Week 7 Solutions

CSE 332 Autumn 2016

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]

input	8	9	6	3	2	5	7	4
output								

Figure out what information you need

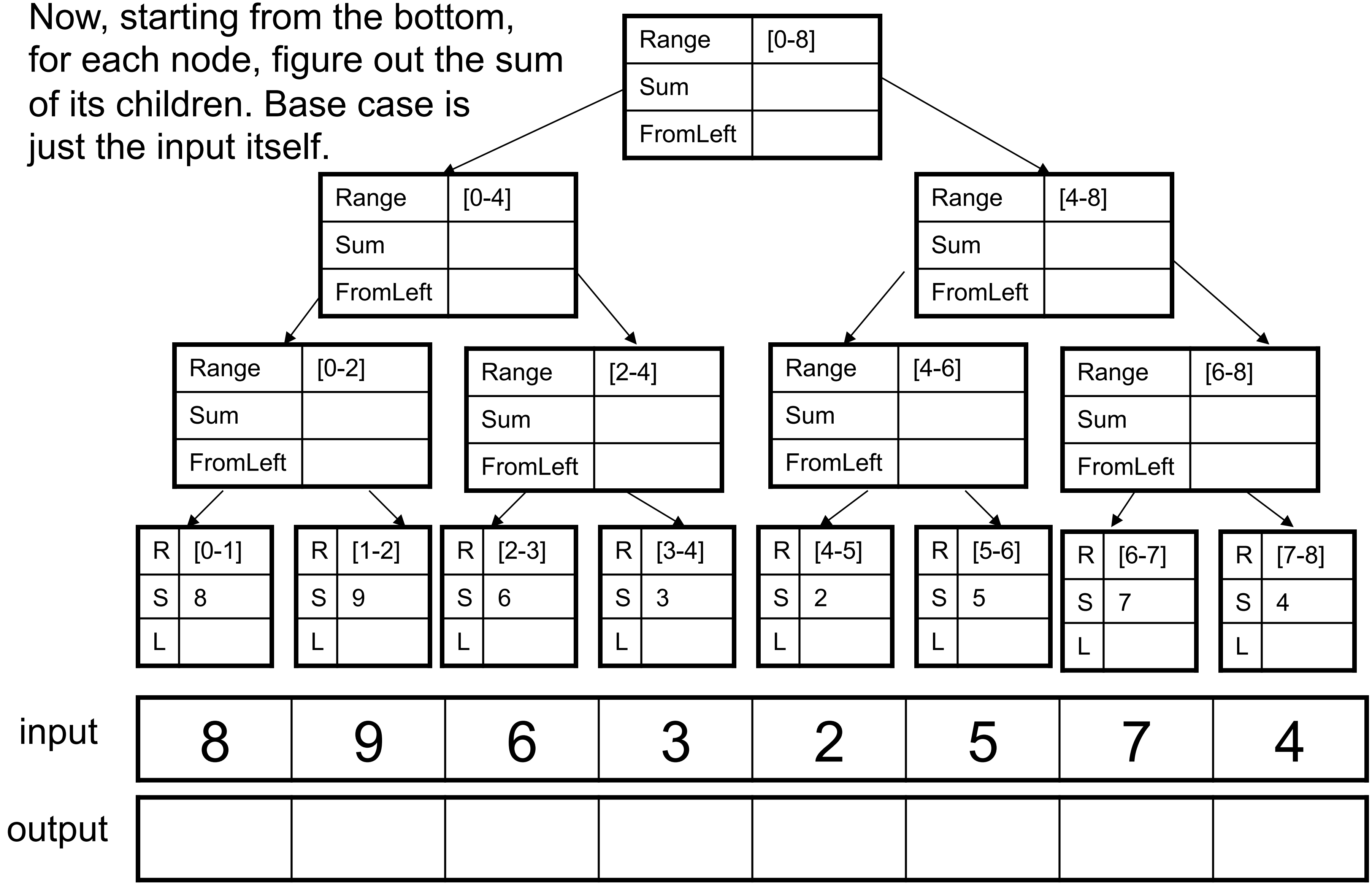
Range	[0-8]
Sum	
FromLeft	

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)

