CSE 332: Data Structures and Parallelism

Section 7: ForkJoin & Parallel Prefix

Part One: ForkJoin

0. LessThan7

LessThan7 returns the number of elements in `arr` that are less than 7. For example, if `arr` is `[21, 7, 6, 8, 17, 1]`, then `lessThan7(arr) == 2`.

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

public class LessThan7 {
   private static final ForkJoinPool POOL = new ForkJoinPool();
   private static int CUTOFF;

   public static int parallelLessThan7(int[] arr, int cutoff) {
      // TODO: Invoke the ForkJoinPool to call the LessThan7Task
       LessThan7.CUTOFF = cutoff;


   }

   public static int sequentialLessThan7(int[] arr, int lo, int hi) {
      // TODO: Step 1. Base Case (i.e. Sequential Case)




   }

   private static class LessThan7Task extends RecursiveTask<Integer> {
       private final int[] arr;
       private final int lo, hi;

       public LessThan7Task(int[] arr, int lo, int hi) {
           this.arr = arr;
           this.lo = lo;
           this.hi = hi;
       }
```

```
        @Override
        protected Integer compute() {
            if (hi - lo <= LessThan7.CUTOFF) {
             // TODO: Step 1. Base Case (i.e. Sequential Case)
                    


            } else {
             // TODO: Step 2. Recursive Case (i.e. Parallel/Forking case)
                    


             // TODO: 1. Make sure to fork() the left task first
                    

             // TODO: 2. Then compute() the right task
                    

             // TODO: 3. Then wait for the leftResult by calling join()on the
             left task before combining results
                    

             // TODO: Step 3. Combining the left and right tasks' results
                    


            }
        }
    }
}
```

1. Parity

Parity returns `true` if there are an even number of even numbers and `false` otherwise. For example, if `arr` is `[1, 7, 4, 3, 6]`, then `parity(arr) == true`. But, if `arr` is `[6, 5, 4, 3, 2, 1]`, `parity(arr) == false`.

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

public class Parity {
   private static final ForkJoinPool POOL = new ForkJoinPool();
   private static int CUTOFF;

   public static boolean parallelParityTask(int[] arr, int cutoff) {
       // TODO: Invoke the ForkJoinPool to call the LessThan7Task
       Parity.CUTOFF = cutoff;


   }

   public static boolean sequentialParityTask(int[] arr, int lo, int hi) {
       // TODO: Step 1. Base Case (i.e. Sequential Case)




   }

   private static class ParityTask extends RecursiveTask<Boolean> {
       private final int[] arr;
       private final int lo, hi;

       public ParityTask(int[] arr, int lo, int hi) {
           this.arr = arr;
           this.lo = lo;
           this.hi = hi;
       }
```

