Sorting

(PSC2 §1.8 )
Quicksort (Hoare 1962)

input: \( x \): vector of length \( n \), interval \([a, b]\) with \( 0 \leq a \leq b < n \).

output: \( x \) is sorted with \( x_i \leq x_j \) for all \( i, j \) with \( a \leq i \leq j \leq b \).

function \textsc{Quicksort}(x, a, b)

\[
\begin{align*}
    i & := \text{Split}(x, a, b); \\
    \text{if } i - 1 > a & \text{ then} \\
       & \text{Quicksort}(x, a, i - 1); \\
    \text{if } i + 1 < b & \text{ then} \\
       & \text{Quicksort}(x, i + 1, b);
\end{align*}
\]
Index $r$, with $0 \leq r < n$, is a splitter if

\[ x_i < x_r \text{ for } i < r, \]
\[ x_i \geq x_r \text{ for } i \geq r. \]

The vector $\mathbf{x}$ of length $n = 10$ has one splitter, $i = 5$, with value $x_5 = 8$:

\[
\begin{array}{cccccccccc}
  x_i = & 3 & 6 & 2 & 7 & 5 & 8 & 13 & 14 & 10 & 11 \\
  i = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]
Splitting a vector based on a random pivot

**function** `SPLIT(x, a, b)`

1. pick `piv`, with `a ≤ piv ≤ b`;
2. `val := x_{piv}`;
3. `swap(x_{piv}, x_{b})`;
4. `i := a`;
5. for `j = a to b − 1` do
   - if `x_j < val` then
     - `swap(x_i, x_j)`;
     - `i := i + 1`;
   - `swap(x_i, x_{b})`;
6. return `i`;

**Loop invariant:** at the start of iteration `j`

- `x[a, i − 1] < val`
- `x[i, j − 1] ≥ val`
- `x[j, b − 1]` have not been processed yet.
Parallel regular sample sort (Shi and Schaeffer 1992)
BSP algorithm: supersteps 0, 1

input: \(x: \) vector of length \(n\), \(n \mod p^2 = 0\), \(x_i \neq x_j\) for all \(i \neq j\).
\[\text{distr}(x) = \phi, \text{ with } \phi(i) = i \text{ div } b, \text{ for } 0 \leq i < n, \text{ where } b = n/p.\]

output: \(x\) is sorted with \(x_i \leq x_j\) for all \(i, j\) with \(0 \leq i \leq j < n\).
\(x\) is block distributed with variable block size \(b_s \leq 2b\).

\{ Sort the local block and create samples \} \quad \rightarrow \quad \text{Superstep (0)}
Quicksort\((x, sb, (s + 1)b - 1)\);
\text{for } i := 0 \text{ to } p - 1 \text{ do }
\quad sample_s[i] := x[sb + i \cdot \frac{n}{p^2}];

\{ Broadcast the samples \} \quad \rightarrow \quad \text{Superstep (1)}
\text{for } t := 0 \text{ to } p - 1 \text{ do }
\quad \text{put } sample_s \text{ in } P(t);
\ldots
Cost analysis: supersteps 0, 1

- Assumption: \( n \mod p = 0 \) so each processor has a block of exactly \( \frac{n}{p} \) array elements.
- Assumption: \( \frac{n}{p} \mod p = 0 \) so each processor has \( p \) subblocks with exactly \( \frac{n}{p^2} \) array elements.
- Sorting an array of length \( \frac{n}{p} \) costs
  \[
  T_{(0)} = \frac{n}{p} \log_2 \frac{n}{p} + l
  \]
- Broadcasting \( p \) samples to \( p - 1 \) other processors costs
  \[
  T_{(1)} = p(p - 1)g + l.
  \]
BSP algorithm: superstep 2

{ Concatenate and sort the samples }
for $t := 0$ to $p - 1$ do
  for $i := 0$ to $p - 1$ do
    \( sample[tp + i] := sample_t[i] \)
    \( start[t] := tp \)
  \( start[p] := p^2 \)
  Mergesort(\( sample, 0, p^2 - 1, start, p \))

{ Create splitters }
for $t := 0$ to $p - 1$ do
  \( splitval[t] := sample[tp] \)
  \( splitval[p] := \infty \)
Cost analysis: superstep 2

