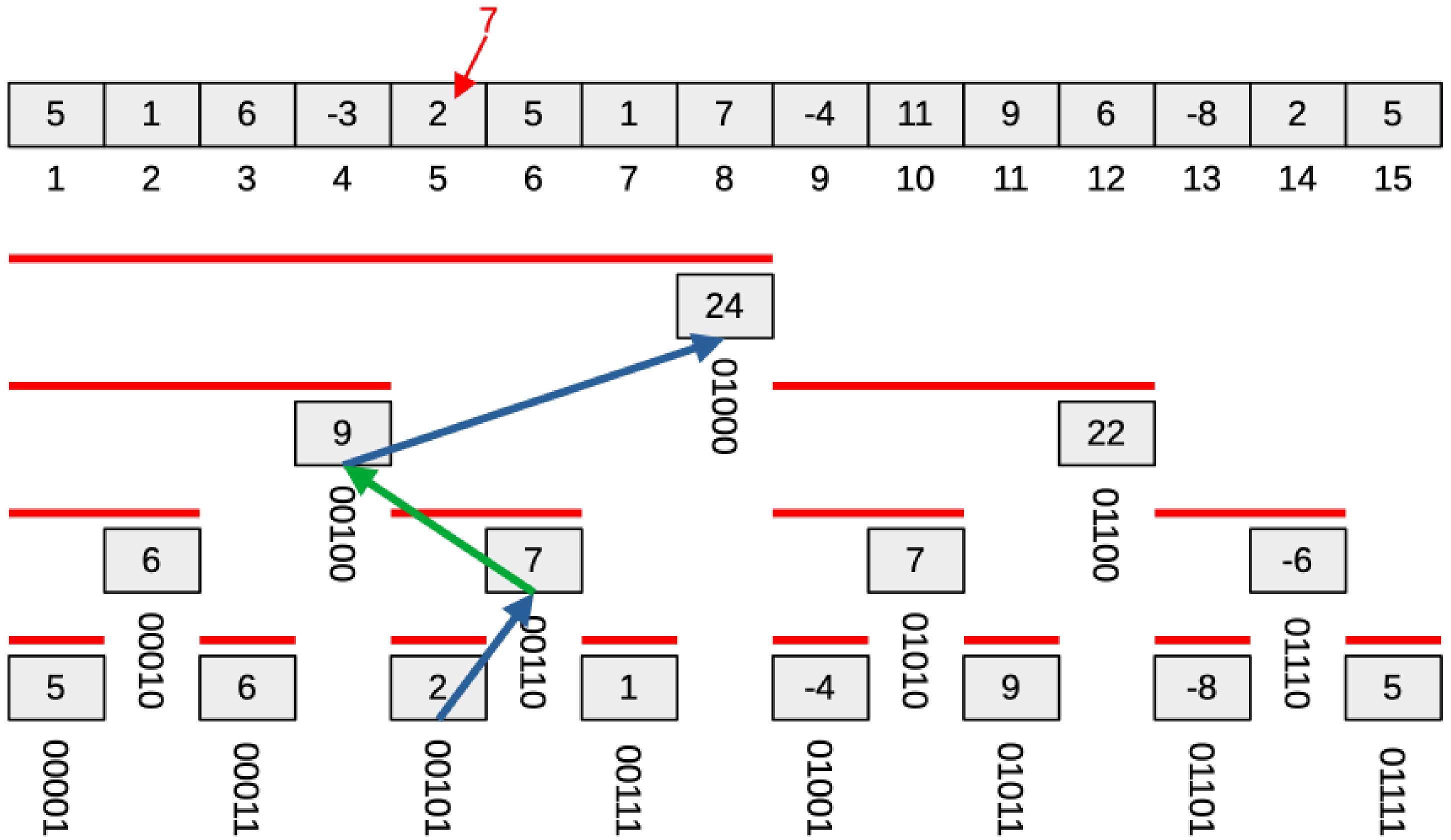


All sums change by the same amount.

We want all range sums that overlap with the change.

