Week 7 Solutions

CSE 332

1) Parallel Prefix Sum

Goal: Output array needs to store sums of everything up to a certain index. Meaning:
Output[i] = input[i]+input[i-1]+input[i-2]+…+input[0]

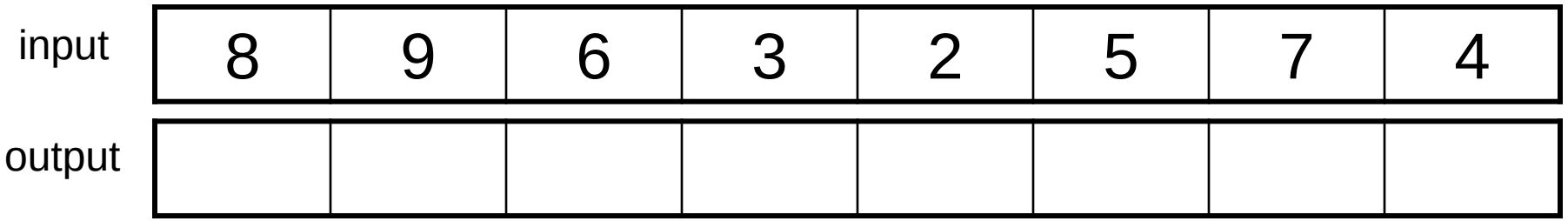


Figure out what information you need

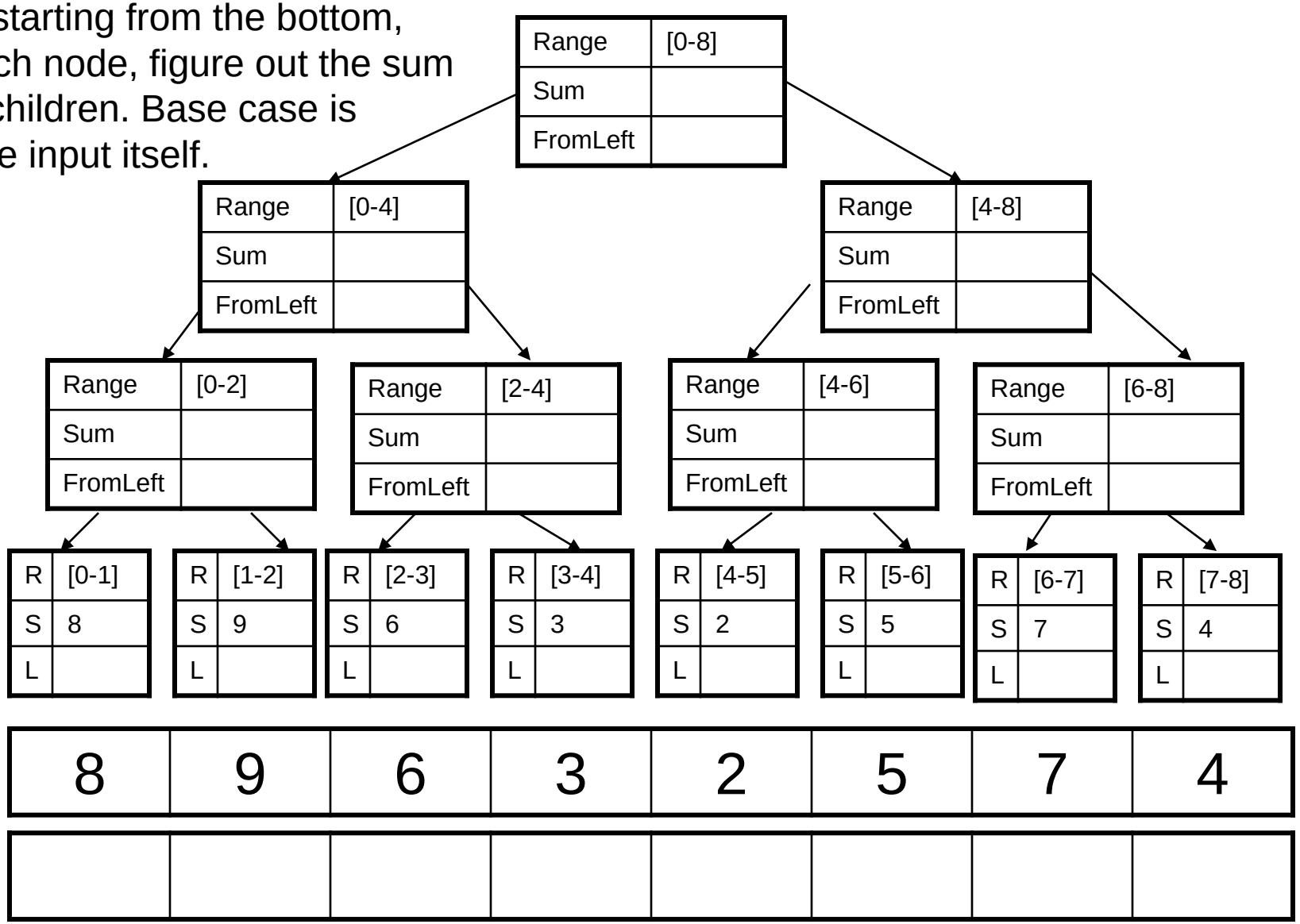
Range	[0-8]
Sum	
FromLeft	

Start off at root with the entire range of the problem (low=0, high=8). We need to find the Sum and the FromLeft value of the root, but we will do this in two passes. First pass, go down and split up the problem until we get to the cutoff of one item (high-low=1)