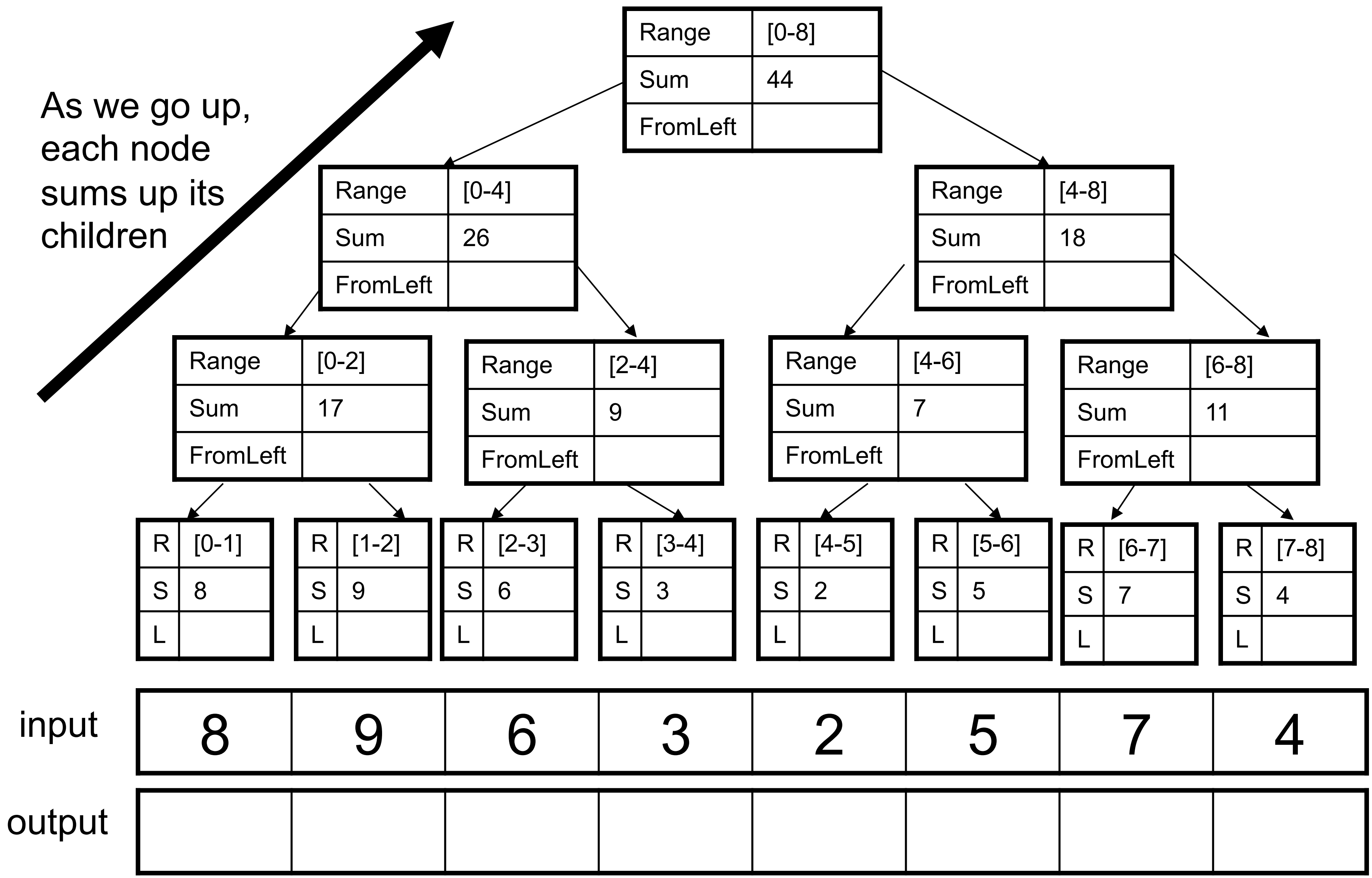
input	8	9	6	3	2	5	7	4
output								

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.

