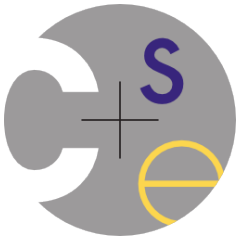
CSE332: Data Abstractions
Section 8



Nicholas Shahan

Winter 2015

Adapted from slides by Hye In Kim

Today

- Announcements
- Questions?
- Worksheet

Announcements

- Project 3 Phase A:
  - Due Monday March 2nd 11pm
- Written HW 7:
  - Due Wednesday March 4th 11pm
- The last week of class is busy… get ahead now

Questions

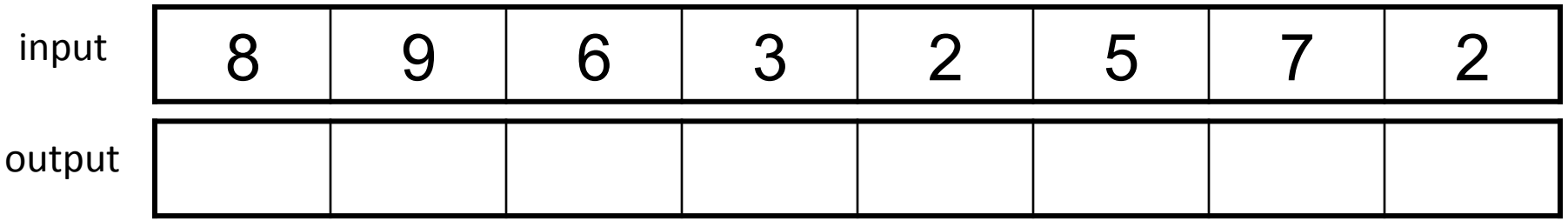
- Written Homework?
- Project 3?
- Parallel or Concurrent Programming so far?

Worksheet

- Work together, these are tricky.

Question 1: Parallel Prefix Sum

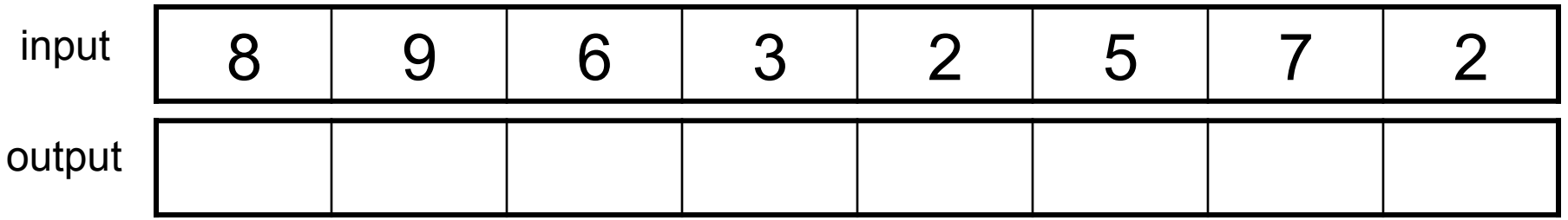
- Output array needs to store sums of everything in the input array up to a certain index.
- Meaning:
  `output[i] =`
  `input[i]+input[i-1]+input[i-2]+…+input[0]`



What Information Do We Need?

