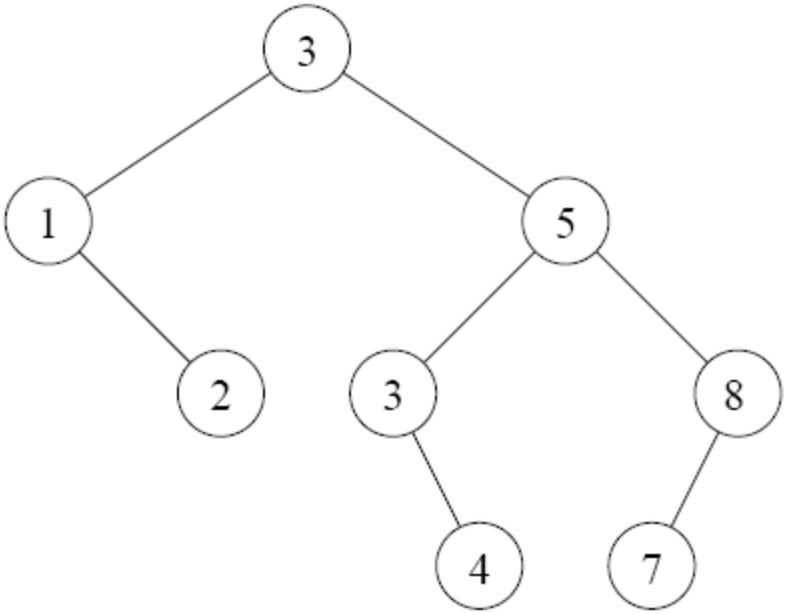
# Parallelizing Quicksort

- Lets start with recursive decomposition - the list is partitioned serially and each of the subproblems is handled by a different processor.
- The time for this algorithm is lower-bounded by $\Omega(n)$!
- Can we parallelize the partitioning step - in particular, if we can use $n$ processors to partition a list of length $n$ around a pivot in $O(1)$ time, we have a winner.
- This is difficult to do on real machines, though.

# Parallelizing Quicksort: PRAM Formulation

- We assume a CRCW (concurrent read, concurrent write) PRAM with concurrent writes resulting in an arbitrary write succeeding.
- The formulation works by creating pools of processors. Every processor is assigned to the same pool initially and has one element.
- Each processor attempts to write its element to a common location (for the pool).
- Each processor tries to read back the location. If the value read back is greater than the processor's value, it assigns itself to the `left' pool, else, it assigns itself to the `right' pool.
- Each pool performs this operation recursively.
- Note that the algorithm generates a tree of pivots. The depth of the tree is the expected parallel runtime. The average value is $O(\log\ n)$.

# Parallelizing Quicksort: PRAM Formulation




A binary tree generated by the execution of the quicksort algorithm. Each level of the tree represents a different array-partitioning iteration. If pivot selection is optimal, then the height of the tree is $\Theta(\log\ n)$, which is also the number of iterations.

# Parallelizing Quicksort: PRAM Formulation

(a)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 33 | 21 | 13 | 54 | 82 | 33 | 40 | 72 |

(b) root = 4

(c)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| *leftchild* | | | | 1 | | | | |
| *rightchild* | | | | 5 | | | | |

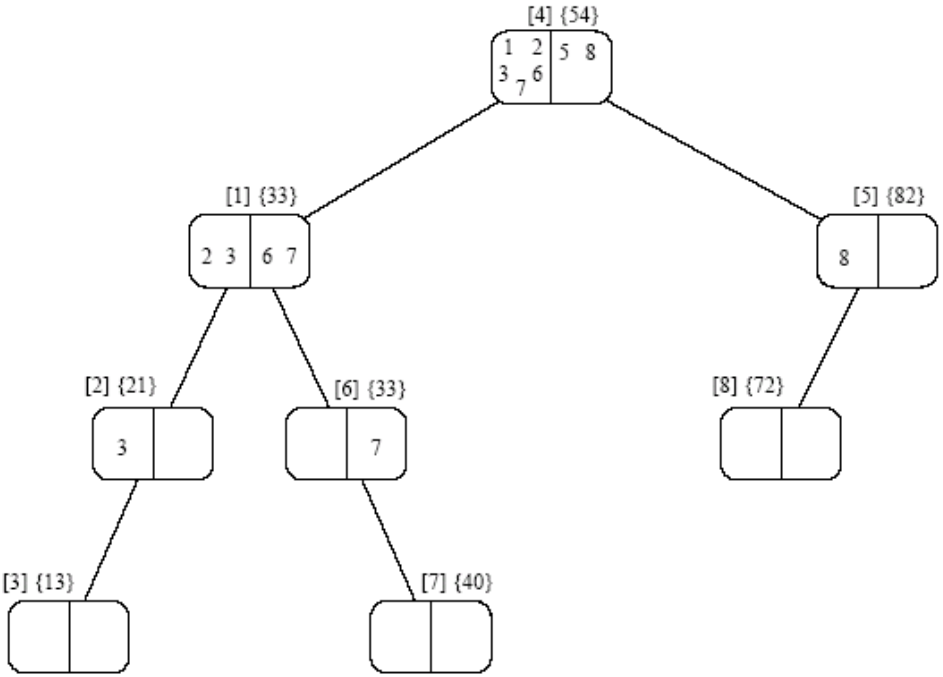
(d)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| *leftchild* | 2 | | | 1 | 8 | | | |
| *rightchild* | 6 | | | 5 | | | | |