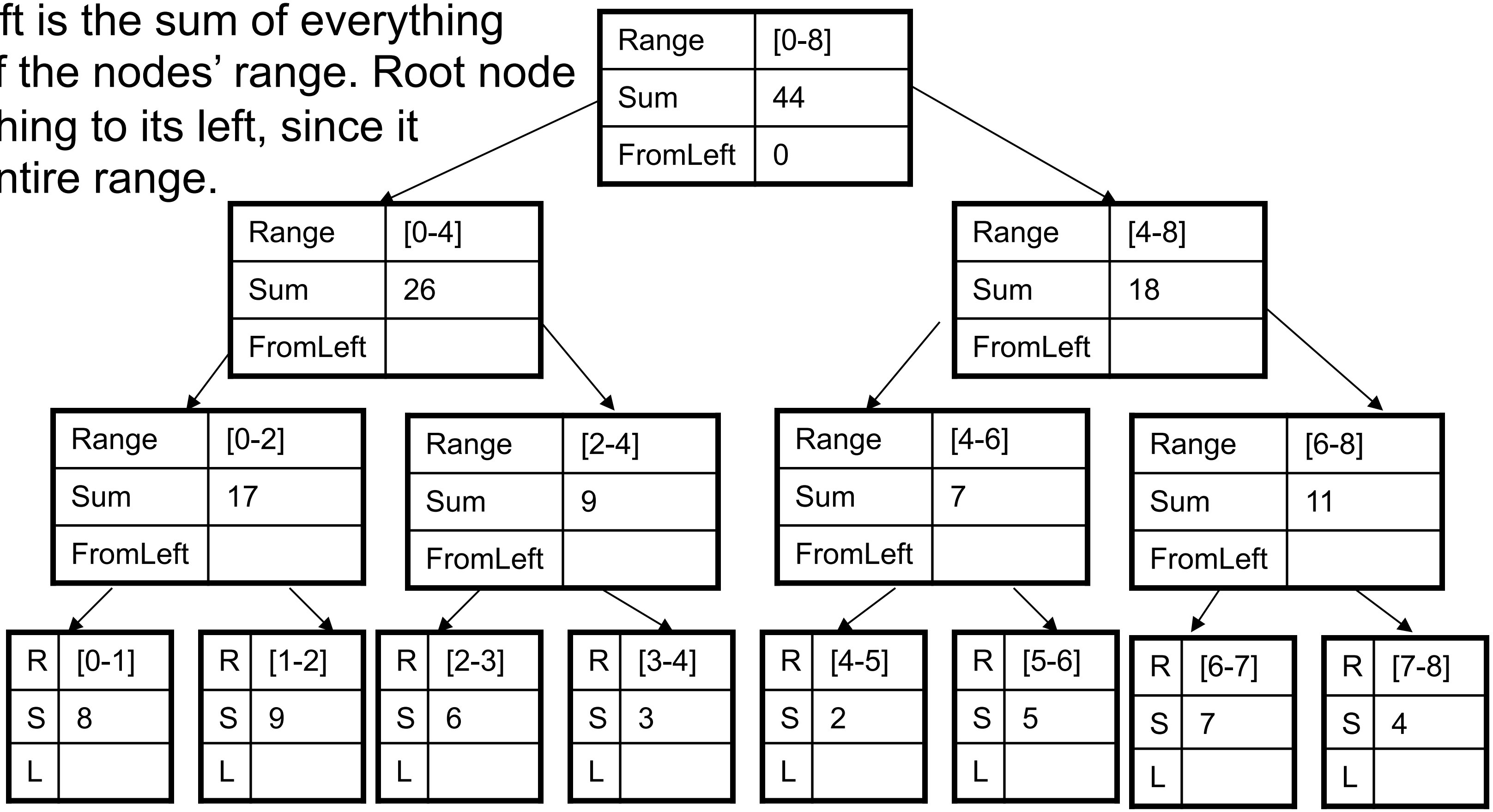


1st pass, find sums going up.