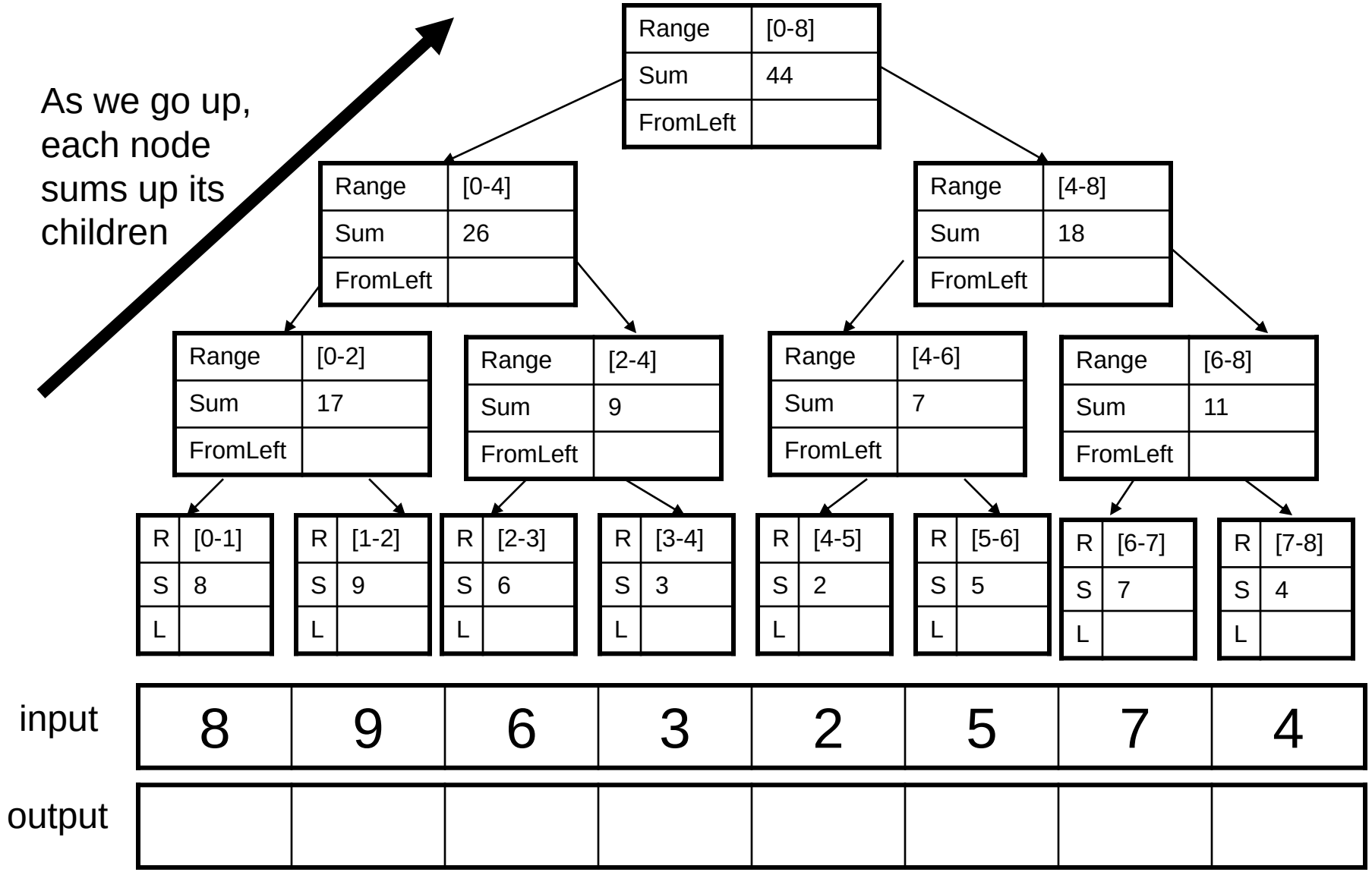
input	8	9	6	3	2	5	7	4
output								

Divide problem into parallel pieces

Now, starting from the bottom, for each node, figure out the sum of its children. Base case is just the input itself.

