Parallel Algorithms III

- Topics: graph and sort algorithms

Graph Algorithms

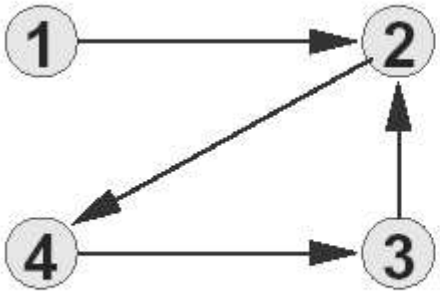
$G = (V, E)$: a directed graph, $V = \{1, \ldots, N\}$

The *adjacency matrix* $A = (a_{ij})$ of G is

$$a_{ij} = \begin{cases} 1 & \text{if either } (i,j) \in E \text{ or } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

The transitive closure of G is $G^* = (V, E^*)$,

$$E^* = \{(i,j) \mid j \text{ is reachable from } i \text{ in } G\}.$$



$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \qquad A^* = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

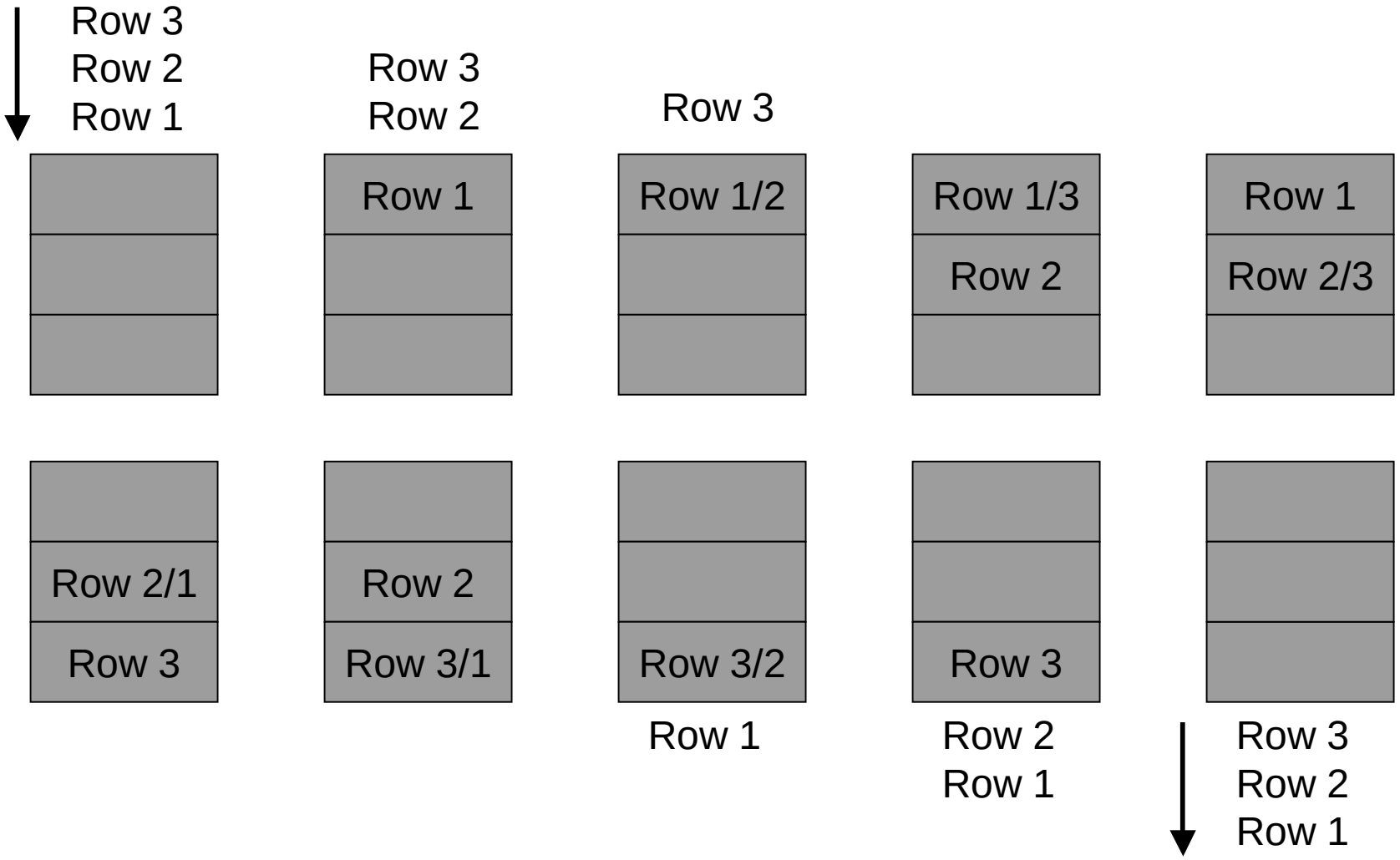
Floyd Warshall Algorithm

$A^{(k)} =_{\text{def}} (a_{ij}^{(k)})$, where for each $k, 0 \leq k \leq N$, $a_{ij}^{(k)} = 1$ if j is reachable from i *passing through* only nodes $\leq k$ and 0 otherwise.

Then $A^{(N)} = A^*$, $A^{(0)} = A$, and for all $k \geq 1$,

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} \vee \left(a_{ik}^{(k-1)} \wedge a_{kj}^{(k-1)} \right).$$

Implementation on 2d Processor Array



Algorithm Implementation

- Diagonal elements of the processor array can broadcast to the entire row in one time step (if this assumption is not made, inputs will have to be staggered)
- A row sifts down until it finds an empty row – it sifts down again after all other rows have passed over it
- When a row passes over the 1st row, the value of a_{i1} is broadcast to the entire row – a_{ij} is set to 1 if $a_{i1} = a_{1j} = 1$ – in other words, the row is now the ith row of $A^{(1)}$
- By the time the kth row finds its empty slot, it has already become the kth row of $A^{(k-1)}$

Algorithm Implementation

- When the i^{th} row starts moving again, it travels over rows a_k ($k > i$) and gets updated depending on whether there is a path from i to j via vertices < k (and including k)

Shortest Paths

- Given a graph and edges with weights, compute the weight of the shortest path between pairs of vertices

- Can the transitive closure algorithm be applied here?

Shortest Paths Algorithm

$D = (d_{ij})$: the distance matrix, where $d_{ij} = \infty$ if there is no edge from i to j
Define $D^{(k)} = (d_{ij}^{(k)})$, where $d_{ij}^{(k)}$ is the length of the shortest path from i to j that passes through only nodes $\leq k$.

Then we have only to compute $D^{(N)}$. Note $D^{(0)} = D$ and for all $k \geq 1$,

