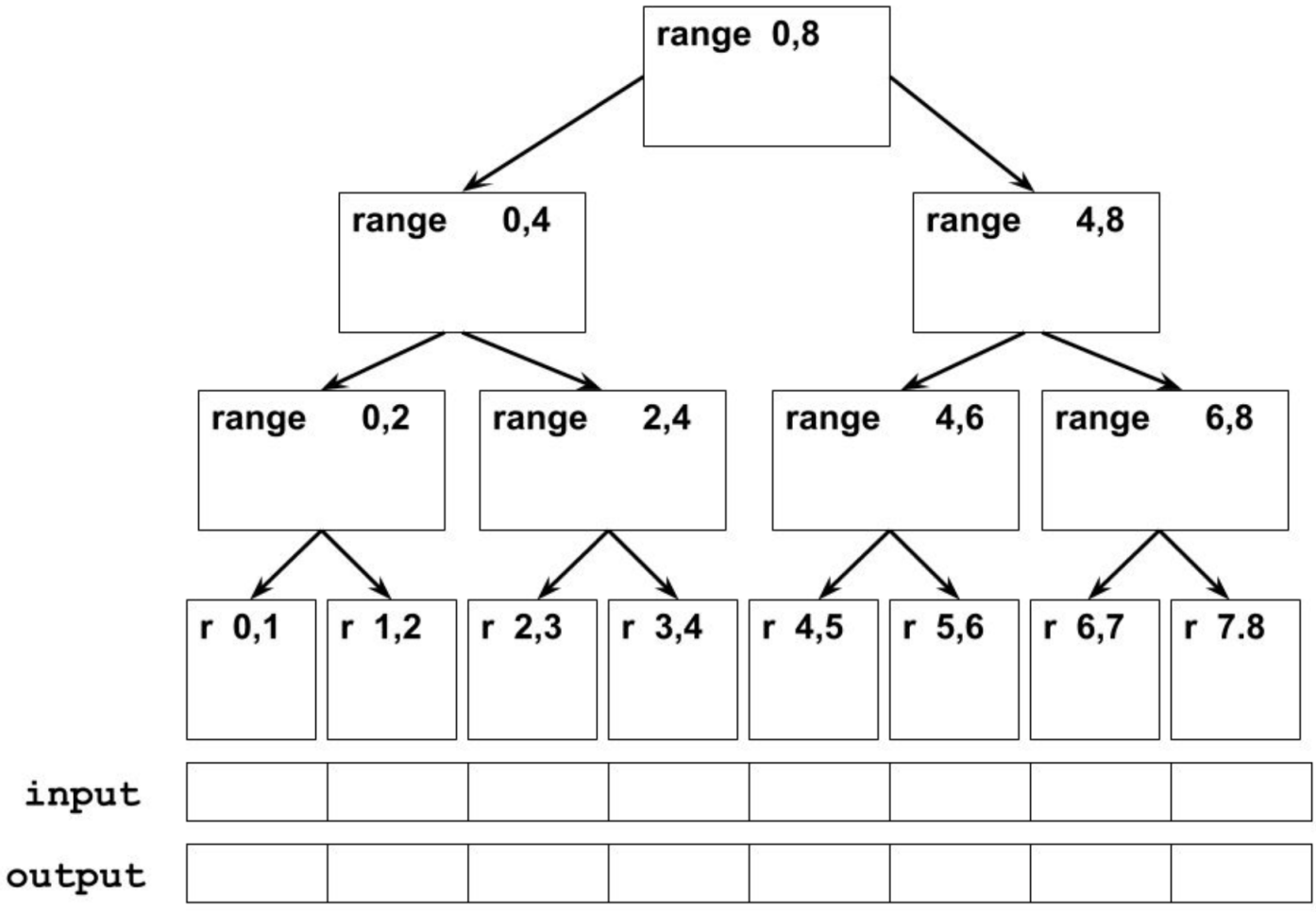
CSE 332: Data Structures and Parallelism