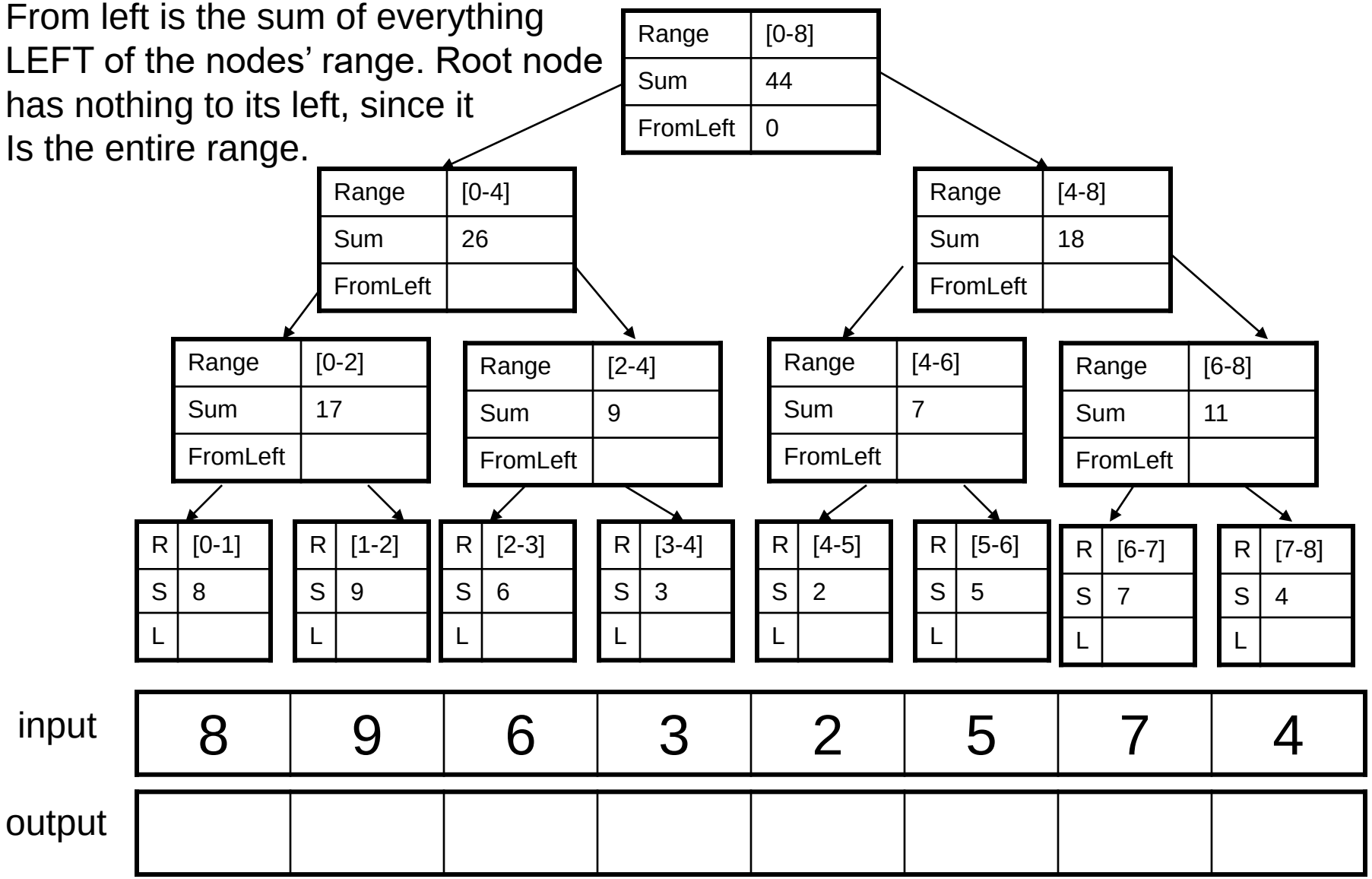


1st pass, find sums going up.



