works on any binary architecture, provided only that $s < 2^w$. For example, assuming that $1 \le s < 2^4 = 16$, *size*(4) and *size*(10) are calculated to be 4 and 2 respectively:

	$s = 4$	$s = 10$
s in binary:	0100	1010
$16 - s$ in binary:	1100	0110
result after AND:	0100	0010
result in decimal:	4	2

The cost of calculating *size*() is thus $O(1)$, and the overall time taken to process symbol s is $O(\log n)$. Compared to the sorted array method of Witten et al., this mechanism represents a rare instance of algorithmic development in which both space and time are saved. Using it, the overall cost of adaptively maintaining statistics for message M containing m symbols is $O(m \log n)$ time.

One further function is required in the decoder, illustrated in Algorithm 6.5. The *target* value returned by *arithmetic_decode_target*() (described in Algorithm 5.4 on page 104) must be located in the array *fen_prob*. This is accomplished by a binary search variant based around powers of two. The first location inspected is the largest power of two less than or equal to n, the current alphabet size. That is, the search starts at position $2^{\lfloor \log_2 n \rfloor}$. If the value stored at this position is greater than *target*, then the desired symbol cannot have a greater index, and the search focuses on the first section of the array. On the other hand, if the *target* is larger than this middle value the search can move right, looking for a diminished *target*. Once the desired symbol number s has been determined, function *fenwick_get_lbound*() is used to determine the bound l for the arithmetic decoding function, and *fenwick_get_and_increment_count*() is used to determine c and to increment the frequency of symbol s. Both of these latter two functions are shared with the encoder.

The attentive reader might still, however, be disappointed, as $O(m \log n)$ time can still be superlinear in the number of bits emitted. Consider, for example, the probabilities

$$P = [1/2, 1/4, \ldots, 1/2^i, \ldots, 1/2^{n-1}, 1/2^{n-1}] .$$

Then a message of length m has an expected coded length of approximately $2m$ bits irrespective of the value of n, yet the time taken to generate that bitstream using a Fenwick tree is $\Theta(m \log n)$. In particular, the reader may recall that adaptive Huffman coding does take time linear in the number of bits produced. Can similar linear-time behavior be achieved for adaptive arithmetic coding?

Pleasingly, the answer is “yes” – it is possible to modify the Fenwick tree data structure and obtain asymptotic linearity [Moffat, 1999]. Consider again