2^{nd} pass, fill out FromLeft going down

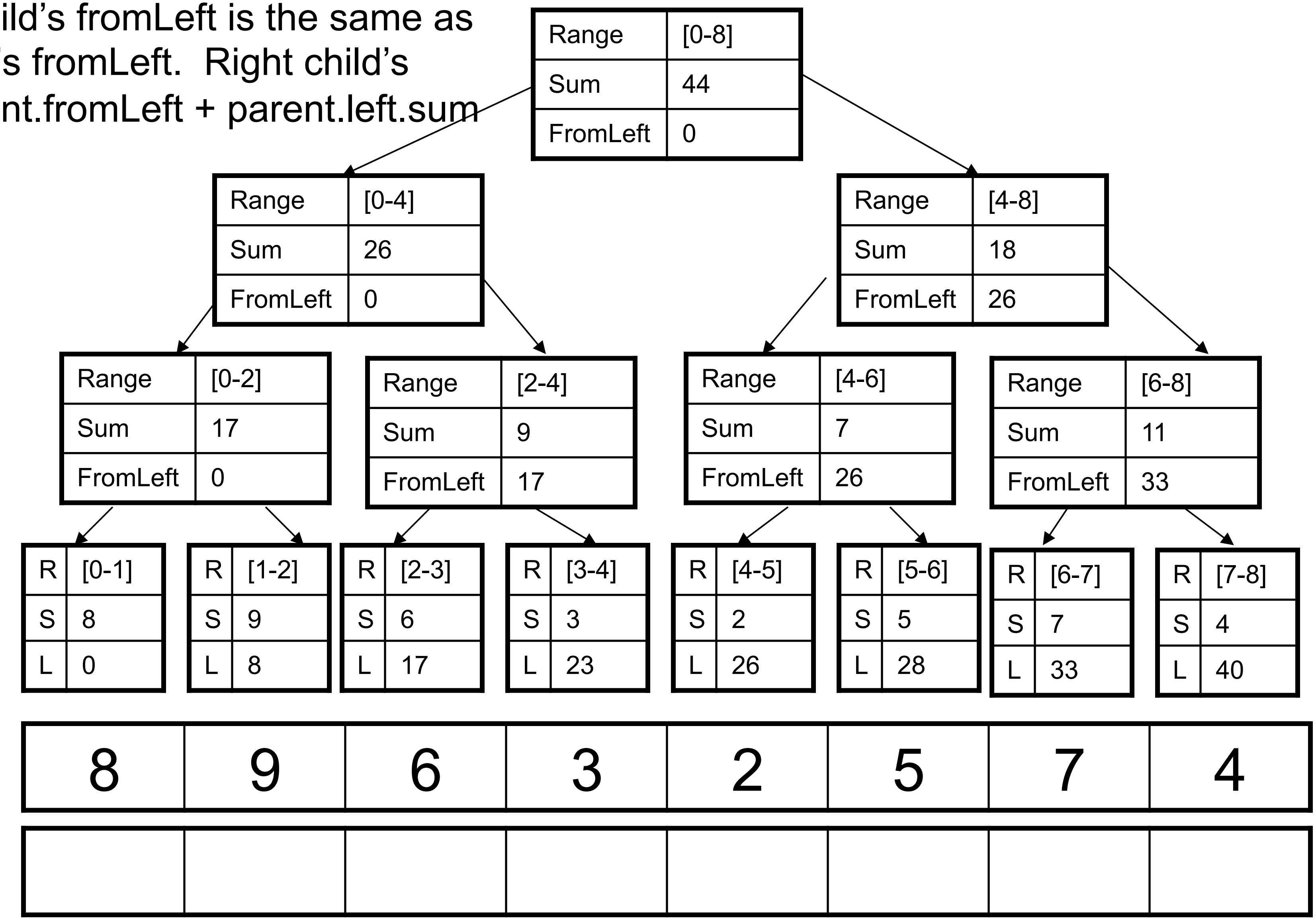
From left is the sum of everything LEFT of the nodes' range. Root node has nothing to its left, since it Is the entire range.



