Finishing fork/join + some priority queues

5/8/26

Administrivia

- HW 6 (BSTs, tree traversals, design problems) due tonight
- Exam 2 is on Monday
  - In class, open note, written exam
  - Topics:
    - Data structures: Stack, Queue, Set, Map, Priority Queue, StringBuilder
    - Sorting and comparators
    - Design problems
    - Binary search and binary search trees
    - Tree traversals
    - More questions on running time
    - Search (backtracking and connected components)
    - Inheritance

Where we were

```
int sum(int[] array, int low, int hi) {
        //base case
        int left = fork sum(array, low, (hi+low)/2);
        int right = sum(array, (hi+low)/2, hi);
        join;
        return left + right;
}
```

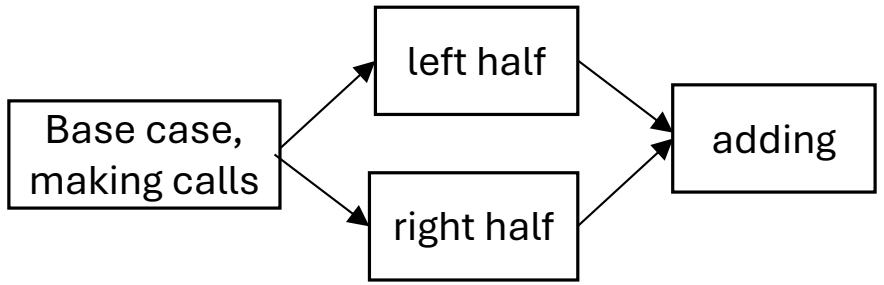
Creates many more strands than cores

As each core finishes a strand, it gets another

Strands may have initial assignment, but any core can potentially take them (*work stealing*)

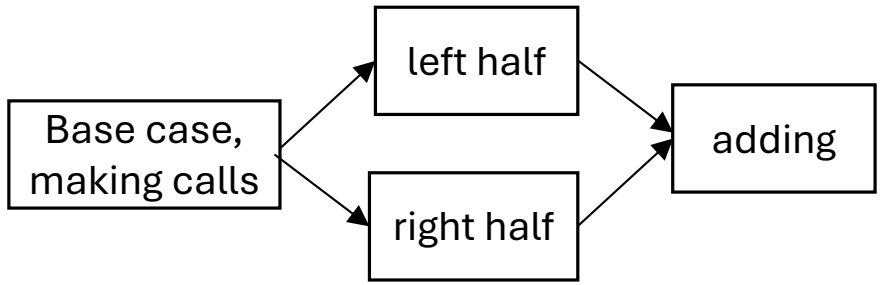
Precedence graph for this code

```
int sum(int[] array, int low, int hi) {
  //base case


  int left = fork sum(array, low, (hi+low)/2);
  int right = sum(array, (hi+low)/2, hi);
  join;
  return left + right;
}
```



Precedence graph for this code

```
int sum(int[] array, int low, int hi) {
  //base case

  int left = fork sum(array, low, (hi+low)/2);
  int right = sum(array, (hi+low)/2, hi);
  join;
  return left + right;
}
```

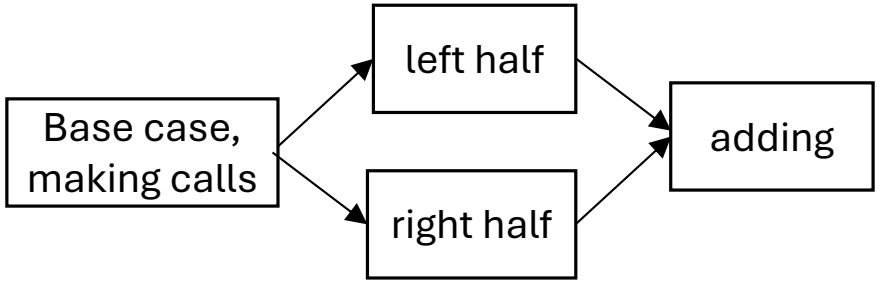


What is the span of this code? (n=hi-low)

A. Θ(1)
B. Θ(log n)
C. Θ(n)
D. Θ(n log n)
E. None of the above

Precedence graph for this code

```
int sum(int[] array, int low, int hi) {
  //base case

  int left = fork sum(array, low, (hi+low)/2);
  int right = sum(array, (hi+low)/2, hi);
  join;
  return left + right;
}
```

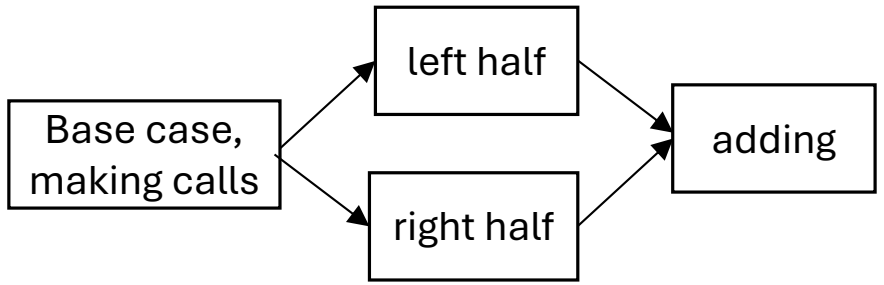


What is the span of this code? (n=hi-low)

A. Θ(1)
B. Θ(log n)
C. Θ(n)
D. Θ(n log n)
E. None of the above

Precedence graph for this code

```
int sum(int[] array, int low, int hi) {
  //base case

  int left = fork sum(array, low, (hi+low)/2);
  int right = sum(array, (hi+low)/2, hi);
  join;
  return left + right;
}
```