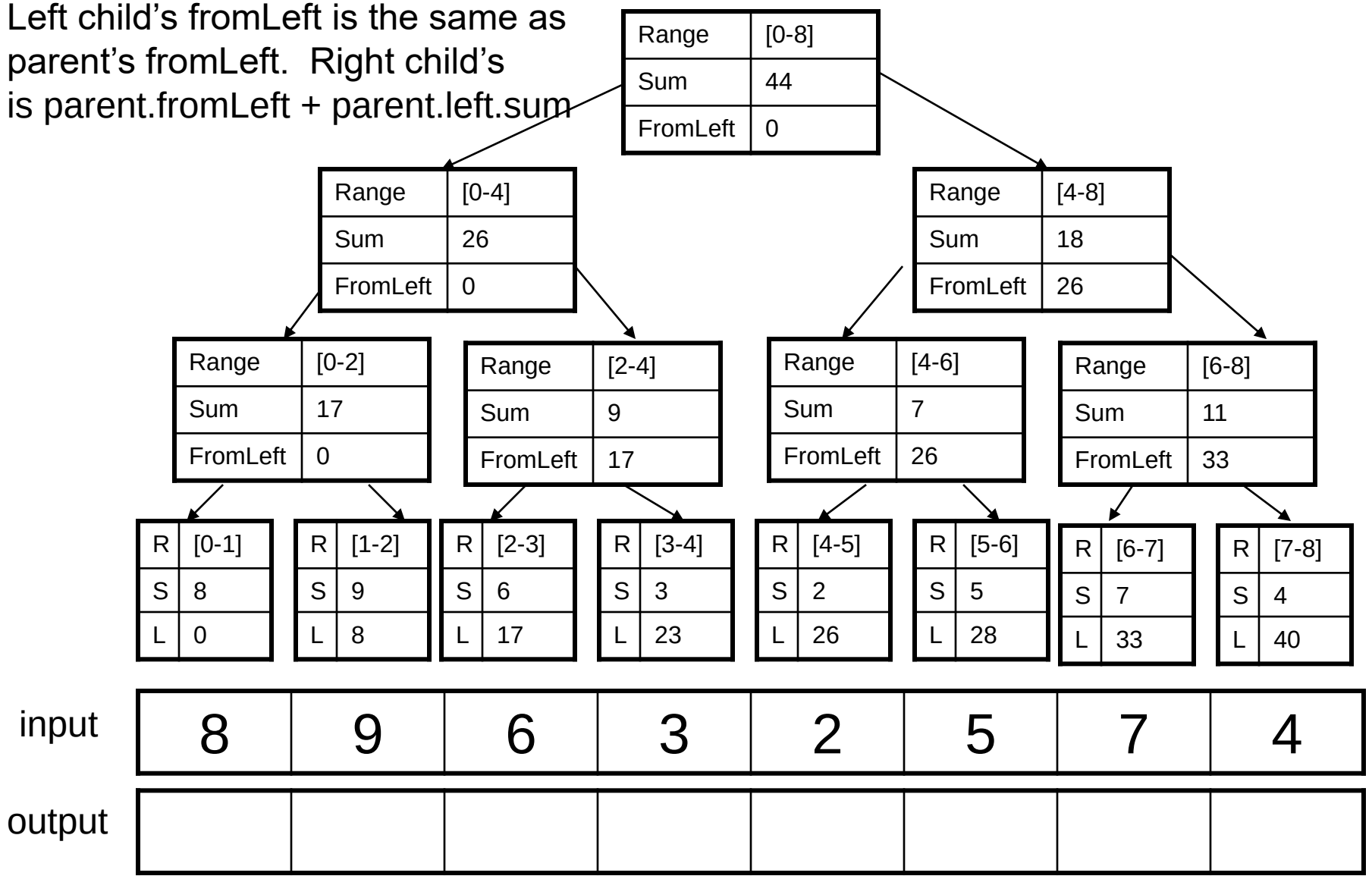
2nd pass, fill out FromLeft going down

From left is the sum of everything LEFT of the nodes' range. Root node has nothing to its left, since it Is the entire range.