input	8	9	6	3	2	5	7	4
output								

2nd pass, fill out FromLeft going down

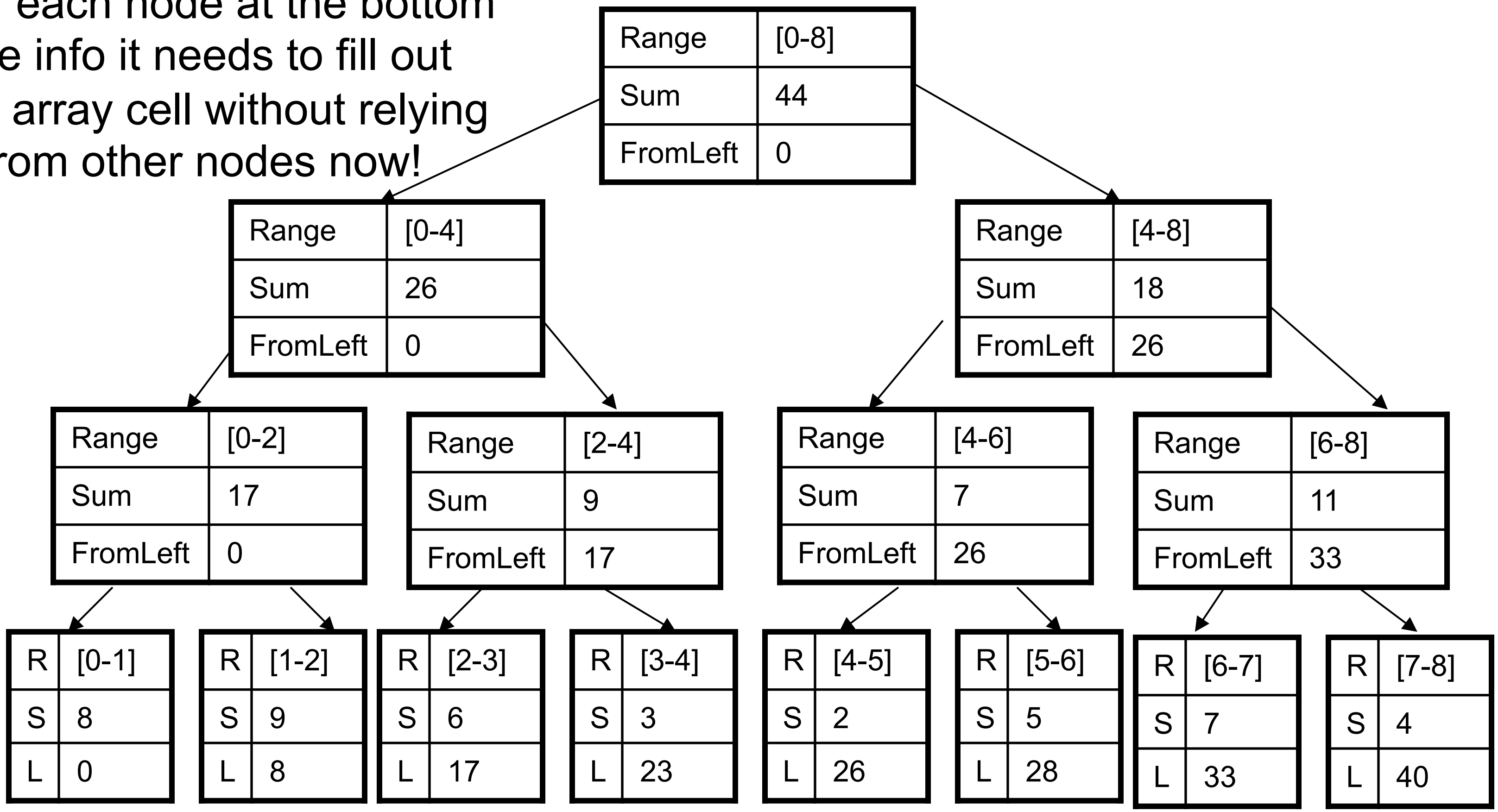
Left child's fromLeft is the same as parent's fromLeft. Right child's is parent.fromLeft + parent.left.sum