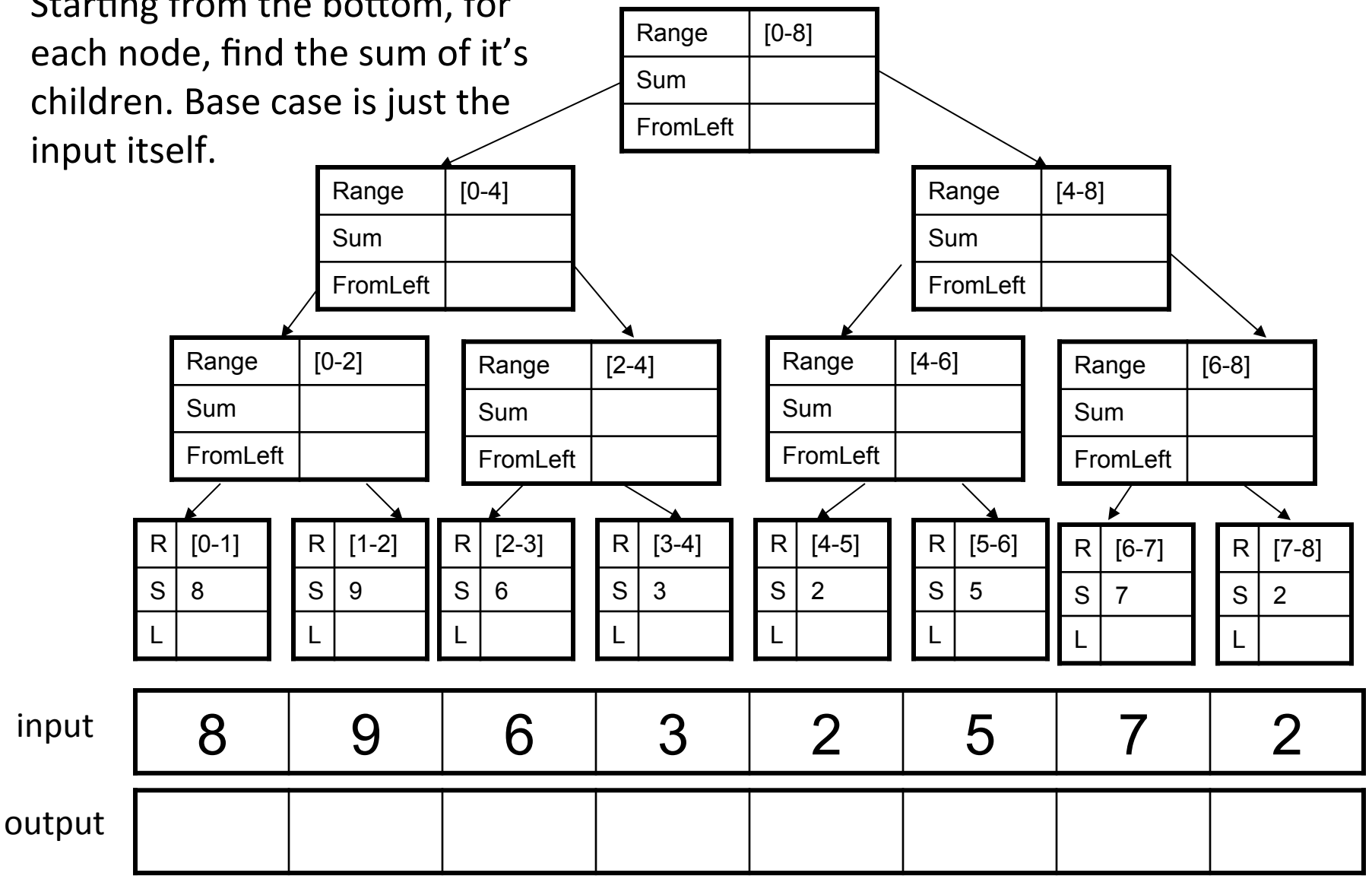
Range	[0-8]
Sum	
FromLeft	

- 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)