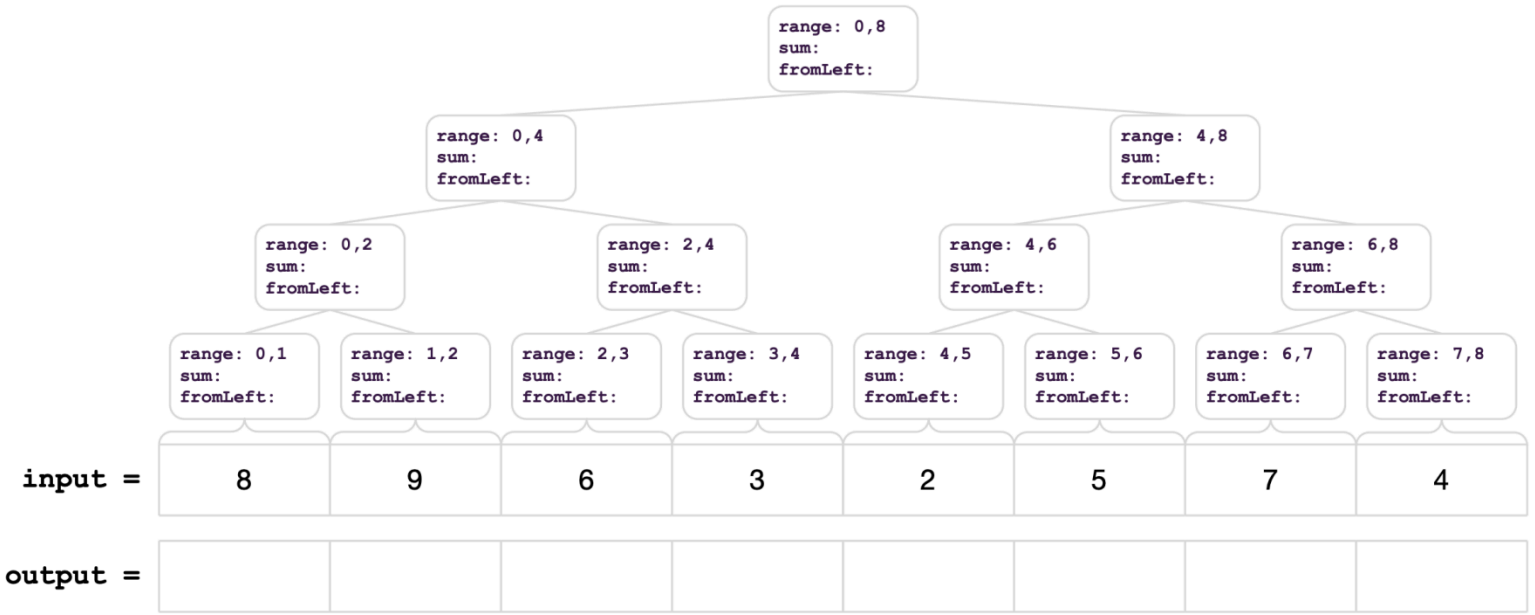
```
        @Override
        protected Boolean compute() {
            if (hi - lo <= CUTOFF) {
                // TODO: Step 1. Base Case (i.e. Sequential Case)
                


            } else {
                // TODO: Step 2. Recursive Case (i.e. Parallel/Forking case)
                


                // TODO: Step 3. Combining the left and right tasks' results
                


            }
        }
    }
}
```

Part Two: Parallel Prefix

0a. 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])`.

Use 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.



leaves[i].sum = _____;

parent.sum = _____;

