Analysis of Algorithms

• Chapel material
  – Assign basic tutorial
  – Teach forall & cobegin (also algorithmic notation)

• Projects
  – Partition integers
  – BubbleSort
  – MergeSort
  – Nearest Neighbors
Algorithms Project: List Partition

• Partition a list to two equal-summing halves.
• Brute-force algorithm (don't know P vs NP yet)
• Questions:
  – What are longest lists you can test?
  – What about in parallel?
• Trick: enumerate possibilities and use forall
Algorithms Project: BubbleSort

• Instead of left-to-right, test all pairs in two steps!
• Two nested forall loops (in sequence) inside a for loop
Algorithms Project: BubbleSort

for i in 0..n-1 {
    forall k in 0..n/2
        //compare 2k to 2k+1 (maybe swap)
    forall k in 0..n/2-1
        //compare 2k+1 to 2k+2 (maybe swap)
}
for i in 0..n-1 {
    forall k in 0..n/2
        //compare 2k to 2k+1 (maybe swap)
    forall k in 0..n/2-1
        //compare 2k+1 to 2k+2 (maybe swap)
}

\lim_{p \to n} T(n,p) = O(n)
Algorithms Project: MergeSort

Parallel divide-and-conquer: use cobegin

\[
\begin{array}{cccccccccc}
12 & 8 & 5 & 15 & 7 & 4 & 4 & 0 & 16 & 7 & 1 & 9 \\
\end{array}
\]
Algorithms Project: MergeSort

Parallel divide-and-conquer: use cobegin

12 8 5 15 7 4 4 0 16 7 1 9

4 5 7 8 12 15

0 1 4 7 9 16
Algorithms Project: MergeSort

Parallel divide-and-conquer: use cobegin

12  8  5  15  7  4  4  0  16  7  1  9

4  5  7  8  12  15

0  1  4  7  9  16

0  1  4  4  5  7  7  8  9  12  15  16
Algorithms Project: Nearest Neighbors

• Find closest pair of (2-D) points.
• Two algorithms:
  – Brute Force
    • (use a forall like bubbleSort)
  – Divide-and-Conquer
    • (use cobegin)
    • A bit tricky
• Value of parallelism: much easier to program the brute-force method