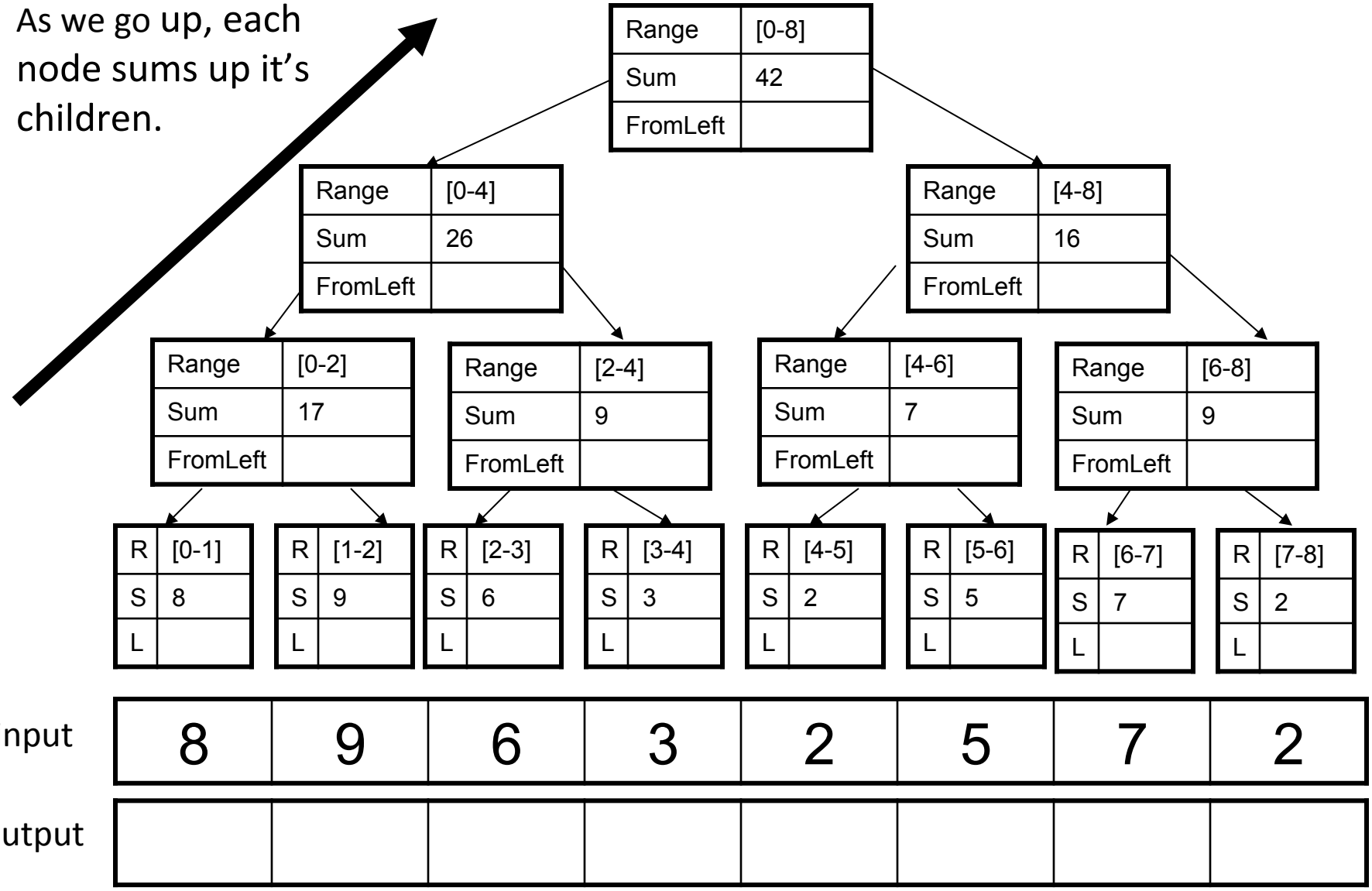
Divide the Problem into Parallel Pieces

Starting from the bottom, for each node, find the sum of it's children. Base case is just the input itself.



1st Pass Finds Sums, Going Up

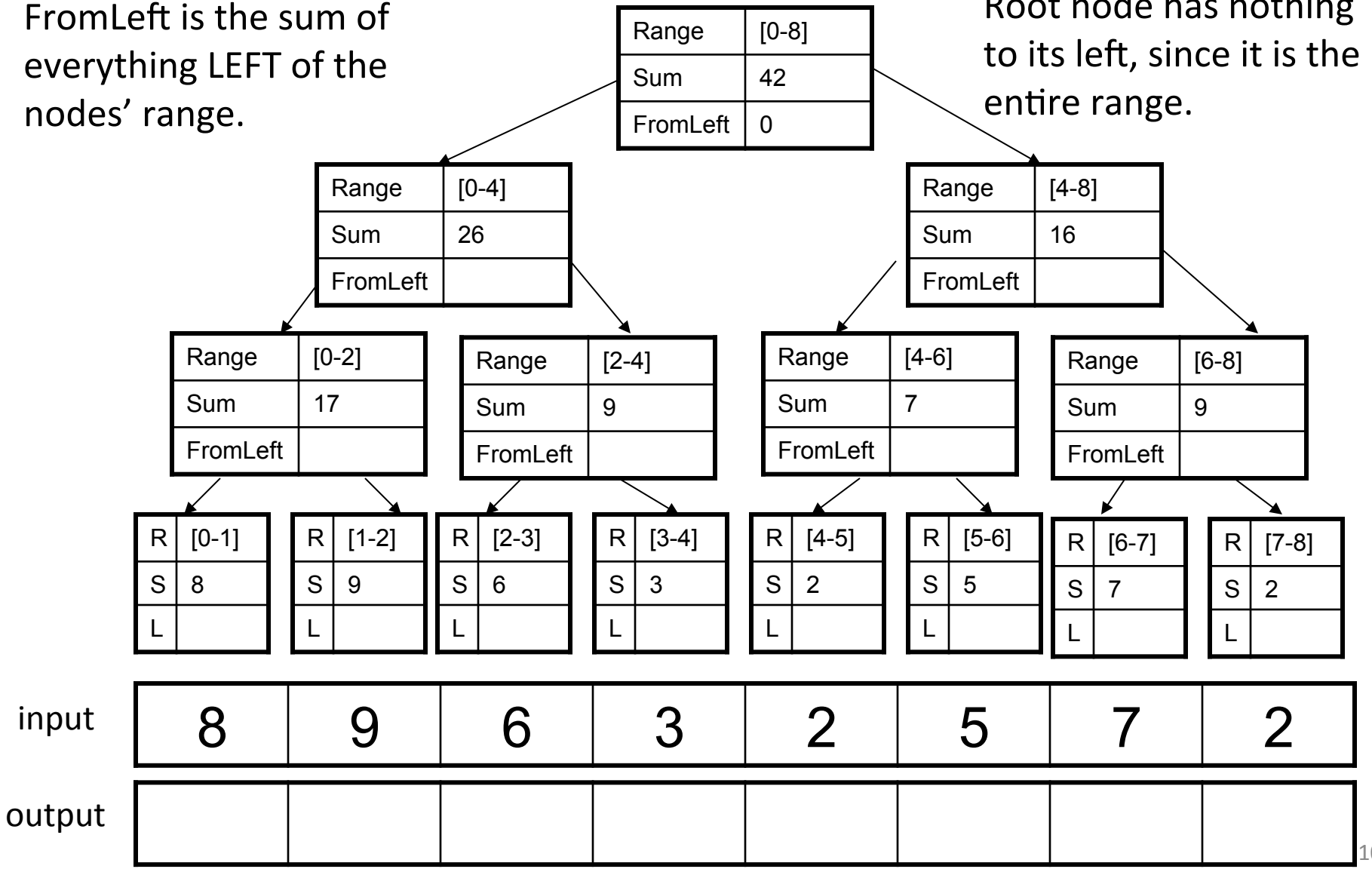
As we go up, each node sums up it's children.