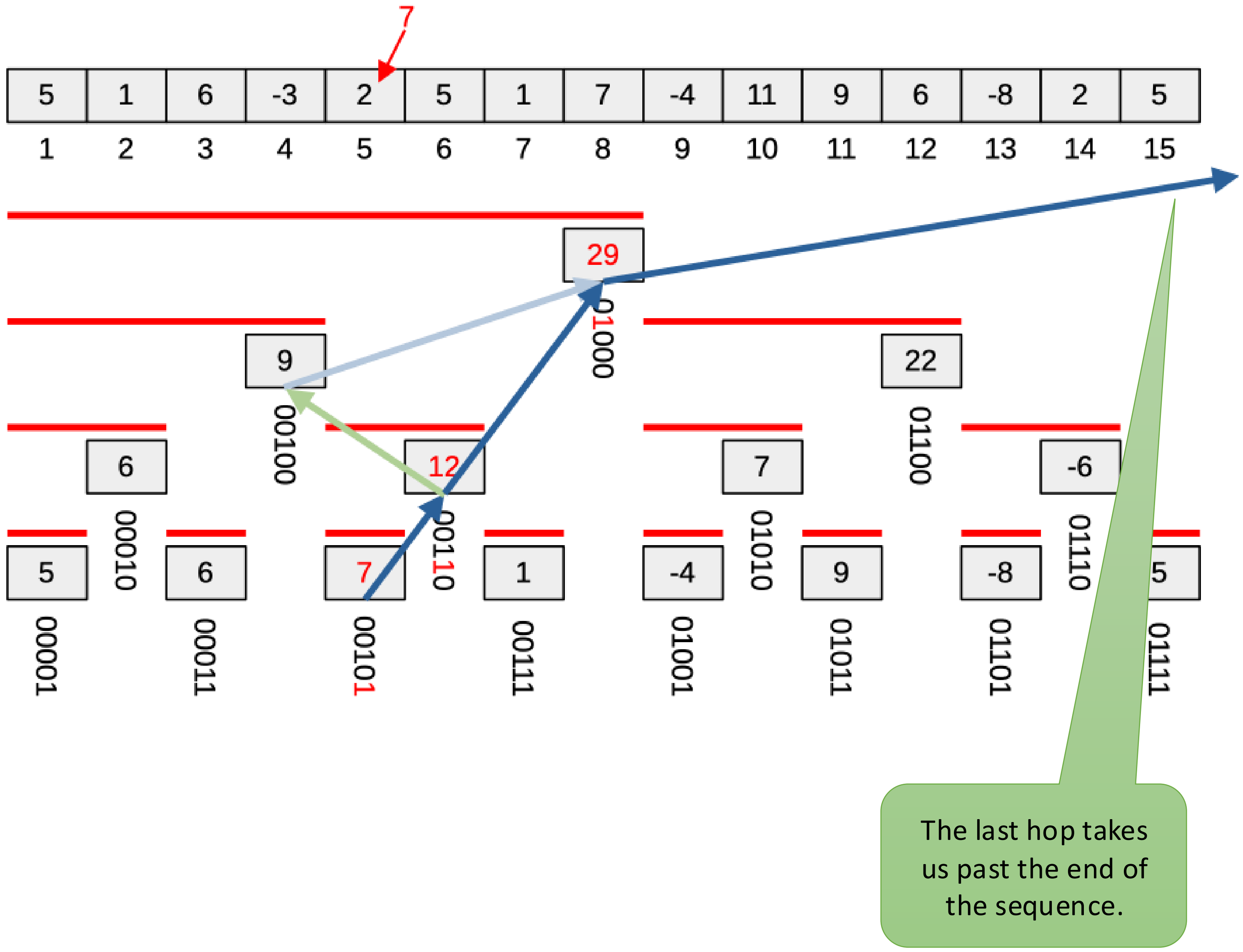
Changing the Sequence

- Here, we need to update any sum we reach via a left child.
- Parent of a right child won't overlap the updated value



Changing the Sequence ... a little bit faster

- Here also, there's a shortcut.
- If we add the least-significant 1 bit, we skip all the left turns.



Updating Range Sums

Starting index and change in value.

```
void fenwickAdd( int idx, int d ) {
  while ( idx < tree.length ) {
    tree[ idx ] += d;
    idx += LSB( idx );
  }
}
```

Updating Range Sums

Starting index and change in value.

Move up until you fall off the top (to the right this time)

```
void fenwickAdd( int idx, int d ) {
  while ( idx < tree.length ) {
    tree[ idx ] += d;
    idx += LSB( idx );
  }
}
```

Updating Range Sums

Starting index and change in value.

```
void fenwickAdd( int idx, int d ) {
  while ( idx < tree.length ) {
    tree[ idx ] += d;
    idx += LSB( idx );
  }
}
```

Move up until you fall off the top (to the right this time)

Modify the sum at this node.

Updating Range Sums

Starting index and change in value.

```
void fenwickAdd( int idx, int d ) {
  while ( idx < tree.length ) {
    tree[ idx ] += d;
    idx += LSB( idx );
  }
}
```

Move up until you fall off the top (to the right this time)

Modify the sum at this node.

Move to the next ancestor to the right.

Initializing the Tree

- You can initialize the tree with any sequence
  - In linear time
- Or, you can start with a sequence of zeros.

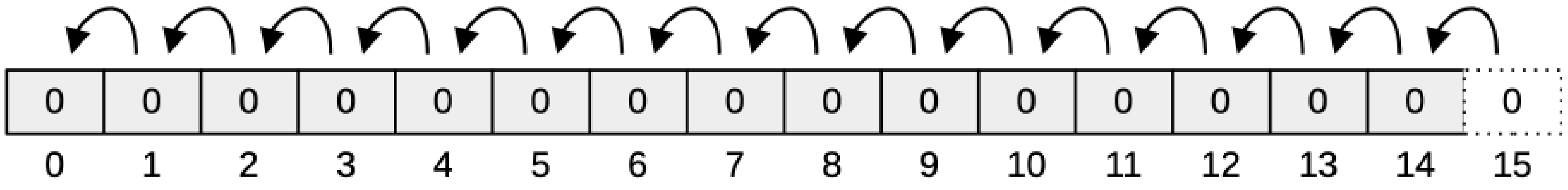
One larger than the number of elements you need.

	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Then use fenwickAdd() to set all the values to what you want.
  - O(n log n) time.

Variant Tree Representations

- You could make use of elements 0 .. n -1
  - By using an offset of one when you do the tree traversal.



- You could omit element i from fenwickSum(i)