Quicksort

$O(n^2)$ worst case, $O(n \log n)$ average

Algorithm
- Partition to "left, pivot, right"
- Sort left and right with quicksort.

First try:
- Do both sub sorts in parallel.
- Pivot takes $O(n)$
- $\log n$ rounds
- Time bounded below by $n$ (more like $n \log n$)
- Time-processor product bounded below by $n^2$

Pivot in $O(1)$?
- If so, time bounded below by $\log n$
- Only on a PRAM!
CRCW PRAM algorithm

Arbitrary model CRCW
Algorithm builds a binary tree
- pivot is root
- less is left, greater is right
- sorted is inorder traversal
- sequential moves data so tree is inorder in array

CRCW algorithm ... build tree!

- root - common
  - parent[1..n] (p[i])
- leftchild[1..n] (lc[i])
- rightchild[1..n] (rc[i])
algorithm

procedure build_tree (a[1...n]) {
    foreach process i {
        root <- i; /* CW step */
        p[i] <- root;  lc[i] <- rc[i] <- n+1;
    }
    if i != root {
        repeat until all processes terminated {
            if a[i] < a[p[i]] or
                (a[i] = a[p[i]] and i < p[i]) {
            lc[p[i]] <- i; /* CW step */
            if lc[p[i]] = i { exit; }
            p[i] <- lc[p[i]];
        } else {
            rc[p[i]] <- i; /* CW step */
            if rc[p[i]] = i { exit; }
            p[i] <- rc[p[i]];
        }
    }
}
Final details ....

- number the nodes ($O(\log n)$ on CRCW)
- move into position ($O(1)$)

Quicksort $A[1..n]$:
- build_tree ($A[1..n]$) yeilding $p[]$, $rc[]$, $lc[]$
- number_nodes ($A$, $p$, $rc$, $lc$)
- move to proper location
Quicksort on "practical architectures"

Shared Memory version
- n numbers, p processors
- initial n/p per processor

Quicksort (A, start, stop, p processors (pstart, pstop))
- Processor pstart selects pivot (usual methods)
- Broadcasts pivot to all processors in pstart->pstop
- Each processor does local partition
- Do 2 prefix sums on number < pivot and >= pivot
  - keep prefix & total sums. S is set < pivot
- Using result of Prefix sums
  - New <- reordered numbers (barrier)
  - A <- new (barrier)
- Recurse: (Sub Quicksorts in parallel)
  - np <- ceil(|S|p/n+.5)
  - Quicksort (A, start, start+|S|-1, np, (pstart, pstart+np-1))
  - Quicksort (A, start+|S|, stop, p-np, (pstart+np-1, pstop))
Quicksort on a message passing machine?

Do the shared memory algorithm in a distributed fashion!

- pivot -> broadcast
- prefix sums ... doable
- movement ... All to All comm
- Similar ...

Quicksort on a hypercube? (12.1.3 Hyperquicksort)

- each PE sorts the sub array assigned to that PE
  - for i = d downto 1
    - for each i-cube:
      - root of the i-cube broadcasts its median to all in the i-cube, to serve as pivot
      - consider the two (i-1)-subcubes of this i-cube
        - each pair of partners in the (i-1)-subcubes exchanges data:
          - low-numbered PE gives its partner its data larger than pivot
          - high-numbered PE gives its partner its data smaller than pivot

Issues?

- final placement of elements, e.g. sort is sub step in most algorithms
  - parallel prefix needed to calculate final location
  - final movement?
Batcher’s bitonic mergesort

- bitonic sequence
  - at most two "changes of direction"
  - and remains so for any rotation of the sequence
- originally designed for "hardware" (circuits), yet another "parallel model"
  - gates -- the computation elements
  - width of circuit -- parallelism, time -- parallel time
- Bitonic merge
  - compare corresponding items in each half
  - results in two bitonic sequences ready to be merged, all numbers in one greater than other
- Full bitonic mergesort algorithm, p == n
  - for i = 1 to log p
    - perform a bitonic merge on PE groups of size $2^i$ in alternating directions

- Bitonic merge sort, p < n?
  - a) Local sort
  - b) "exchange in merge" sends up to n/p elements neighbor and saves lower or upper half