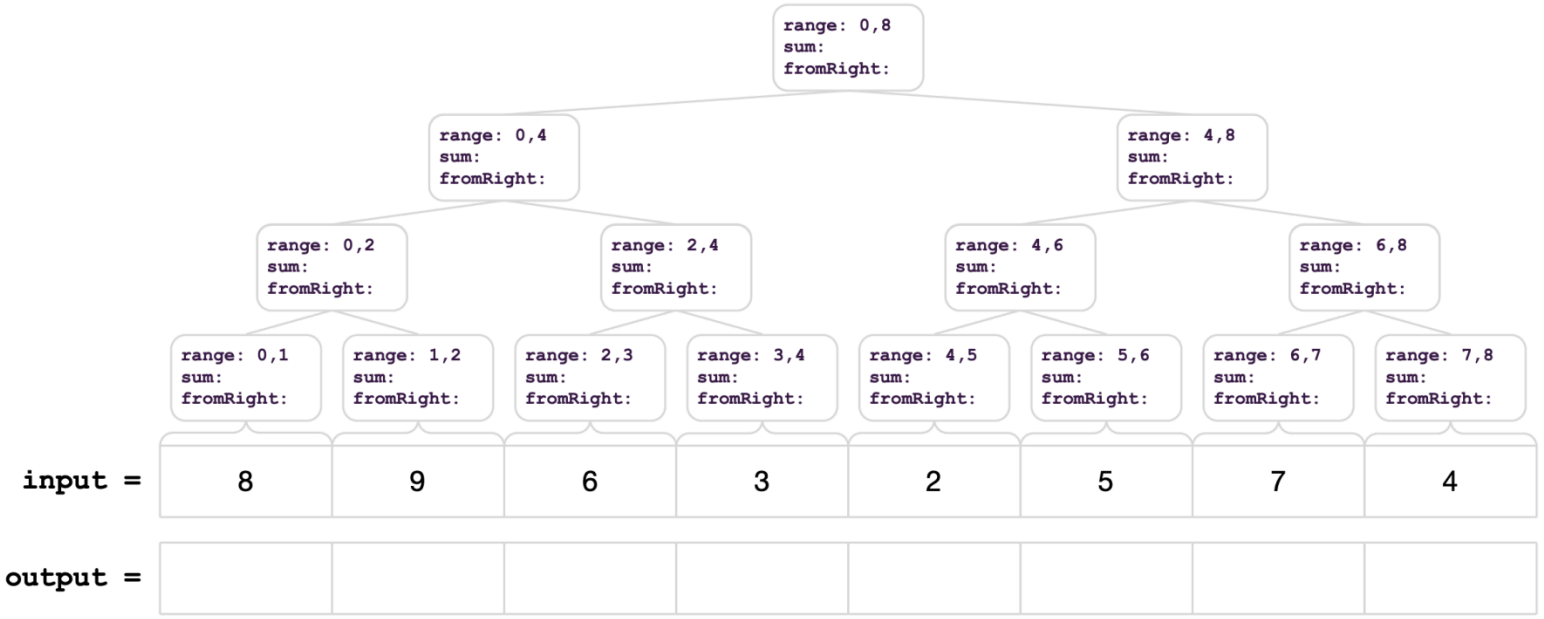
left.fromLeft = _____;
right.fromLeft = _____;

output[i] = _____;

0b. Parallel Suffix 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])`.

Use the Parallel Suffix algorithm. 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.



leaves[i].sum = _____;

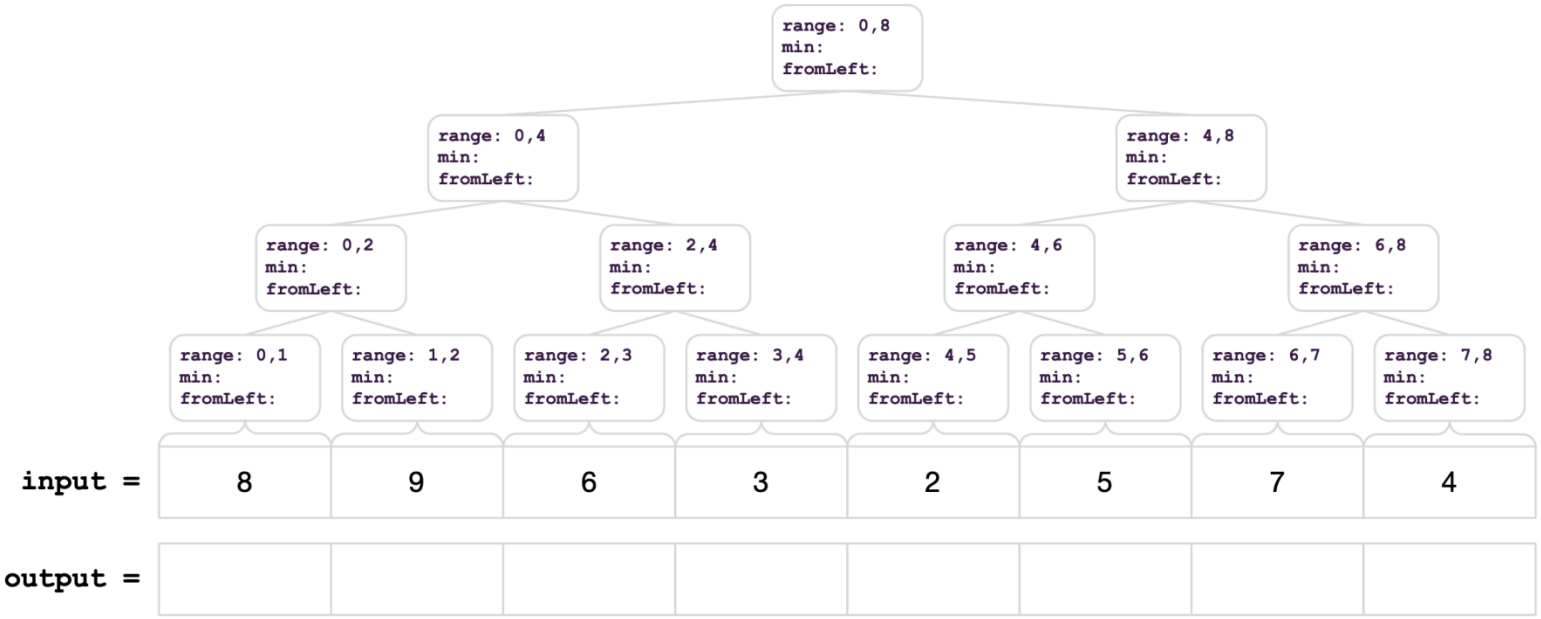
parent.sum = _____;