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}).$$

The above equation is very similar to that in transitive closure

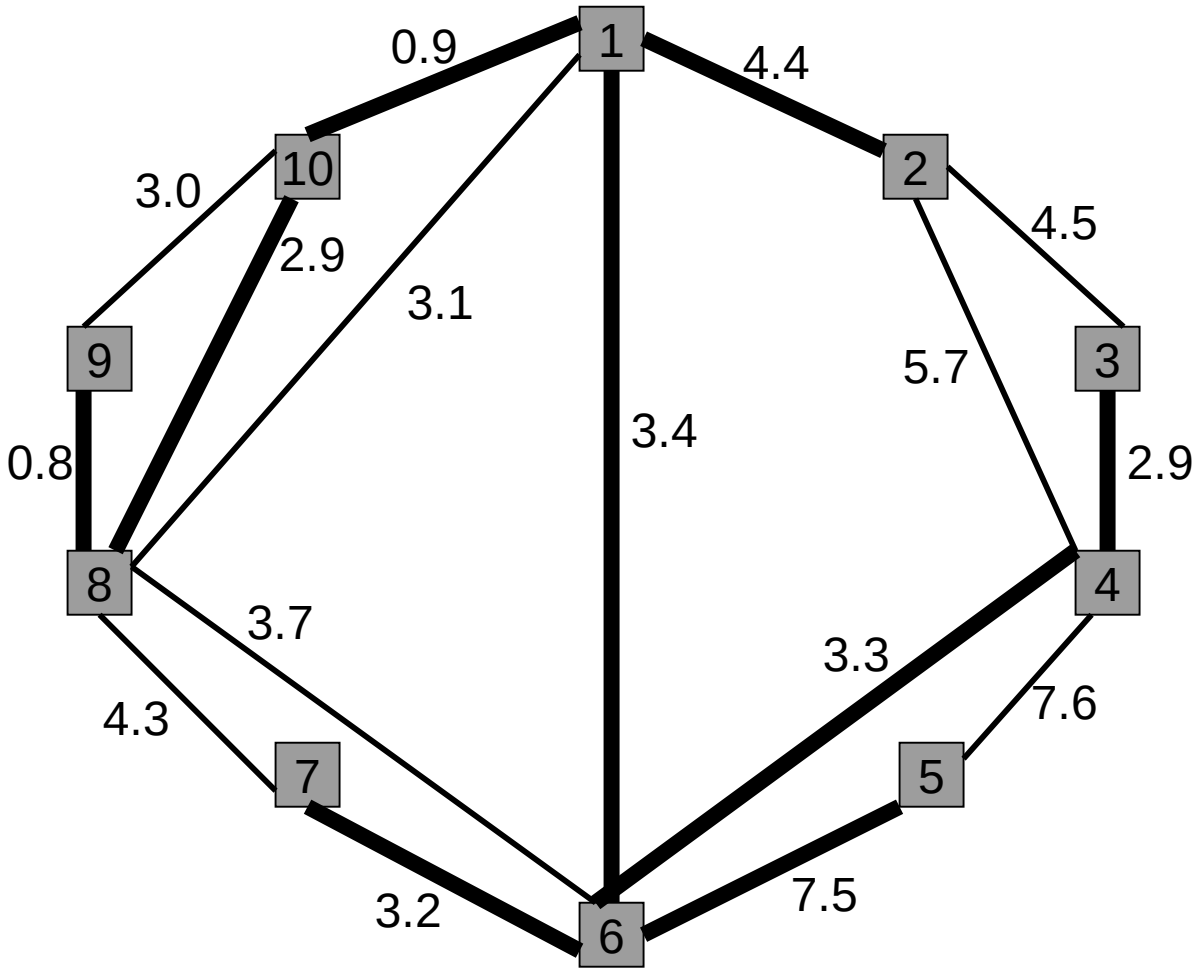
Minimum Weight Spanning Tree

- Given an undirected connected graph and edges with weights, compute the spanning tree with minimum sum of weights

- The following theorem simplifies the algorithm design:

Theorem With pairwise distinct weights, for every i, j, the edge (i, j) is in the MWST if and only if every i-j path of length ≥ 2 contains an edge with weight $> w_{ij}$.

Example



Theorem Proof

Proof ($\Leftarrow$) Let $T, (i,j) \notin T$, be a spanning tree. The path in T from i to j is of length > 1, and thus, contains an edge, u, whose weight is $> w_{ij}$. Replacing u by (i,j) yields a spanning tree with a smaller weight.

($\Rightarrow$) Let $T, (i,j) \in T$, be given. Suppose there is a path p of length ≥ 2 from i to j containing only edges with weight $< w_{ij}$. Eliminating (i,j) from T splits T into trees T_1 and T_2. Take an edge e on p connnecting T_1 and T_2 and replace (i,j) by e. Then its weight is smaller than that of T. ∎

Parallel Algorithm Implementation

- Employ shortest-paths algorithm again – the weight of a path is now the heaviest edge on the path

For each i, j, k, let $w_{ij}^{(k)}$ be the minimum of all maximum weights appearing in some i-j path in G that pass through only nodes $\leq k$. Then $w_{ij}^{(0)} = w_{ij}$ and

$$w_{ij}^{(k)} = \min(w_{ij}^{(k-1)}, \max(w_{ik}^{(k-1)}, w_{kj}^{(k-1)})).$$

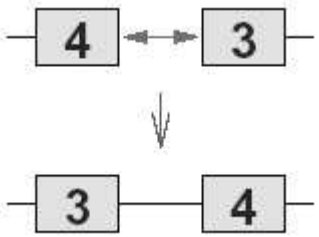
So, we can compute $w_{ij}^{(N)}$ by a similar method.

Now (i, j) is in the maximum weight spanning tree if and only if $w_{ij} < w_{ij}^{(N)}$.

