GRAPH SEARCH. There are many ways to conduct tree traversals (such as In-Order, Pre-Order, and Post-Order). These algorithms take linear time. Depth First Search (DFS) and Breadth First Search (BFS) can be used to conduct generic graph searches. These algorithms take $O(|V| + |E|)$ time.

PARALLEL ALGORITHMS. The model we used was the PRAM (Parallel Random Access Machine). Processors communicate by writing into and reading from memory cells that are accessible to all. Depending on how read and write conflicts are resolved, there are variants of the PRAM. In an Exclusive Read Exclusive Write (EREW) PRAM, no concurrent reads or concurrent writes are permitted. In a Concurrent Read Exclusive Write (CREW) PRAM, concurrent reads are permitted but concurrent writes are prohibited. In a Concurrent Read Concurrent Write (CRCW) PRAM both concurrent reads and concurrent writes are allowed. Concurrent writes can be resolved in many ways. In a Common CRCW PRAM, concurrent writes are allowed only if the conflicting processors have the same message to write (into the same cell at the same time). In an Arbitrary CRCW PRAM, an arbitrary processor gets to write in cases of conflicts. In a Priority CRCW PRAM, write conflicts are resolved on the basis of priorities (assigned to the processors at the beginning).

We presented a Common CRCW PRAM algorithm for finding the Boolean AND of $n$ given bits in $O(1)$ time. We used $n$ processors. As a corollary we gave an algorithm for finding the minimum (or maximum) of $n$ given arbitrary real numbers in $O(1)$ time using $n^2$ Common CRCW PRAM processors. It was shown that the maximum or minimum of $n$ integers in the range $[1, n^c]$ ($c$ being any constant) can be computed in $O(1)$ time using $n$ Common CRCW PRAM processors.

We also discussed a CREW PRAM algorithm for the prefix computation problem. This algorithm uses $\frac{n}{\log n}$ processors and runs in $O(\log n)$ time on any input of $n$ elements. (For the prefix computation problem the input is a sequence of elements from some domain $\Sigma$: $k_1, k_2, \ldots, k_n$ and the output is another sequence: $k_1, k_1 \oplus k_2, \ldots, k_1 \oplus k_2 \oplus k_3 \oplus \cdots \oplus k_n$, where $\oplus$ is any binary associative and unit-time computable operation on $\Sigma$.) As an application of prefix computation, we proved that sorting of $n$ elements can be done in $O(\log n)$ time using $\frac{n^2}{\log n}$ CREW PRAM processors. We discussed the odd-even merge algorithm and proved that we can merge two sorted sequences of length $n$ each in $O(\log n)$ time using $n$ EREW PRAM processors. As a result, we can sort a given sequence of $n$ keys in $O(\log^2 n)$ time using $n$ EREW PRAM processors.

INTRACTABLE PROBLEMS. A problem $\pi_1$ is said to be polynomially reducible to another problem $\pi_2$ (denoted as $\pi_1 \leq \pi_2$) if the following statement holds: "If $\pi_2$ can be solved in deterministic polynomial time then $\pi_1$ can also be solved in deterministic polynomial time".

A problem $\pi$ is said to be $\mathcal{NP}$-hard if $\pi' \leq \pi$ for every $\pi' \in \mathcal{NP}$. Equivalently, a problem $\pi$ is $\mathcal{NP}$-hard if $\pi' \leq \pi$ where $\pi'$ is known to be $\mathcal{NP}$-hard. A problem $\pi$ is $\mathcal{NP}$-complete if $\pi$ is in $\mathcal{NP}$ and $\pi$ is $\mathcal{NP}$-hard.

The following are examples of $\mathcal{NP}$-complete problems: SAT, CLIQUE, Node Cover Decision Problem, 3SAT, Subset Sum, and Partition. We briefly summarized Cook’s theorem that states that SAT is $\mathcal{NP}$-complete. Using this theorem we showed that the following problems are also $\mathcal{NP}$-complete: CLIQUE and Node Cover Decision Problem. We also showed that the following problems are $\mathcal{NP}$-complete: 3SAT and Chromatic Number Decision Problem (also known as the graph coloring problem).