2nd Pass Finds FromLeft, Going Down

FromLeft is the sum of everything LEFT of the nodes' range.

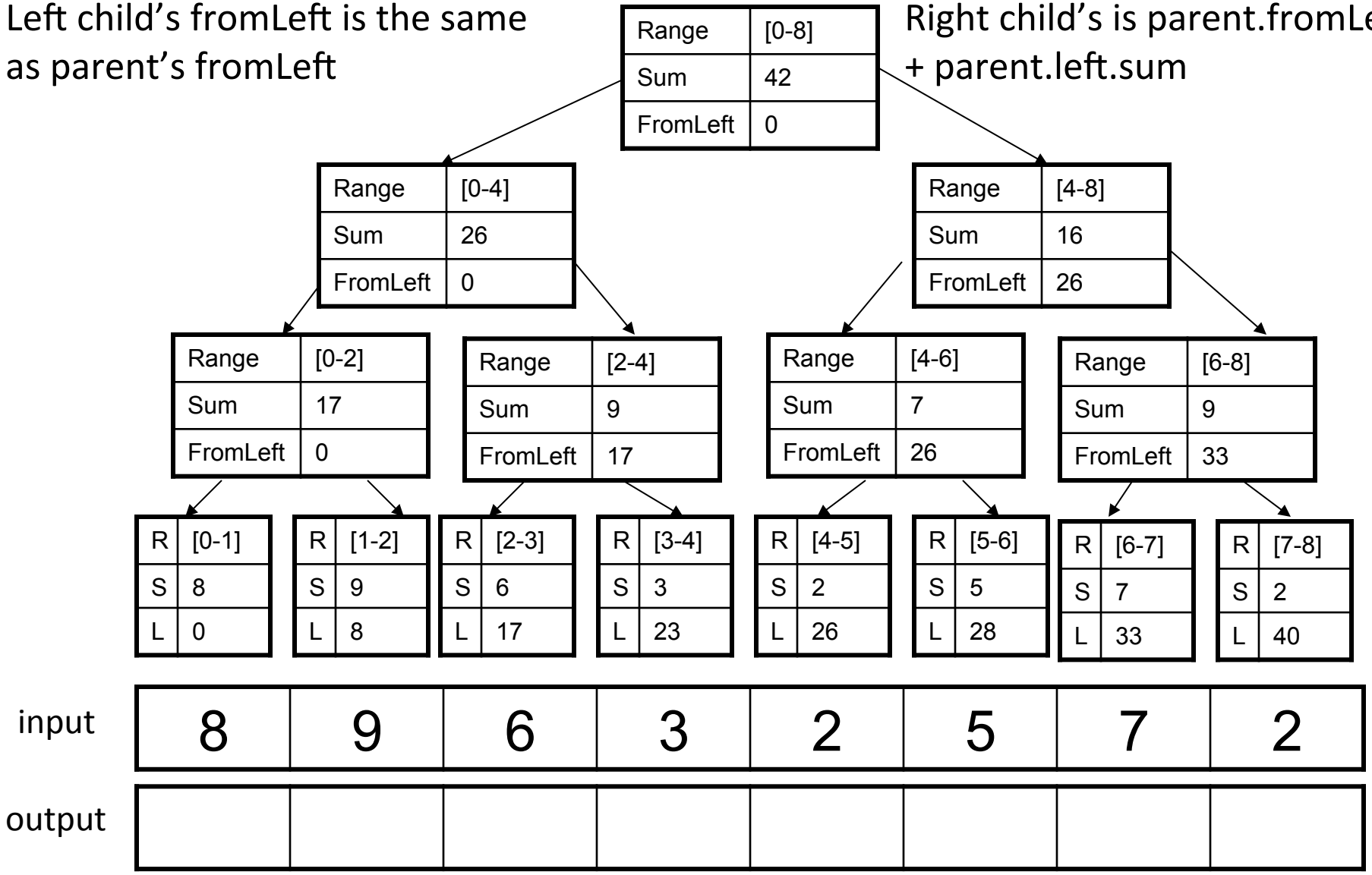
Root node has nothing to its left, since it is the entire range.