2nd pass, fill out FromLeft going down

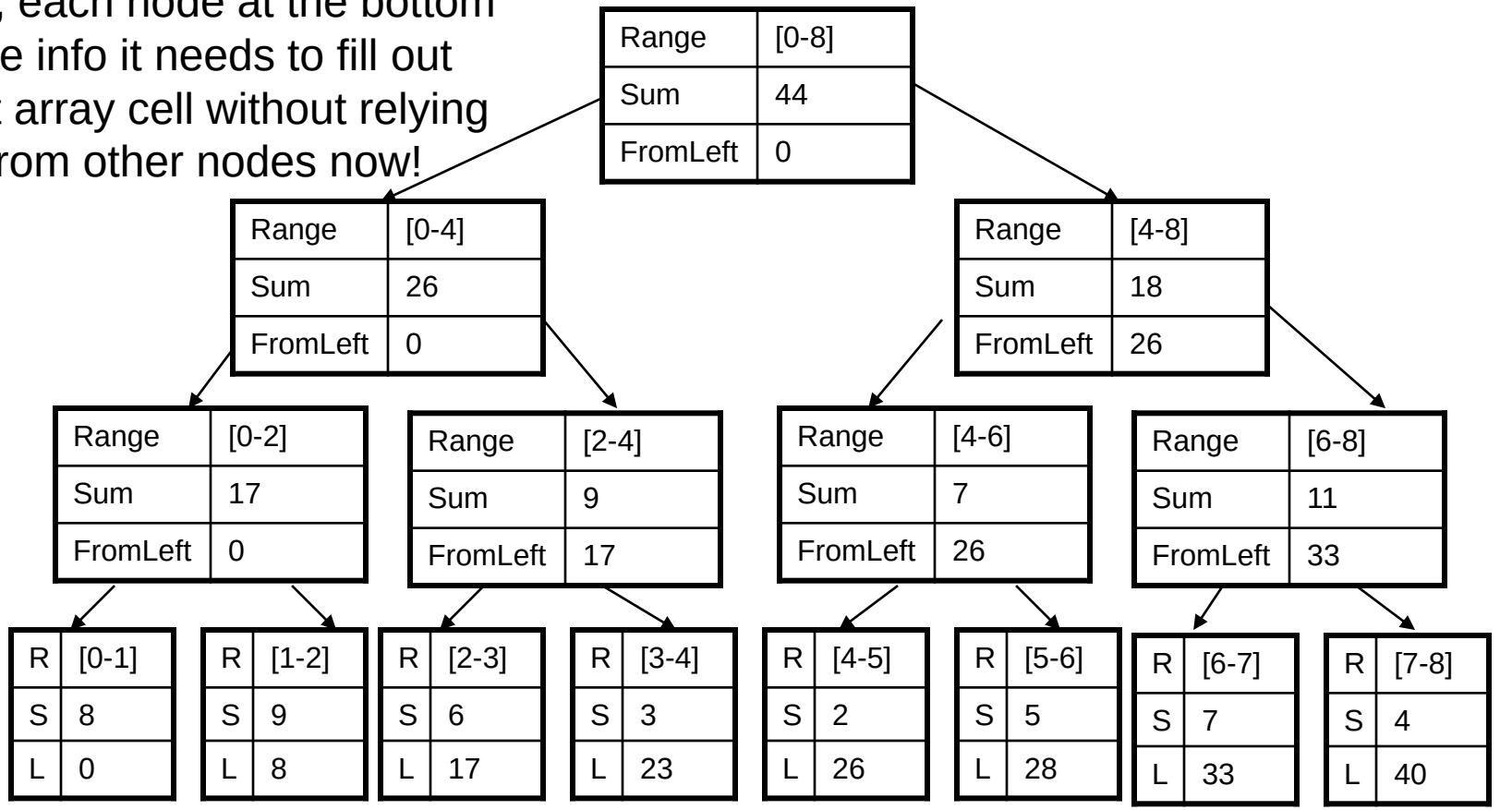
Left child's fromLeft is the same as parent's fromLeft. Right child's is parent.fromLeft + parent.left.sum