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

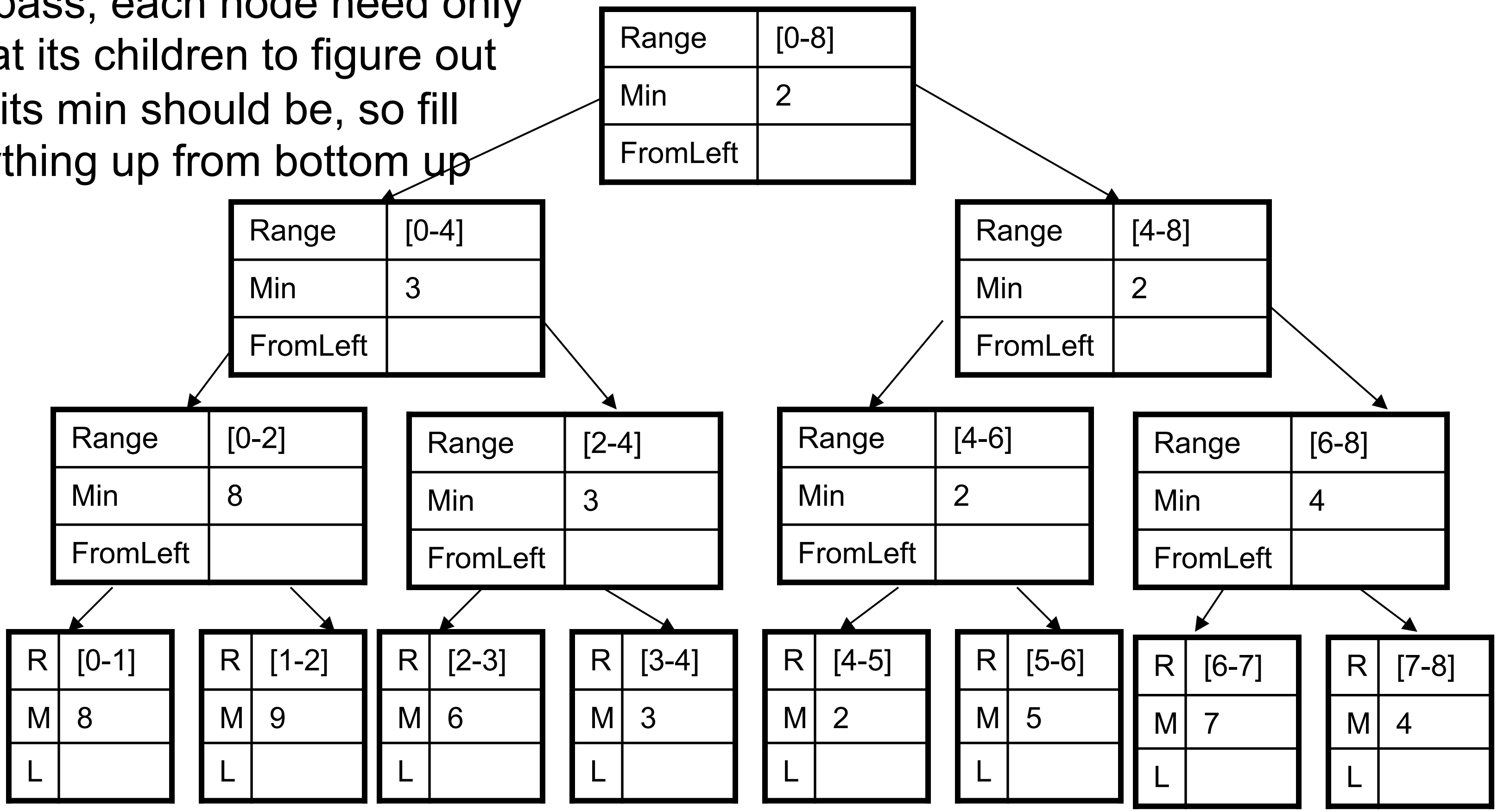
2) Parallel Prefix FindMin

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])

input	8	9	6	3	2	5	7	4
output								