2nd Pass Finds FromLeft, Going Down

Left child's fromLeft is the same as parent's fromLeft

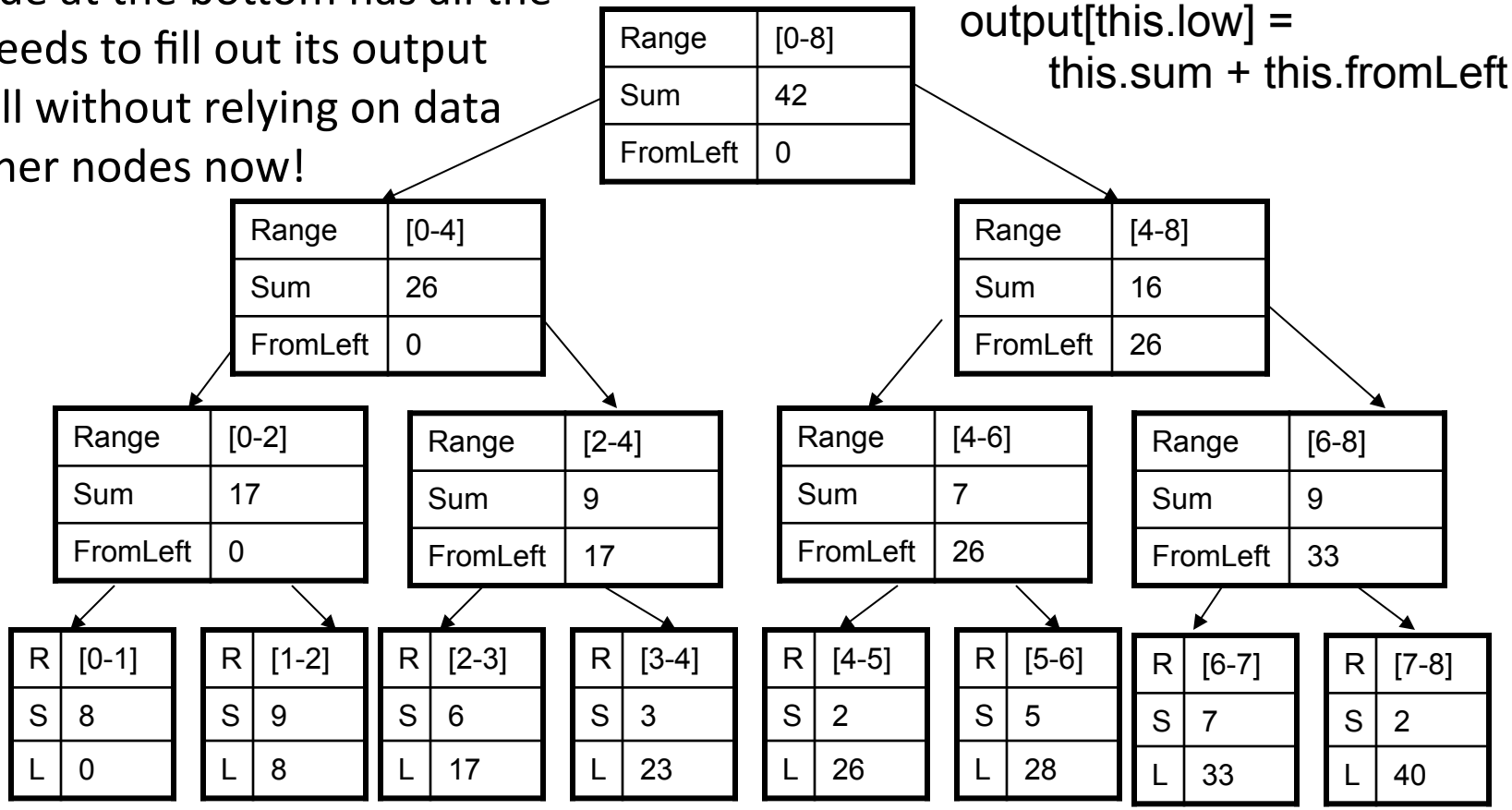
Right child's is parent.fromLeft + parent.left.sum