Finally, fill out output array

output[this.low] = this.sum + this.fromLeft

Basically, each node at the bottom has all the info it needs to fill out its output array cell without relying on data from other nodes now!

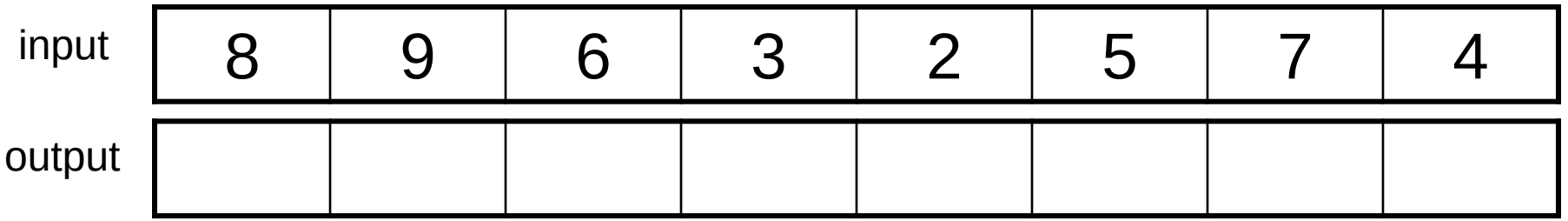


input	8	9	6	3	2	5	7	4

output	8	17	23	26	28	33	40	44

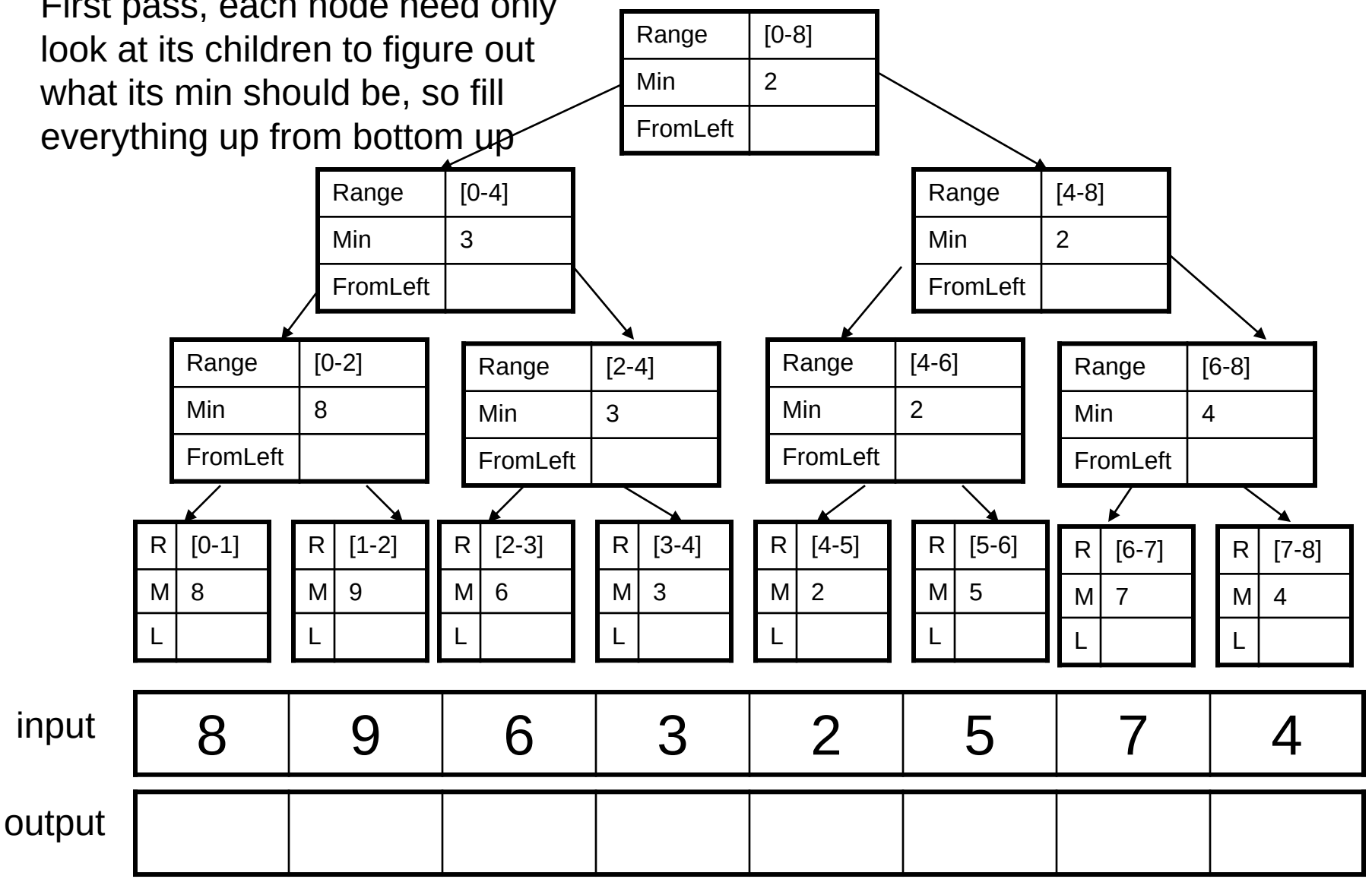
2) Parallel Prefix FindMin

Output an array with the minimum value of all cells to its left.
So, output[i] = min(input[0],input[1],input[2],….input[i-1],input[i])