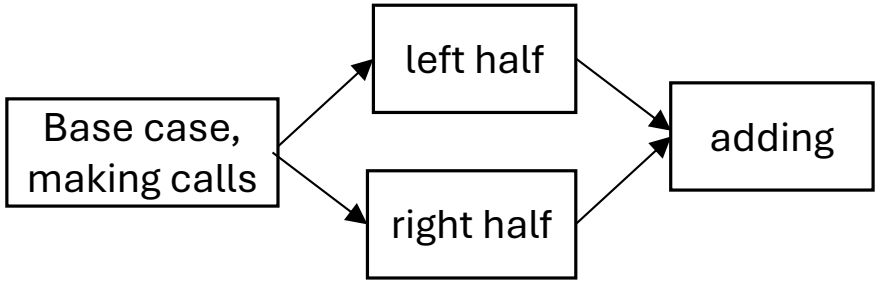


What is the work of this code? (n=hi-low)

A. Θ(1)
B. Θ(log n)
C. Θ(n)
D. Θ(n log n)
E. None of the above

Applying this to a program

```
int sum(int[] array, int low, int hi) {
  //base case

  int left = fork sum(array, low, (hi+low)/2);
  int right = sum(array, (hi+low)/2, hi);
  join;
  return left + right;
}
```



What is the work of this code? (n=hi-low)

A. Θ(1)
B. Θ(log n)
C. Θ(n)
D. Θ(n log n)
E. None of the above

What is the work and span of this code?

(n is hi-low)

```
int sum(int[] array, int low, int hi) {
        int total = 0;
        for(int i=low; i < hi; i++)
                total += array[i];
        return total;
}
```

A. $\Theta(\log n)$ and $\Theta(\log n)$
B. $\Theta(\log n)$ and $\Theta(n)$
C. $\Theta(n)$ and $\Theta(\log n)$
D. $\Theta(n)$ and $\Theta(n)$
E. None of the above

What is the work and span of this code?

(n is hi-low)

```
int sum(int[] array, int low, int hi) {
        int total = 0;
        for(int i=low; i < hi; i++)
                total += array[i];
        return total;
}
```

A. Θ(log n) and Θ(log n)
B. Θ(log n) and Θ(n)
C. Θ(n) and Θ(log n)
D. Θ(n) and Θ(n)
E. None of the above

Putting this into actual Java

```
int sum(int[] array, int low, int hi) {
    …
}
```

```
class Sum extends RecursiveTask<Integer> {

    protected Integer compute() {
        …
    }
}
```

Use a class to represent a call to sum

Parameterized by the return type (must be a reference type)

Putting this into actual Java

```
int sum(int[] array, int low, int hi) {
    …
}
```

Can’t pass arguments into compute

Use constructor to get the information

```
class Sum extends RecursiveTask<Integer> {
    int[] array;    int low;    int high;

    public Sum(int[] array, int low, int hi) {
        this.array = array;  this.low = low;  this.hi = hi;
    }

    protected Integer compute() {
        …
    }
}
```

Putting this into actual Java

```
int sum(int[] array, int low, int hi) {
  //base case

  int left = fork sum(array, low, (hi+low)/2);
  int right = sum(array, (hi+low)/2, hi);
  join;
  return left + right;
}
```

```
...

protected Integer compute() {
  if(hi – low < THRESHOLD)
    return array[low];
  Sum left = new Sum(array, low, (hi+low)/2);
  Sum right = new Sum(array, (hi+low)/2, hi);
  left.fork();
  return right.compute() + left.join();
}
```

Only recurse to create enough strands

fork for only one half
run right half recursively, but before waiting for left half

Putting this into actual Java: Starting off

```
sum = ForkJoinPool.commonPool().invoke(new Sum(array, 0, array.length));
```

Putting it all together

```
class Sum extends RecursiveTask<Integer> {
    //attributes and constructor

    protected Integer compute(){
        if(hi – low < THRESHOLD) return simpleSum(array, low, hi);
        Sum left = new Sum(array, low, (hi+low)/2);
        Sum right = new Sum(array, (hi+low)/2, hi);
        left.fork();
        return right.compute() + left.join();
    }
}

sum = ForkJoinPool.commonPool().invoke(new Sum(array, 0, array.length));
```

Putting it all together

```
class Sum extends RecursiveTask<Integer> {
    //attributes and constructor

    protected Integer compute(){
        if(hi – low < THRESHOLD) return simpleSum(array, low, hi);
        Sum left = new Sum(array, low, (hi+low)/2);
        Sum right = new Sum(array, (hi+low)/2, hi);
        left.fork();
        return right.compute() + left.join();
    }
}
```

What if I wanted the minimum instead of the sum?

```
sum = ForkJoinPool.commonPool().invoke(new Sum(array, 0, array.length));
```

Putting it all together

```
class Sum extends RecursiveTask<Integer> {
    //attributes and constructor

    protected Integer compute(){
        if(hi – low < THRESHOLD) return simpleSum(array, low, hi);
        Sum left = new Sum(array, low, (hi+low)/2);
        Sum right = new Sum(array, (hi+low)/2, hi);
        left.fork();
        return right.compute() + left.join();
    }
}
```

What if I wanted the minimum **and** its index?

```
sum = ForkJoinPool.commonPool().invoke(new Sum(array, 0, array.length));
```

Reductions

- A reduction takes a collection of values (e.g. an array) and combined them into one value
- Many useful reductions can be efficiently implemented using this framework
  - Sum, product, or combination using a boolean operator (and, or, xor)
  - Minimum or maximum (or minIndex/maxIndex)
  - Building a histogram (record of how many times each value occurs)
  - Counting number of values meeting a condition
  - IndexOf or lastIndexOf
  - Finding the longest sequence of matching values