Sorting with Comparison Exchange

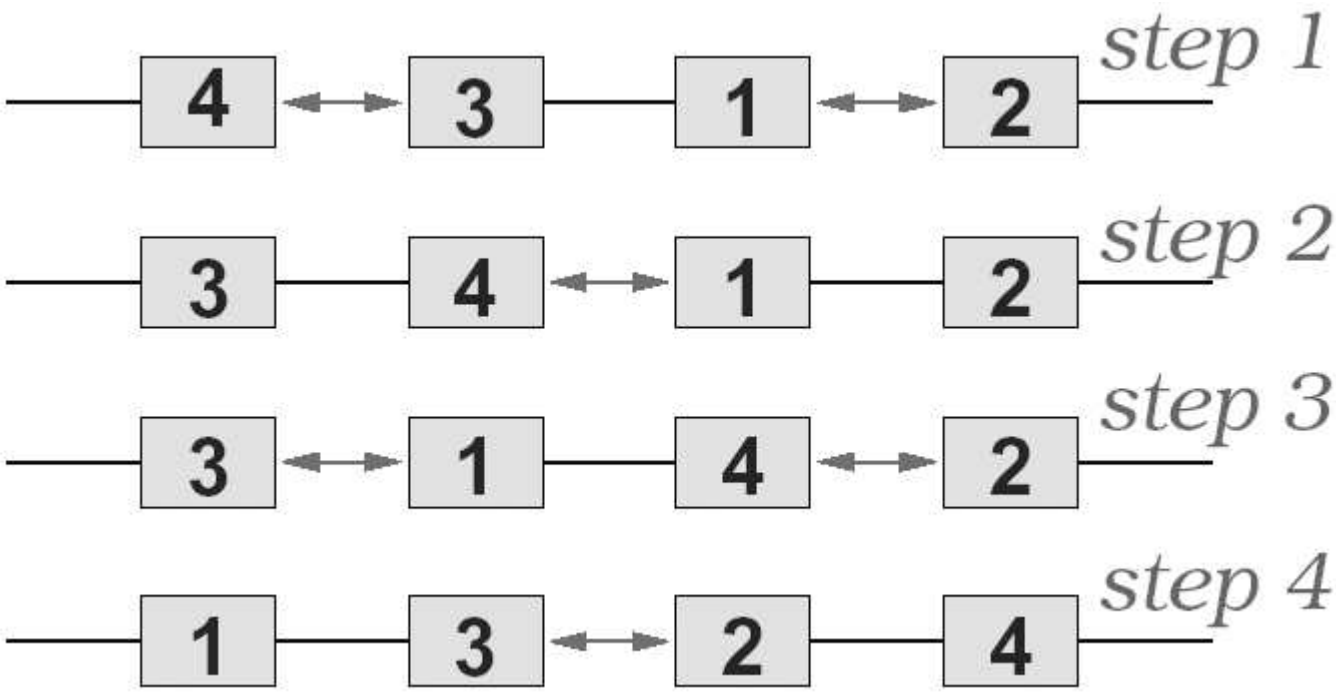
- Earlier sort implementations assumed processors that could compare inputs and local storage, and generate an output in a single time step

- The next algorithm assumes comparison-exchange processors: two neighboring processors I and J (I < J) show their numbers to each other and I keeps the smaller number and J the larger



Odd-Even Sort

- N numbers can be sorted on an N-cell linear array in O(N) time: the processors alternate operations with their neighbors



The 0-1 Sorting Lemma

If a comparison-exchange algorithm sorts input sets consisting solely of 0's and 1's, then it sorts all input sets of arbitrary values

Proof Let an o.c.e. algorithm $\mathcal{A}$ for input size N be given. Suppose that $\mathcal{A}$ works on all 0-1 inputs. Assume that $\mathcal{A}$ fails on an input $[x_1, \ldots, x_N]$ with $[y_1, \ldots, y_N]$ be the output. Then $y_1 \leq \cdots \leq y_m > y_{m+1}$ for some m. Define mapping F by $F(x) = 0$ if $x < y_m$ and 1 otherwise. Since F preserves the order $\leq$, the output of $\mathcal{A}$ on $[F(x_1), \ldots, F(x_N)]$ is $[F(y_1), \ldots, F(y_N)]$, and is of the form $[\ldots, 1, 0, \ldots]$ because of $y_m > y_{m+1}$. This is a contradiction.

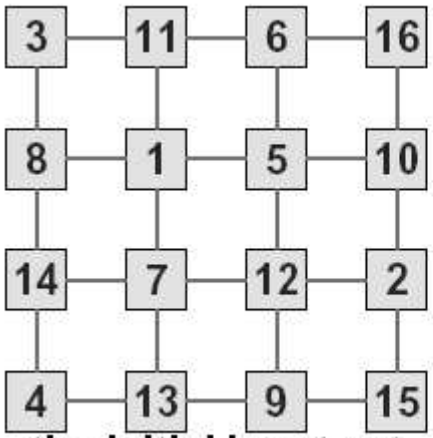
Shearsort

- A sorting algorithm on an N-cell square matrix that improves execution time to O(sqrt(N) logN)