Finally, Fill the Output Array

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!

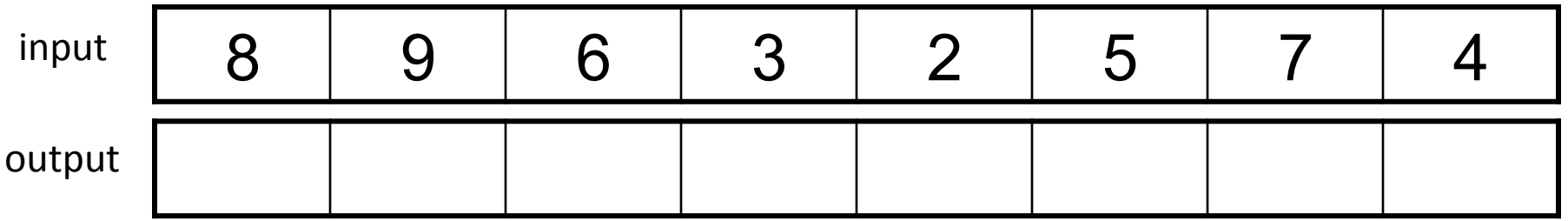
output[this.low] =
this.sum + this.fromLeft



input	8	9	6	3	2	5	7	2
output	8	17	23	26	28	33	40	42

Question 2: Parallel Prefix FindMin

- Output an array with the minimum value of all cells to its left or "the min seen so far".
- Meaning:
  `output[i] =`
  `min(input[0],input[1],...input[i-1],input[i])`



What Information Do We Need?