left.fromRight= _____;
right.fromRight= _____;

output[i] = _____;

1. Parallel Prefix FindMin

Given 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.



leaves[i].min = _____;

parent.min = _____;

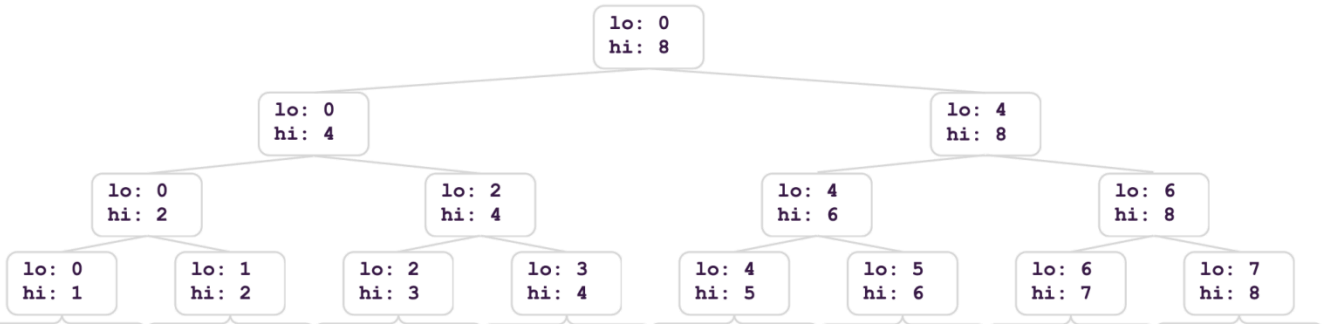
left.fromLeft = _____;
right.fromLeft = _____;

output[i] = _____;

2. Parallel Pack

Given input array `[12, 5, -8, 34, 6, 10, 2, 7]`, output an array that contains only the elements that are less than 10.

Use the Parallel Pack 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.



input =	12	5	-8	34	6	10	2	7
bits =								
bitsum =								

output =

1) Compute bits:

2) Compute bitsum:

3) Produce output:

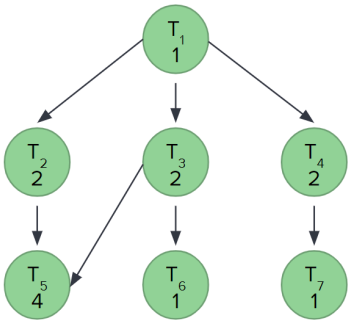
3. Work it Out [for next section]

a) Define work and span.

b) How do we calculate work and span?

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

d) What is the total work and span of this task graph?



e) Suppose a program needs to do 20% of its work sequentially (the rest can be parallelized), what is the maximum speed up with 4 processors?