- The $p^2$ samples are already arranged as $p$ sorted parts, so we use a mergesort instead of quicksort.
- Mergesort repeatedly merges a pair of sorted parts, in $\lceil \log_2 p \rceil$ phases, each costing $p^2$ flops.
- Mergesort$(x, \text{start}, p)$ sorts the vector $x$ using the fact that the intervals $[\text{start}[t], \text{start}[t + 1] - 1]$ are already sorted, for $t = 0, \ldots, p - 1$.

\[ T(2) = p^2 \lceil \log_2 p \rceil + l. \]
BSP algorithm: superstep 3

\{ Split the local block and send its parts \}

\textbf{for} \quad t := 0 \quad \textbf{to} \quad p - 1 \quad \textbf{do}

\{ Contribution from \( P(s) \) to \( P(t) \) \}

\( X_{st} := \{ x_i : \; sb \leq i < (s + 1)b \land \)

\( \text{splitval}[t] \leq x_i < \text{splitval}[t + 1] \}; \)

put \( X_{st} \) in \( P(t) \);
BSP algorithm: superstep 4

\{ Concatenate the received parts \}
\[ X_s := \bigcup_{t=0}^{p-1} X_{ts}; \]

\{ Sort the local block \}
\[ start_s[0] := 0; \]
\[ \text{for } t := 1 \text{ to } p \text{ do} \]
\[ \quad start_s[t] := start_s[t - 1] + |X_{t-1,s}|; \]
\[ bs := start_s[p]; \]
\[ \text{Mergesort}(X_s, start_s, p); \]
Cost analysis: supersteps 3 and 4

- Superstep 3: each processor sends at most all its data ($b$ values), and receives at most $b_s$ data, so

$$T_{(3)} = \max_s \max(b, b_s)g + l.$$

- Superstep 4: $p$ operations to compute the starts.
- Mergesort repeatedly merges a pair of sorted parts, in $\lceil \log_2 p \rceil$ phases, each accessing at most all the $b_s$ local data, so

$$T_{(4)} = p + \max_s b_s \lceil \log_2 p \rceil + l.$$
Proof that $b_s \leq 2b$ (1)

- A **subblock** is a part of the locally sorted vector of length $\frac{n}{p^2}$ that starts with a sample.
- There are $p$ local subblocks.
- Consider a processor $P(s)$ with fixed $s$, containing $b_s$ output data.
- The local output block contains exactly $p$ samples, and hence exactly $p$ subblocks contributing a sample, in total contributing at most $p \cdot \frac{n}{p^2} = \frac{n}{p} = b$ data values.
- Example: subblock $(17,19,23)$ with sample 17 contributes to $P(2)$. 

Proof that $b_s \leq 2b$ (2)

- Thinking alert! Now consider contributions by a subblock that does not contribute a sample.
- Example: subblock (14,16,18) only contributes 18 to $P(2)$ but not the sample 14.
- Each processor $P(t)$ can contribute at most one such subblock to $P(s)$, because the subblock must have a value $< \text{splitval}[s]$ and a value $\geq \text{splitval}[s]$.
- Other subblocks of $P(t)$ will be completely to the left or right of that subblock and cannot contribute.
- Total contribution from all processors is at most $p \cdot \frac{n}{p^2} = \frac{n}{p} = b$ data values.
- Total size is $b_s \leq 2b$. 
Total cost

\[ T_{\text{samplesort}} = \frac{n}{p} \log_2 \frac{n}{p} + p^2 \lceil \log_2 p \rceil + \frac{2n}{p} \cdot \lceil \log_2 p \rceil \]
\[ + \left( p(p - 1) + 2 \frac{n}{p} \right) g + 5l \]
\[ \approx \frac{n}{p} \left( \log_2 n + \log_2 p \right) + p^2 \log_2 p + \left( p^2 + 2 \frac{n}{p} \right) g + 5l \]

▶ Efficient if \( p^2 \leq \frac{n}{p} \), i.e. \( p \leq n^{1/3} \).
Parallel samplesort uses samples at regular intervals to split local data into \( p \) subblocks.

The resulting imbalance of parallel samplesort in memory requirements is a factor of at most 2.

Further oversampling can reduce this factor.

A BSP samplesort algorithm can be formulated with 3 computation supersteps and 2 communication supersteps.