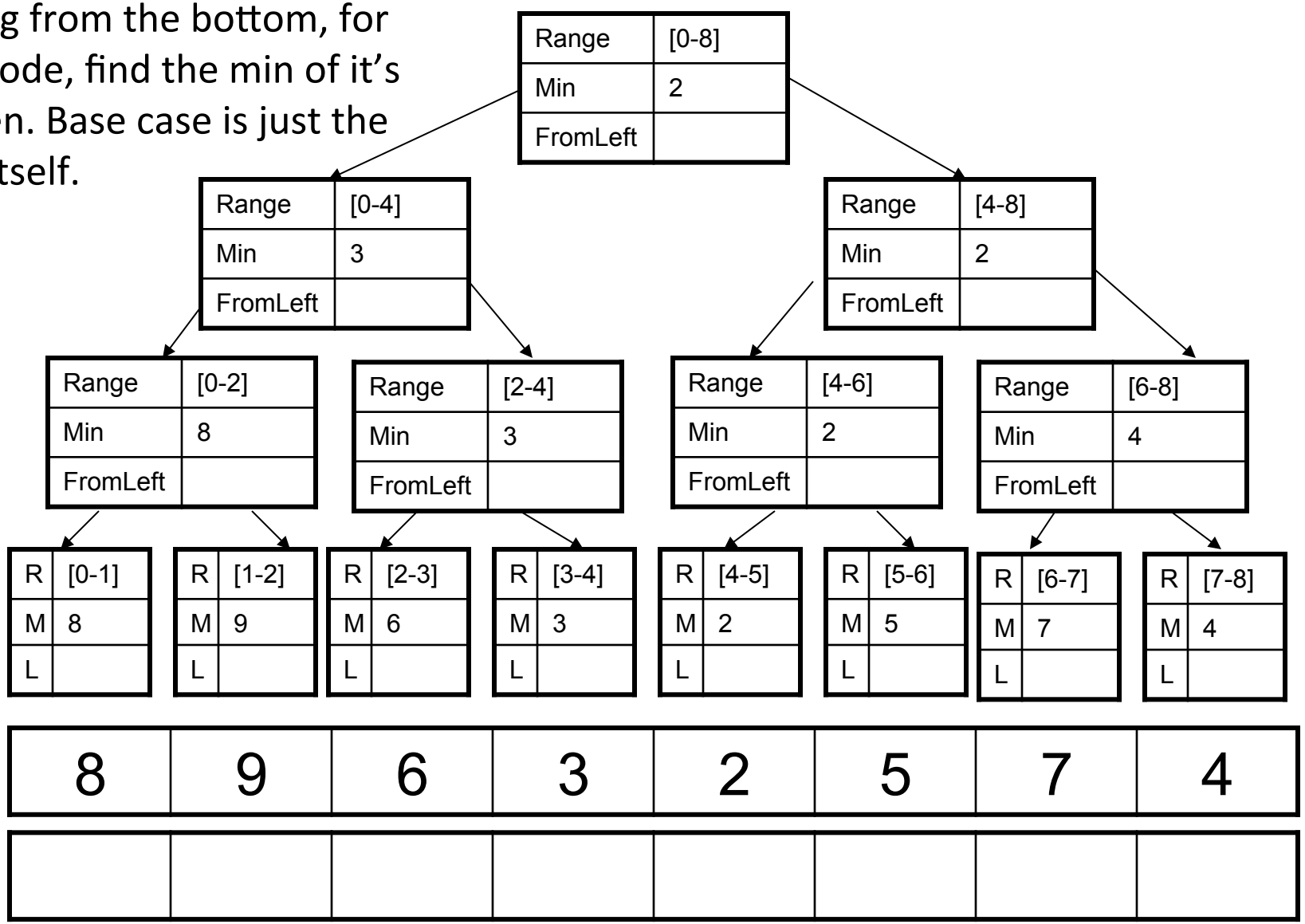
Range	[0-8]
Sum	
FromLeft	

- Same as before, except this time, we want to store the node's range, the min of it's children, and the min of everything to it's left.
- Start off at root with the entire range of the problem (low=0, high=8).

input	8	9	6	3	2	5	7	4
output								

1st Pass Finds Mins, Going Up

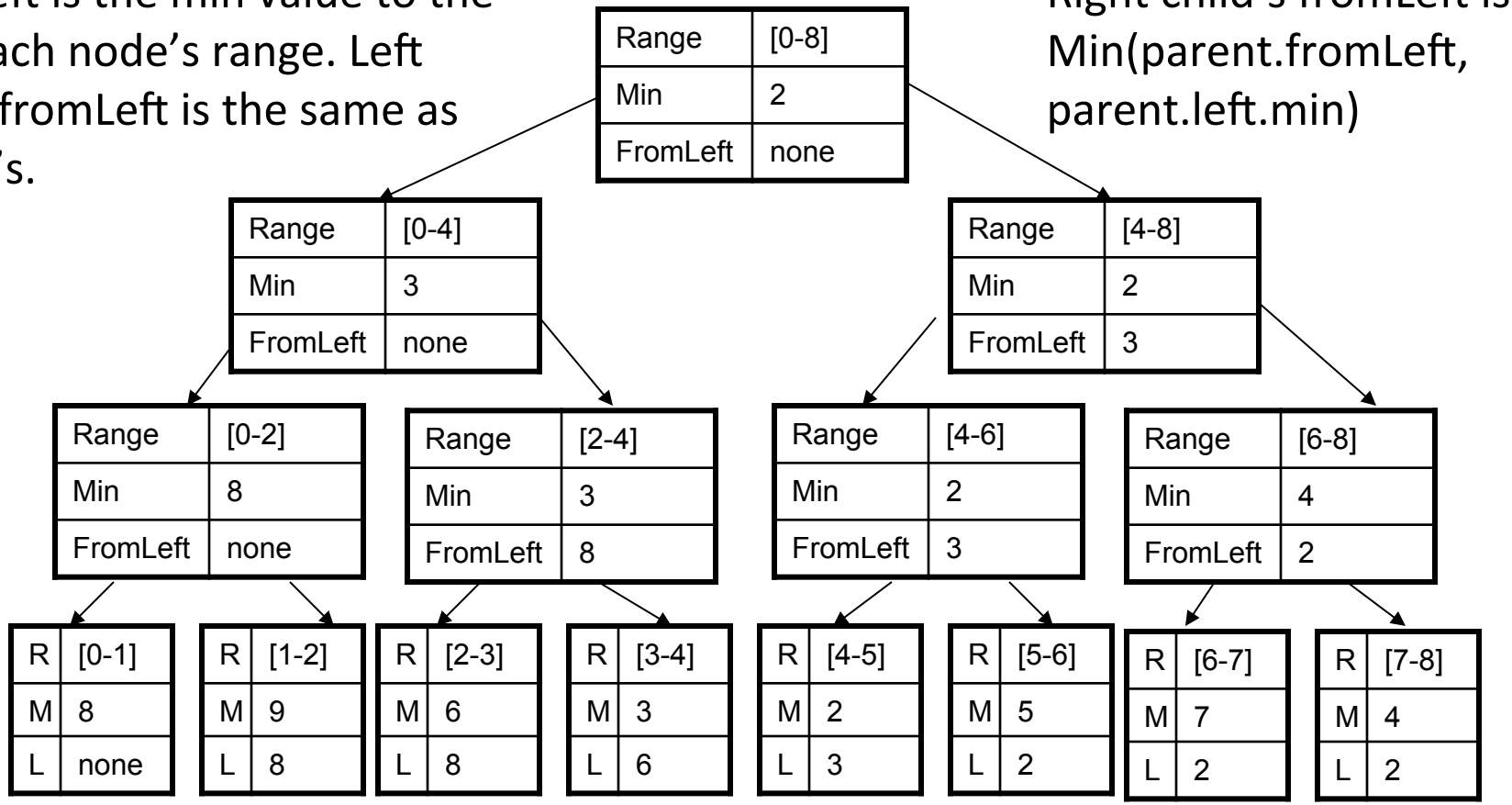
Starting from the bottom, for each node, find the min of it's children. Base case is just the input itself.



