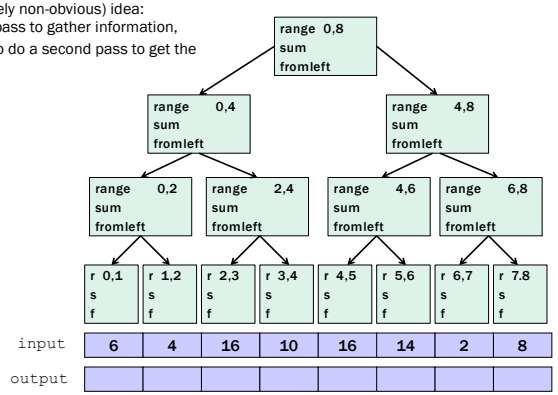
The (completely non-obvious) idea:
Do an initial pass to gather information,
enabling us to do a second pass to get the answer



7

Pack (think "Filter")

Given an array **`input`**, produce an array **`output`** containing only elements such that **`f(element)`** is **`true`**

Example: **`input [17, 4, 6, 8, 11, 5, 13, 19, 0, 24]`**

`f: "is element > 10"`

`output [17, 11, 13, 19, 24]`

Parallelizable?

- Determining *whether* an element belongs in the output is easy
- But determining *where* an element belongs in the output is hard; seems to depend on previous results....

15

Parallel Quicksort VERSION 2

Best / expected case *work*

1. Pick a pivot element — O(1)
2. Partition all the data into: — O(n)
   A. The elements less than the pivot
   B. The pivot
   C. The elements greater than the pivot
3. Recursively sort A and C — 2T(n/2)

Idea: Do the partition with some parallel prefix packing

Work:

Span:

27

Parallel Merge Sort

Let's just analyze the ***merge***:

What's the worst case?

One subarray has ¾ of the elements, the other has ¼ .

This is why we start with the median of the larger array.

Work: $T_1(n) =$

Span: $T_\infty(n) =$

38