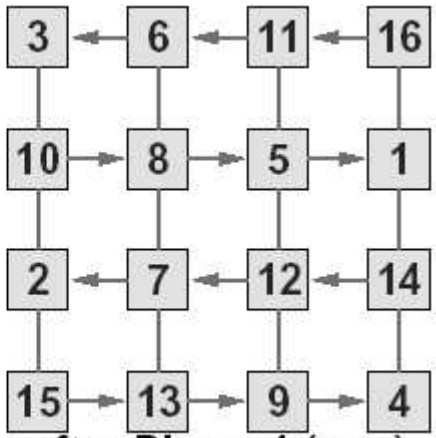
- Algorithm steps:
  Odd phase: sort each row with odd-even sort (all odd rows are sorted left to right and all even rows are sorted right to left)
  Even phase: sort each column with odd-even sort
  Repeat

- Each odd and even phase takes O(sqrt(N)) steps – the input is guaranteed to be sorted in O(logN) steps

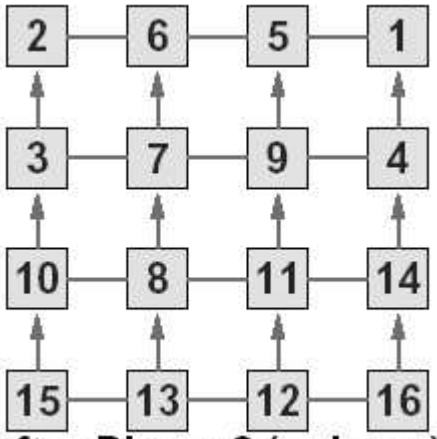
Example



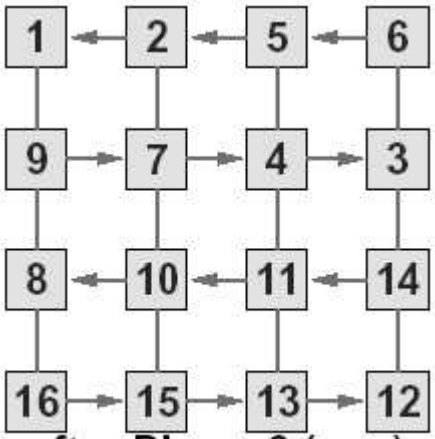
the initial input set



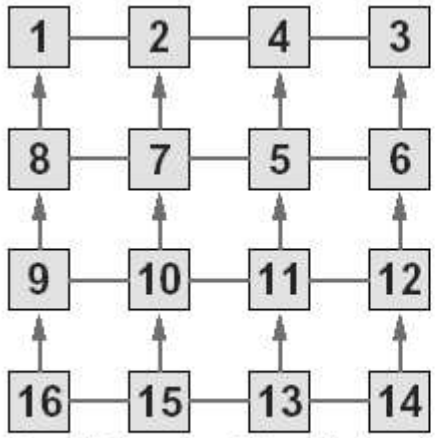
after Phase 1 (row)



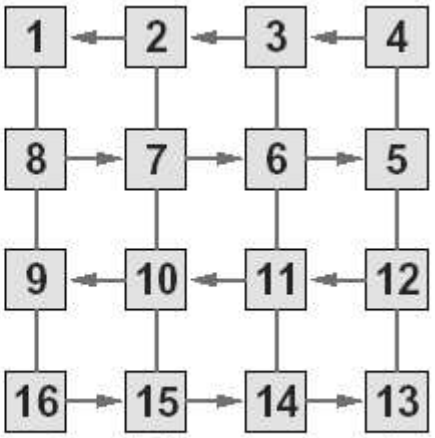
after Phase 2 (column)



after Phase 3 (row)



after Phase 4 (column)