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.

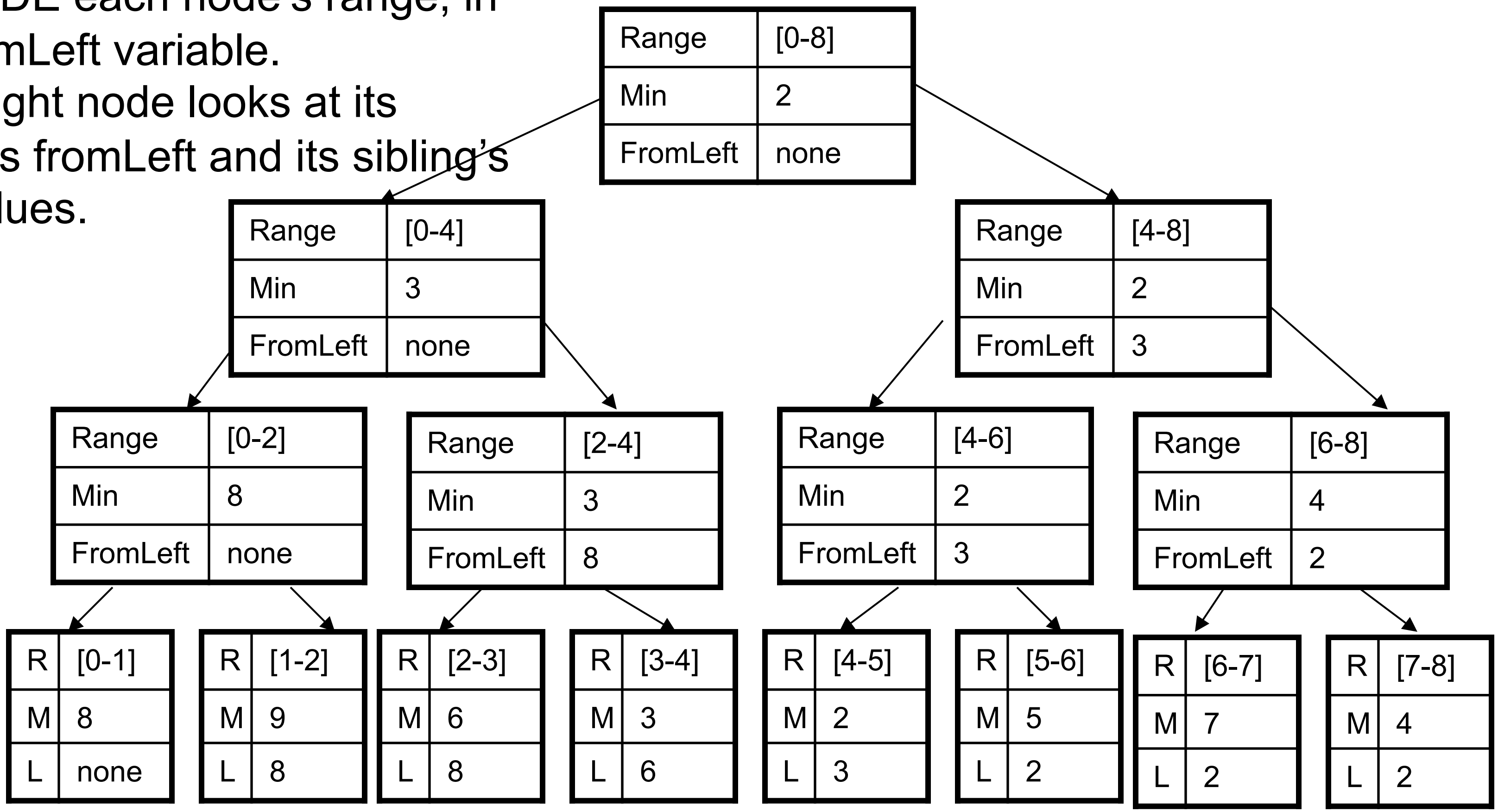
First pass, each node need only look at its children to figure out what its min should be, so fill everything up from bottom up