(e)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| *leftchild* | 2 | 3 | | 1 | 8 | | | |
| *rightchild* | 6 | | | 5 | | 7 | | |

(f)

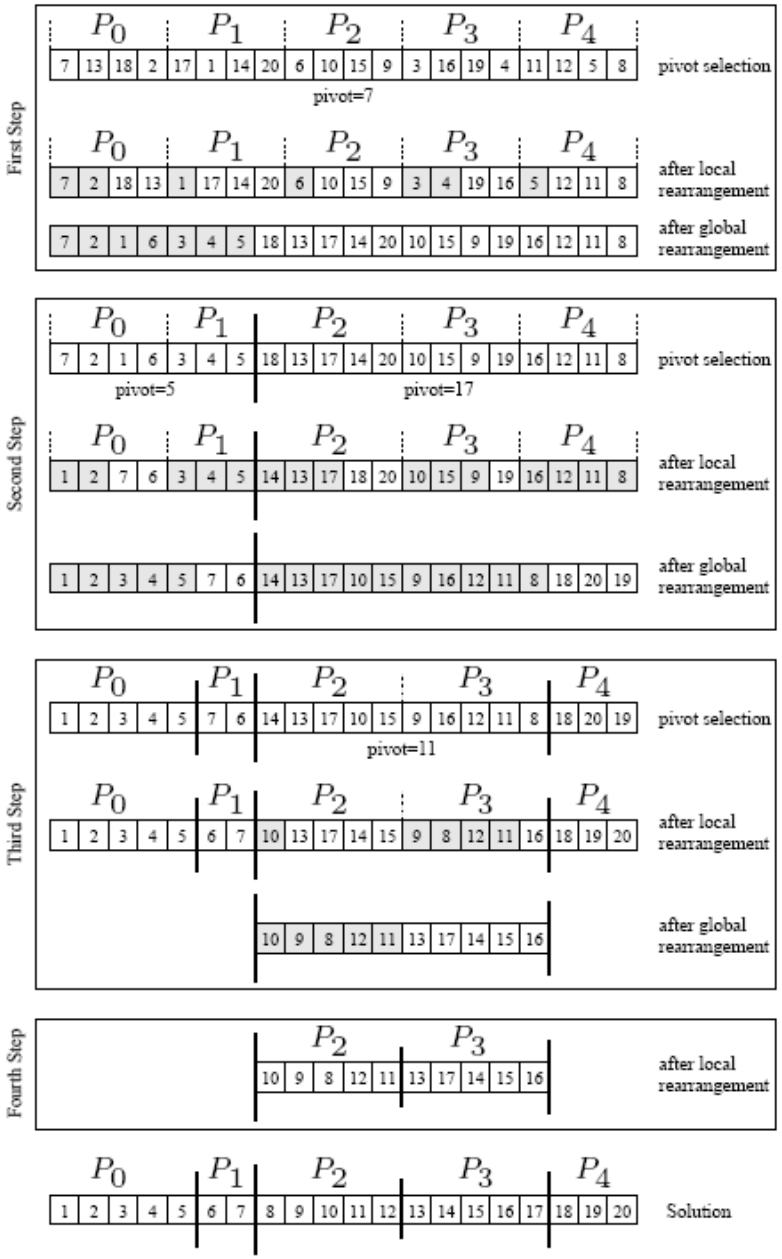



The execution of the PRAM algorithm on the array shown in (a).

# Parallelizing Quicksort: Shared Address Space Formulation

- Consider a list of size $n$ equally divided across $p$ processors.
- A pivot is selected by one of the processors and made known to all processors.
- Each processor partitions its list into two, say $L_i$ and $U_i$, based on the selected pivot.
- All of the $L_i$ lists are merged and all of the $U_i$ lists are merged separately.
- The set of processors is partitioned into two (in proportion of the size of lists $L$ and $U$). The process is recursively applied to each of the lists.