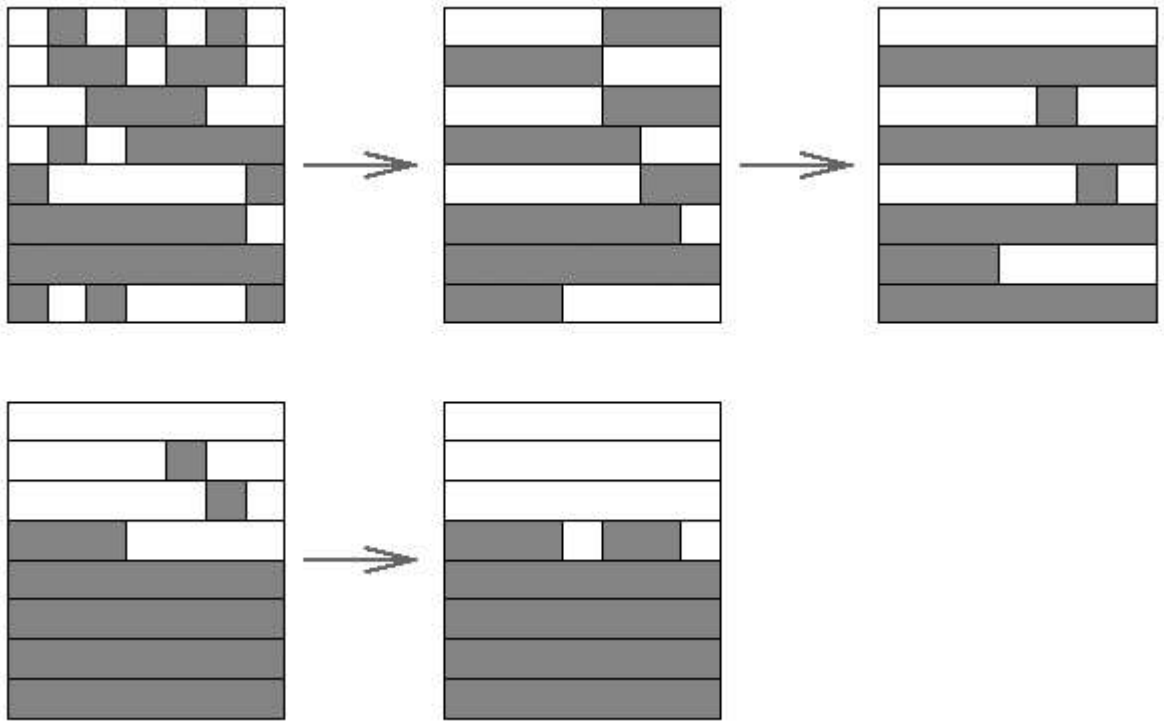
after Phase 5 (row)

Complexity Proof

- How do we prove that the algorithm completes in O(logN) phases? (each phase takes O(sqrt(N)) steps)
- Assume input set of 0s and 1s
- There are three types of rows: all 0s, all 1s, and mixed entries – we will show that after every phase, the number of mixed entry rows reduces by half
- The column sort phase is broken into the smaller steps below: move 0 rows to the top and 1 rows to the bottom; the mixed rows are paired up and sorted within pairs; repeat these small steps until the column is sorted

Example

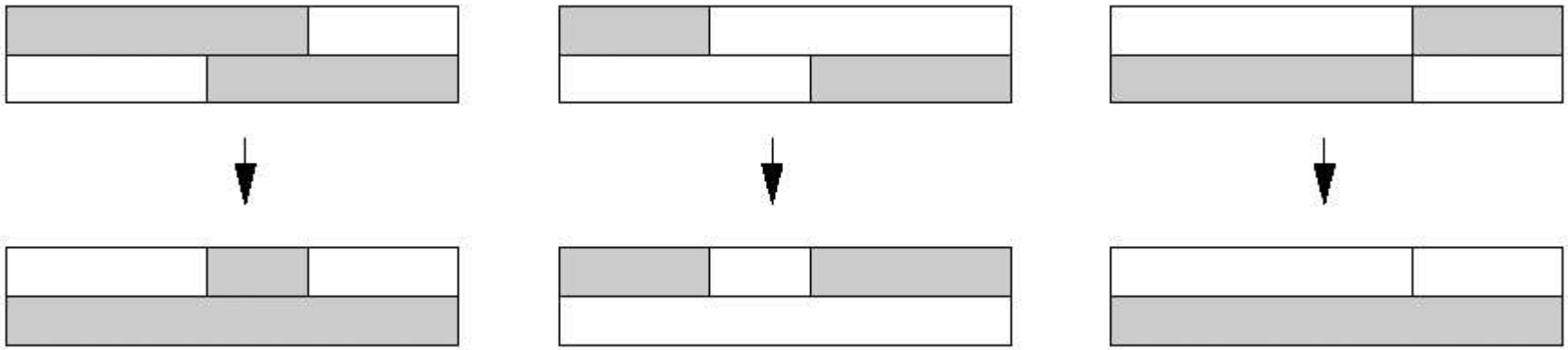
- The modified algorithm will behave as shown below: white depicts 0s and blue depicts 1s



Proof

- If there are N mixed rows, we are guaranteed to have fewer than N/2 mixed rows after the first step of the column sort (subsequent steps of the column sort may not produce fewer mixed rows as the rows are not sorted)

Each pair of mixed rows produces at least one pure row when sorted



Title

- Bullet