Section 7: Parallel Primitives Solutions

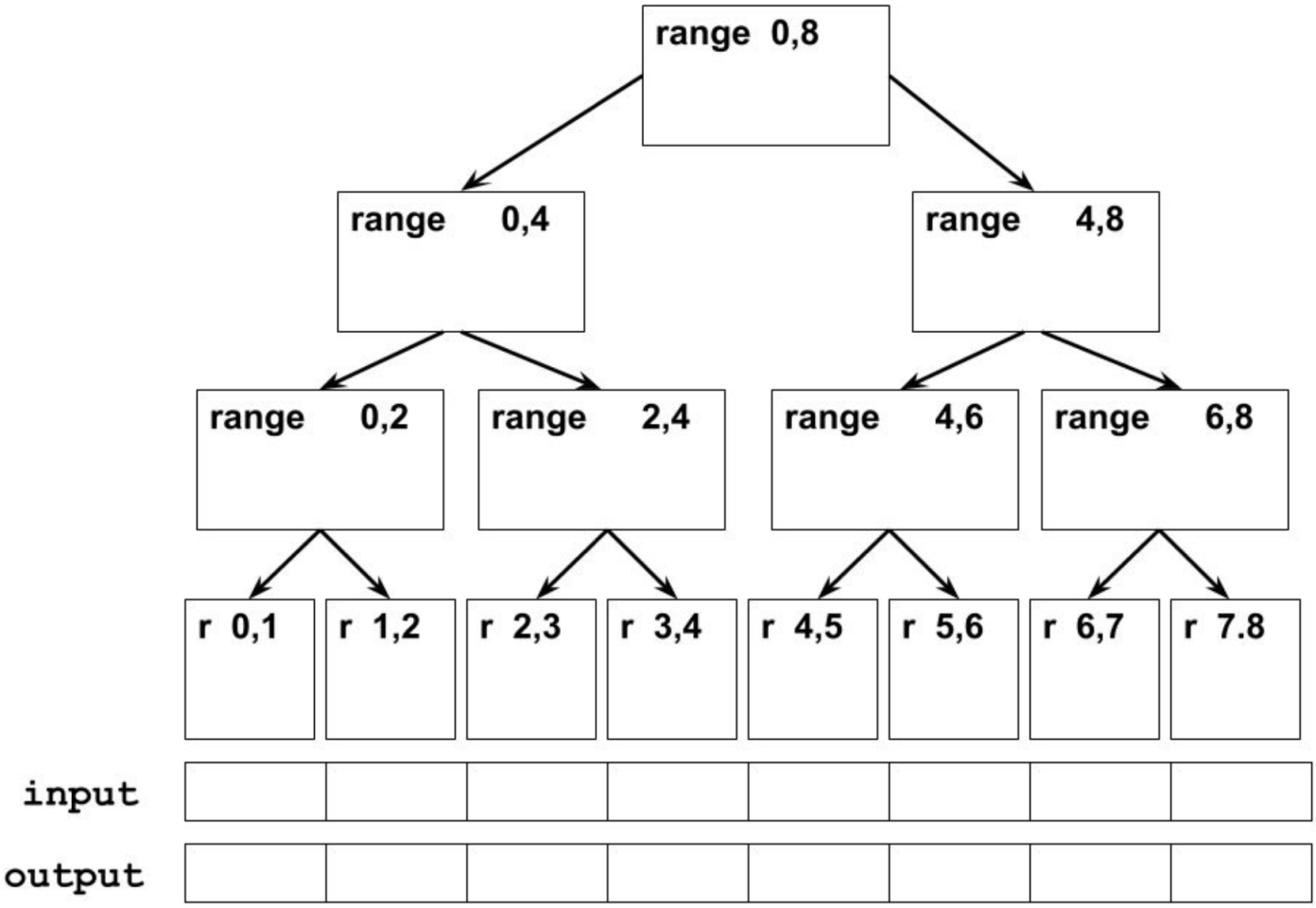
0. Parallel Prefix Sum

Given input array $[8, 9, 6, 3, 2, 5, 7, 4]$, output an array such that each output$[i]$ = sum(array$[0]$, array$[1]$, ... array$[i]$), using the Parallel Prefix Sum algorithm from lecture. Show the intermediate steps. Draw the input and output arrays, and for each step, show the tree of the recursive task objects that would be created (where a node's child is for two problems of half the size) and the fields each node needs. Do not use a sequential cut-off.



1. Parallel Prefix FindMin

Given an input array $[8, 9, 6, 3, 2, 5, 7, 4]$, output an array such that each output$[i]$ = min(array$[0]$, array$[1]$, ... array$[i]$). Show all steps, as above.



2. MiniMax

(a) Explain why we *negate* the result of the recursive call in MiniMax.

Solution:

We do this in order to simulate the idea of switching between your perspective and your opponent's perspective. By using the assumption that your opponent is playing optimally, if you negate the result of the recursive call, then on your opponent's turn, they will choose the move with the lowest value for you. When the call returns, you choose the highest value from the choices you've simulated for your opponent.

3. Cutoffs

Provide a short diagram or description to explain the following parameters from P3:

(a) `ply`

Solution:

The total number of levels ahead looked (so, the height of your search tree).

(b) `cutoff`

Solution:

The number of levels remaining in the tree when your search switches to a non-parallel algorithm.

(c) `divideCutoff`

Solution:

The maximum number of threads which should be forked sequentially when dividing-and-conquering a list of moves. Similar to the `sequentialCutoff` parameter from exercises.

4. Efficiency

Circle the **most efficient** option from each pair of possible implementation strategies for P3:

(a) To create threads for each move in a `List<M>` during Parallel Minimax:

Create threads in a `for` loop **OR** Create threads with divide-and-conquer

Solution:

Create threads with divide-and-conquer.

(b) To pass copies of boards to these threads:

Copy the board *inside* the thread **OR** Copy the board *before* passing it to the thread

Solution:

Copy the board *inside* the thread.