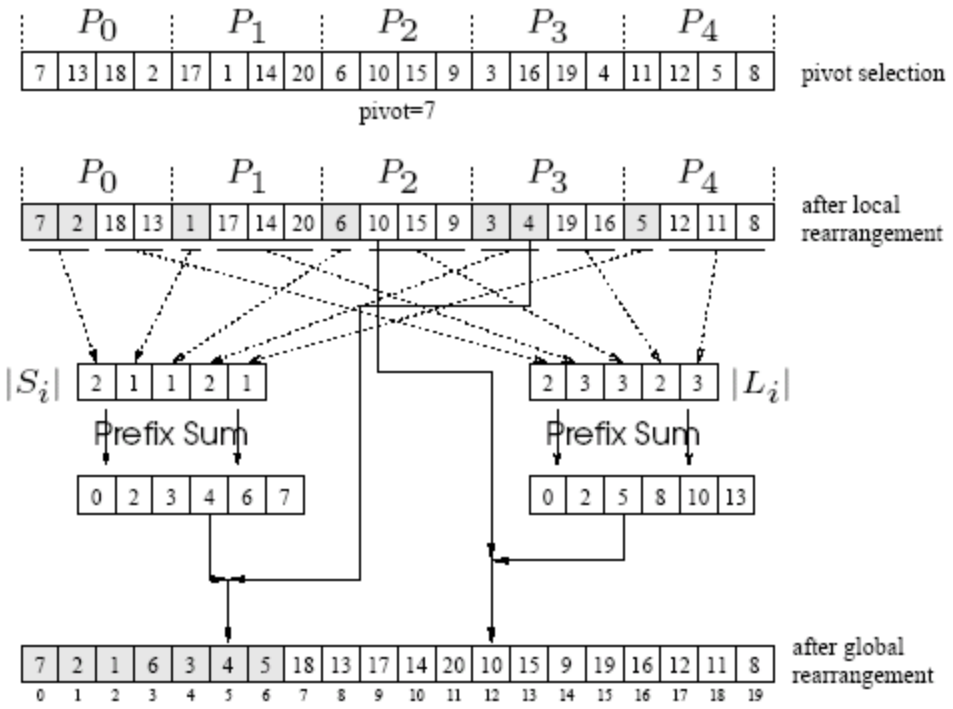
# Shared Address Space Formulation



# Parallelizing Quicksort: Shared Address Space Formulation

- The only thing we have not described is the global reorganization (merging) of local lists to form $L$ and $U$.
- The problem is one of determining the right location for each element in the merged list.
- Each processor computes the number of elements locally less than and greater than pivot.
- It computes two sum-scans to determine the starting location for its elements in the merged $L$ and $U$ lists.
- Once it knows the starting locations, it can write its elements safely.

# Parallelizing Quicksort: Shared Address Space Formulation




Efficient global rearrangement of the array.

# Parallelizing Quicksort: Shared Address Space Formulation

- The parallel time depends on the split and merge time, and the quality of the pivot.
- The latter is an issue independent of parallelism, so we focus on the first aspect, assuming ideal pivot selection.
- The algorithm executes in four steps: (i) determine and broadcast the pivot; (ii) locally rearrange the array assigned to each process; (iii) determine the locations in the globally rearranged array that the local elements will go to; and (iv) perform the global rearrangement.
- The first step takes time $\Theta(\log p)$, the second, $\Theta(n/p)$ , the third, $\Theta(\log p)$ , and the fourth, $\Theta(n/p)$.
- The overall complexity of splitting an $n$-element array is $\Theta(n/p) + \Theta(\log p)$.

# Parallelizing Quicksort: Shared Address Space Formulation

- The process recurses until there are $p$ lists, at which point, the lists are sorted locally.
- Therefore, the total parallel time is:

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(\frac{n}{p}\log p\right) + \Theta(\log^2 p)}^{\text{array splits}}. \qquad (4)$$

- The corresponding isoefficiency is $\Theta(p\log^2 p)$ due to broadcast and scan operations.

# Parallelizing Quicksort: Message Passing Formulation

- A simple message passing formulation is based on the recursive halving of the machine.
- Assume that each processor in the lower half of a $p$ processor ensemble is paired with a corresponding processor in the upper half.
- A designated processor selects and broadcasts the pivot.
- Each processor splits its local list into two lists, one less ($L_i$), and other greater ($U_i$) than the pivot.
- A processor in the low half of the machine sends its list $U_i$ to the paired processor in the other half. The paired processor sends its list $L_i$.
- It is easy to see that after this step, all elements less than the pivot are in the low half of the machine and all elements greater than the pivot are in the high half.

# Parallelizing Quicksort: Message Passing Formulation

- The above process is recursed until each processor has its own local list, which is sorted locally.
- The time for a single reorganization is $\Theta(\log\ p)$ for broadcasting the pivot element, $\Theta(n/p)$ for splitting the locally assigned portion of the array, $\Theta(n/p)$ for exchange and local reorganization.
- We note that this time is identical to that of the corresponding shared address space formulation.
- It is important to remember that the reorganization of elements is a bandwidth sensitive operation.

# Assignment

Q.1)What is parallelizing Quicksort?

Q.2)How message passing formulation is possible in parallellizing quicksort?