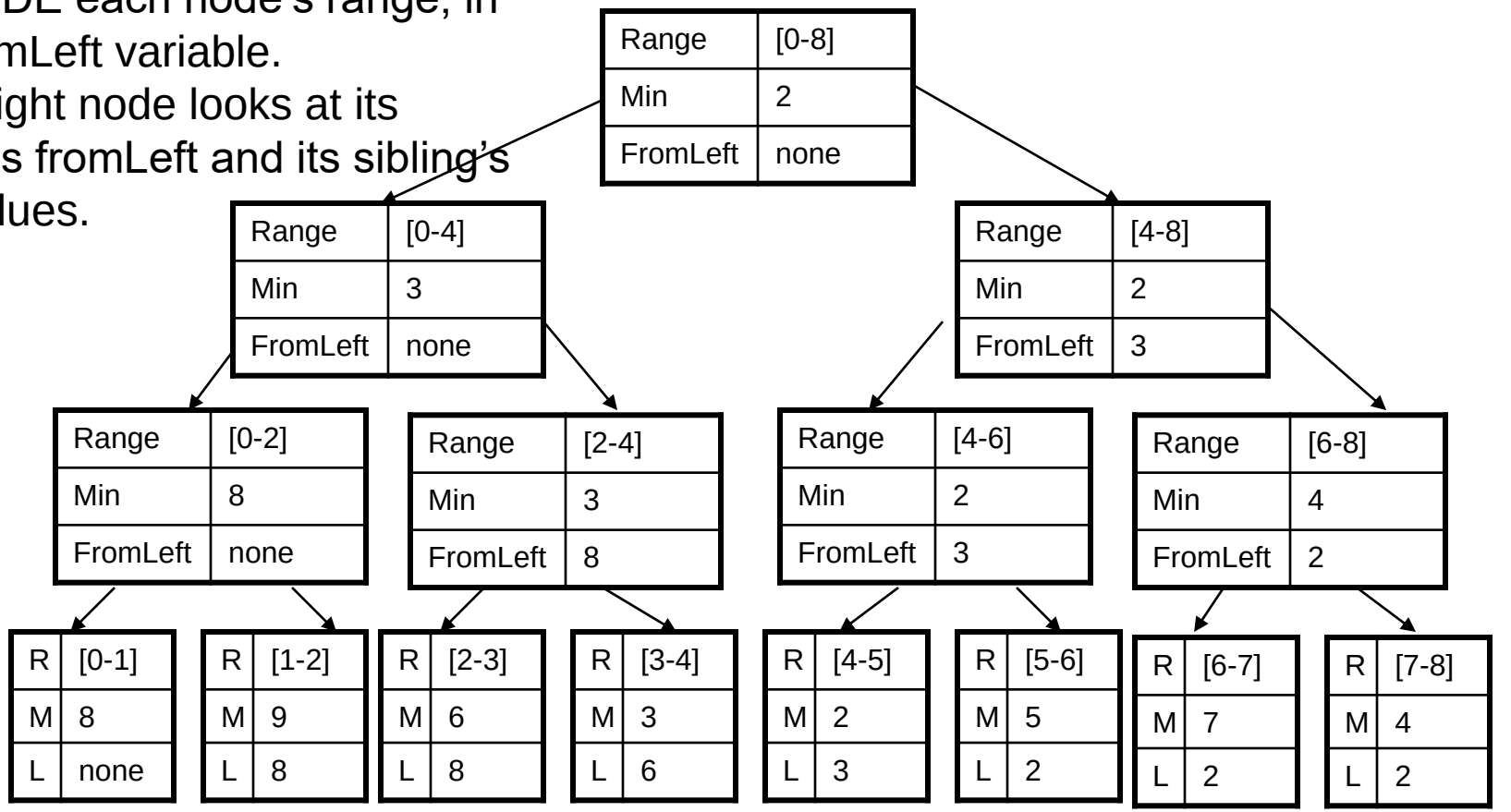
Same as before, except this time, we want to store the node's range, the min of its children, and the min of everything to its left.

First pass, each node need only look at its children to figure out what its min should be, so fill everything up from bottom up



Second pass, we need to fill everything starting from the root going down.

Fill out the min value from OUTSIDE each node's range, in the fromLeft variable.
Each right node looks at its parent's fromLeft and its sibling's min values.



input	8	9	6	3	2	5	7	4
output	8	8	6	3	2	2	2	2

2. Work it Out [the Span]

(a) Define work and span

Solution:
Work - how long the running time of a program would be with just one processor
Span - the running time with an infinite number of processors

(b) How do we calculate work and span?

Solution:
Work - sum all the work done by each processor
Span - calculate the longest dependence chain (the longest 'branch' in the parallel 'tree')

(c) Does adding more processors affect the work or span?

Solution:
Neither - both work and span are defined by a fixed number of processors (1 for work and infinity for span) so adding more processors won't affect them