input	8	9	6	3	2	5	7	4
output								

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

3) Quicksort Recurrence Relations

- Recall that sequential Quicksort consists of
  - O(1) Picking a pivot
  - O(n) Partition data into
    - A: Less than pivot
    - B: Pivot
    - C: Greater than pivot
  - 2 T(n/2) – Recursively, sort each of the two halves, A and C.
- $T(n)=1+n+2T(n/2) = O(n \log n)$

To parallelize step 3 (recursion)

- Each partition can be done at the same, so $2T(n/2)$ becomes time $1\ T(n/2)$
- Whole relation becomes: $T(n)=1+n+T(n/2)$
- Ignoring the constant time pivot-picking:
- $T(n) = n + T(n/2)$

Solve recurrence relation

- $T(n) = n + T(n/2)$
- $T(n) = n + (n/2 + T(n/4))$
- $T(n) = n + (n/2 + (n/4 + T(n/8)))$
- $T(n) = n*(1+1/2+1/4+\ldots+1/2^{k-1})+T(n/2^k)$

Assume T(1)=C, that is, that to sort 1 element takes a constant C units of time.

Substitute in base case T(1)=1 and solve for k:
$n/2^k=1$
$k = \log n$

- $T(n) = n*(1+1/2+1/4+\ldots+1/2^{\log n-1})+C$
- Sum of geometric series (1+1/2+1/4+…) converges to 2
- T(n) = 2n+C which is O(n), linear

4) Parallelizing step 2, partition

- Do 2 filters, one to filter less-than-pivot partition, one to filter greater-than-pivot partition.
- Filter is work O(n), span O(log n)
- So total quicksort is now (partition+recursion):
- $T(n) = O(\log n) + T(n/2)$

Solve recurrence relation

- $T(n) = \log n + T(n/2)$ *expand out recurrence*
- $T(n) = \log n + (\log(n/2) + T(n/4))$
- $T(n) = \log n + \log(n/2) + \log(n/4) + T(n/8)$
- $T(n) = \log n + \log(n/2) + \log(n/4) + \log(n/8) + T(n/16)$
- $T(n) = \log n + (\log n - \log 2) + (\log n - \log 4) + (\log n - \log 8) + T(n/16)$
- $T(n) = 4*\log n - \log 2 - \log 4 - \log 8 + T(n/16)$
- $T(n) = 4*\log n - 1 - 2 - 3 + T(n/2^4)$ *because we’re doing log base 2*
- $T(n) = k*\log n - (1+2+3+\ldots+(k-1))+T(n/2^k)$
- $T(n) = k*\log n - (k(k-1))/2 + T(n/2^k)$
- As usual, assuming $T(1)=C$, set $n/2^k=1$, gives $k=\log n$
- $T(n) = (\log n)*(\log n) - ((\log n-1)(\log n))/2 + C$
- $T(n) = (\log n)*(\log n) - ((\log n * \log n)-\log n)/2 + C$
- Which is $O(\log n * \log n)$