2nd Pass Finds FromLeft Going Down

FromLeft is the min value to the LEFT each node's range. Left child's fromLeft is the same as parent's.

Right child's fromLeft is Min(parent.fromLeft, parent.left.min)



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

Question 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:
  $T(n) = n + T(n/2)$

Solving the Recurrence Relation

$T(n) = n + T(n/2)$

$T(n) = n + (n/2 + T(n/4))$

$T(n) = n + (n/2 + (n/4 + T(n/8)))$

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

This will eventually expand down to T(1), the base case.

$n/2^k=1$

$k = \log n$

Solving the 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.

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

- Sum of geometric series $(1 + 1/2 + 1/4 + \ldots)$ converges to 2

$T(n) = 2n + C$ which is O(n), linear

To Parallelize Step 2 (partitioning)

- Do two filters
  - Filter to get the less-than-pivot partition
  - Filter to get the greater-than-pivot partition
- Filter is work O(n), span O(log n)
- Quicksort is now (partition + recursion):
  $T(n) = O(\log n) + T(n/2)$

Solving the Recurrence Relation

Expand out the Recurrence

$T(n) = \log n + T(n/2)$

$T(n) = \log n + (\log(n/2) + T(n/4))$

$T(n) = \log n + \log(n/2) + \log(n/4) + T(n/8)$

$T(n) = \log n + \log(n/2) + \log(n/4) + \log(n/8) + T(n/16)$

$T(n) = \log n + (\log n - \log 2) + (\log n - \log 4)$

$\quad + (\log n - \log 8) + T(n/16)$

$T(n) = 4*\log n - \log 2 - \log 4 - \log 8 + T(n/16)$

Solving the Recurrence Relation

$T(n) = 4*\log n - \log 2 - \log 4 - \log 8 + T(n/16)$

Because we're using log base 2

$T(n) = 4*\log n - 1 - 2 - 3 + T(n/2^4)$

$T(n) = k*\log n - (1 + 2 + 3 + \ldots + (k-1)) + T(n/2^k)$

$T(n) = k*\log n - (k(k-1))/2 + T(n/2^k)$

Solving the Recurrence Relation

$T(n) = k*\log n - (k(k-1))/2 + T(n/2^k)$

As usual, set $n/2^k=1$, gives $k=\log n$

Then assume $T(1)=C$

$T(n) = (\log n)*(\log n) - ((\log n-1)(\log n))/2 + C$

$T(n) = (\log n)*(\log n) - ((\log n * \log n)-\log n)/2 + C$

Which is $O(\log n * \log n)$