[Overall Information]

We had the third quiz of CSCI303 in today’s class. It is a twenty minutes quiz with four questions concerning Minimum Spanning Tree and Shortest Path problems.

After the quiz, we analyzed the running time of the Merge Sort algorithm by drawing a recursion tree. The running time of Merge Sort can also be easily calculated with the help of Master Theorem, which was introduced afterwards. We also looked at another divide and conquer algorithm for the problem of multiplying two integers. Its running time was calculated using the Master Theorem.
**[Announcements]**

The midterm is scheduled on March 22, the Wednesday right after the spring break. It will be an open-book, open-notes exam. The questions will be more difficult than quiz ones.

**[During the Lecture]**

1. We continued the running time analysis of Merge Sort. As was said at last class, we tried to guess a solution for the recurrence relation $T(n) = 2T(n/2) + cn$, and then prove the solution’s correctness by induction.

Since guessing is a bit different, we started looking at the approach to use a recursion tree. If the steps within the recursive function calls are not counted, the running time of the algorithm on a size $n$ array is $cn$. After the array is divided into two sub-arrays with size $n/2$, the time
needed to deal with each of them (also without counting the recurrence steps) is $c \frac{n}{2}$. Continue calculating this until we reach the array of size 1 and the whole tree structure of each step’s running time was written onto the white board.

At each level $k$, we have $2^k$ subarrays of size $n/(2^k)$ each, so the total time for all those calls is $cn$. Summing this up over all $1+\log(n)$ levels of this tree, we obtain that $T(n) = cn(1+\log(n)) = \Theta(n \log(n))$. 
2. Next, we proved the correctness of our guess $T(n) = cn (1+\log(n))$ by induction on $n$.

For the base case $n = 1$, $T(1)$ evaluates to $c$, which is correct.

For the induction step, assume $T(k) = ck (\log(k)+1)$ is correct for $k < n$.

According to the equation $T(n) = 2T(n/2) + cn$ and the induction hypothesis for $k = n/2$, we have $T(n) = cn+2 (n/2)(1+ \log(n/2)) = cn + cn (1+\log(n)-1) = cn (\log(n)+1)$. So the expression of $T(n)$ is also correct for $k = n$.

3. The above running time analysis process is rather time-consuming. To speed up the running time calculation for divide and conquer algorithms,
we introduced the Master Theorem. This theorem is actually not as famous or amazing as the name might suggest, but it is a convenient summarization of many standard Divide&Conquer recurrences.

The theorem gives the solution for the recurrence $T(n) = a \cdot T(n/b) + f(n)$, which can be viewed as dividing a problem of size $n$ into a sub-problems of size $(n/b)$ and then applying the same algorithm on those sub-problems. The theorem gives the solution $T(n)$ based on the asymptotic relation between $f(n)$ and $(n^{\log_b a})$.

With the help of the Master Theorem, we can often solve the running time of divide and conquer algorithms quickly without drawing the above running time analysis tree.

4. Take Merge Sort as an example, where $a = b = 2$, so $\log_b a = 1$ and $f(n) = cn = \Theta(n)$. It belongs to the second case of the Master Theorem, as a
result, we directly have the conclusion $T(n) = \Theta(n \log(n))$.

A few extra examples were provided (in this class and in the textbook as well) as exercises of the Master Theorem. Notice that in some cases, the Master Theorem does not cover the recurrence. For instance, the theorem tells you nothing about the solution of $T(n) = T(n/2) + T(2n/3)$. So sometimes you have to turn to the recurrence tree or other methods.

5. The next problem we studied is to compute the multiplication of two integers in bit form. The traditional shift&add method (from elementary school) has a running time of $\Theta(n^2)$. 

\[ \text{Merge Sort: } a+b=2 \quad \text{and } a=1, b=1, c=1 \]
\[ f(n) = c \cdot n = \Theta(n) \]

By case (ii) of Master Theorem, $T(n) = \Theta(n \log n)$.

Examples:

(i) $T(n) = a \cdot T(n/3) + n \log n$
   - $a = 3, b = 3, f(n) = n \log n, \ \log_3 a = 2$
   - $f(n) = O(n^{\log_3 a - \frac{1}{2}}) = O(n^{1/3})$

By case (ii), $T(n) = \Theta(n \log n)$.

\[ \text{Caution: } \text{This does not cover uneven divisions.} \ (T(n) = T(n/2) + T(2n/3)?) \]

Even for equal divisions, some smaller cases are not covered.

\[ (\text{e.g. } T(n) = 2T(n/2) + \log n) \]
6. Trying to improve the method by using the divide and conquer strategy, we considered a division of $x$ and $y$ into the $n/2$ higher-order bits and the $n/2$ lower-order bits. So $x = x_1 * x_0$, and $y = y_1 * y_0$, where $*$ denotes concatenation. Mathematically, we can rewrite this as $x = 2^{(n/2)}x_1 + x_0$, and $y = 2^{(n/2)}y_1 + y_0$. Then, we can rewrite the product $xy$ as $2^n x_1 y_1 + 2^{(n/2)}(x_1 y_0 + x_0 y_1) + x_0 y_0$. So we could recursively calculate $x_1 y_1$, $x_0 y_1$, $x_1 y_0$, $x_0 y_0$, and then use shift&add to combine the results.

The add and shift operations take $\Theta(n)$ time. The recurrence equation thus is $T(n) = 4 \cdot T(n/2) + cn$. It lies in the first case of Master Theorem, and gives us $T(n) = \Theta(n^2)$. This is not an improvement yet (we are where we started), but we will see next class how to get a faster
algorithm using this approach.

\[
\text{Suppose } x, y \text{ are } n\text{-bit numbers (} n \text{ even).} \\
x = x_0 x_1, \quad y = y_0 y_1 \text{ (concatenation).} \\
x = 2^{n/2} \cdot x_1 + x_0, \quad y = 2^{n/2} \cdot y_1 + y_0 \\
x \cdot y = 2^n x_1 y_1 + 2^{n/2} (x_1 y_0 + x_0 y_1) + x_0 y_0
\]

**Divide & Conquer:** Compute \(x_1 y_1, x_1 y_0, y_0 y_1, y_0 x_0\) recursively, shift and add.

Shifting and adding takes \( \Theta(n) \)

\[
T(n) = 4T(n/2) + O(n) \\
(\text{case } 1 \text{ of Master Theorem}) \quad \text{gives} \quad T(n) = \Theta(n^c).
\]