Course Staff

Victor Adamchik
Gary Miller

TAs
Laxman Dhulipala
Eugene Choi
Shen Chen Xu (Ph.D.)

Web Sites

www.cs.cmu.edu/afs/cs/academic/class/15451-s14
Calendar, Slides, Notes, Homeworks,
Course Policy, Grades, ...

http://piazza.com/
Questions, Comments, Announcements, ...

Textbook

There is no textbook.
Slides will be posted on the website.
Some supplementary notes will also be posted.

Grading

30% Homework (weekly)
10% Quizzes (weekly)
30% Tests (3 midterms)
30% Final

Intro to Algorithms

Outline

1. Administration
2. The Master Theorem
3. Karatsuba’s Algorithm
Homework
Homeworks roughly every week
Approx: 8 written and 3 oral
4 late days for written Hwks
2 late days at most per Hwk
We will drop the lowest written Hwk

Collaboration
You may work in a group of ≤ 3 people.
You must report who you worked with.
You must think about each of the problems by yourself for ≥ 30 minutes before discussing them with others.
You must write up all solutions by yourself.

Cheating
You MAY NOT
Share written work.
Get help from anyone besides your collaborators, staff.
Refer to solutions/materials from earlier versions of 251 or the web

Quizzes
Every week, in recitation
Tested on material from the previous 2-3 lectures.
These are designed to be easy, assuming you are keeping up with the lectures.
We will drop 2 lowest quizzes

Midterm Tests
There will be 3 tests given in the evening.
Designed to be doable in 1 hour.
You will have 1.5 hours.
"Semi-cumulative."
We will drop the lowest score.

Feel free to ask questions
Course Goals

1. Understand
   a) Algorithms
   b) Design techniques
2. Analyze algorithm efficiency
3. Analyze algorithm correctness
4. Communicate about code
5. Design your own algorithm

Divide and Conquer

A divide-and-conquer algorithm consists of
- dividing a problem into smaller subproblems
- solving (recursively) each subproblem
- then combining solutions to subproblems to get solution to original problem

Runtime

Suppose $T(n)$ is the number of steps in the worst case needed to solve the problem of size $n$. Let us split a problem into $a>1$ subproblems, each of which is of the input size $n/b$ where $b>1$.

$$T(n) = 2T(n/2) + n \quad T(n) = T(n/2) + 1$$

Merge sort  Binary search

The recurrences have some initial conditions

Runtime

The total complexity $T(n)$ is obtained by all steps needed to solve smaller subproblems $T(n/b)$ plus the work needed $f(n)$ to combine solutions into a final one.

$$T(n) = a \cdot T(n/b) + f(n)$$

Tree method: $T(n) = a \cdot T(n/b) + f(n)$

Draw a tree of recursive calls:

How do we solve this recurrence?

Tree of Recursive Calls!
Tree method: $T(n) = a \cdot T(n/b) + f(n)$

This tree represents the total work:

![Tree Diagram]

Leaves, $O(1)$

The Master Theorem

$T(n) = T(1) n^{\log_b a} \sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right)$

where $h = \log_b n$

$T(n) = \begin{cases} 
\Theta(n^{\log_b a}) & \text{Leaves dominate} \\
\Theta(n^{\log_b a} \log^p n) & \text{Both} \\
\Theta(f(n)) & \text{Internal nodes dominate} 
\end{cases}$

It (all) depend on the function $f(x)$ - a combining step

- Case I
  - if $f(n) \in O(n^{\log_b a - \delta})$, then $T(n) = O(n^{\log_b a})$

  **Proof.** The solution to the recurrence is $T(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right)$
  
  We simplify the sum in the rhs
  
  $\sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right) \leq \sum_{k=0}^{h-1} a^k \left(\frac{n}{b^k}\right)^{\log_b a - \delta} = c n^{\log_b a - \delta} \sum_{k=0}^{h-1} \left(\frac{a}{b^{\log_b a}}\right)^k b^{\delta k}$
  
  $= c n^{\log_b a - \delta} \sum_{k=0}^{h-1} b^{\delta k} \leq c n^{\log_b a - \delta} \sum_{k=0}^{h-1} b^{\delta k}$

  since $b^{\delta k} < 1$. It follows that $T(n) = \Theta(n^{\log_b a})$ QED

- Case II
  - if $f(n) \in \Omega(n^{\log_b a \log^{p-1} n})$, then $T(n) = \Omega(n^{\log_b a \log n})$

  **Proof.** We prove this for $p=1$. The solution to the recurrence is $T(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right)$

  We simplify the sum in the rhs
  
  $\sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right) = \sum_{k=0}^{h-1} a^k \left(\frac{n}{b^k}\right)^{\log_b a} = n^{\log_b a} \sum_{k=0}^{h-1} 1 = n^{\log_b a} \log n$

  It follows that $T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a \log n}) = \Theta(n^{\log_b a \log n})$ QED
Work at leaves is \( n \log_b a = n \log_2 4 = n^2 \)

Example:

\[ T(n) = \begin{cases} 
\Theta(n^{\log_b a}) \\
\Theta(n^{\log_b a} \log^3 n) \\
\Theta(f(n)) 
\end{cases} \]

\[ T(n) = 4T(n/2) + n \]

It follows, \( T(n) \in \Theta(n^2) \)

Example:

\[ T(n) = \begin{cases} 
\Theta(n^{\log_b a}) \\
\Theta(n^{\log_b a} \log^3 n) \\
\Theta(f(n)) 
\end{cases} \]

\[ T(n) = 4T(n/2) + n^2 \]

Work at leaves is \( n \log_b a = n \log_2 4 = n^2 \)

It follows, \( T(n) \in \Theta(n^2 \log n) \)

Example:

\[ T(n) = 2T(n/3) + 1 \]

\[ T(1) = 1 \]

Draw a tree of recursive calls:

Example:

\[ T(n) = 2T(n/3) + 1 \]

\[ T(1) = 1 \]

\[ T(n) = n^{\log_3 2} + \sum_{k=0}^{h-1} 2^k \]

\[ \text{height} \quad h = \log_3 n \]

\[ T(n) = n^{\log_3 2} + 2^h - 1 \]

\[ T(n) = -1 + 2 \cdot n^{\log_3 2} \]
Karatsuba’s Algorithm (1962)

Fast integer multiplication

Integer Multiplication

Given two $n$-digit integers.
Using a grammar school approach,
we can multiply them in $\Theta(n^2)$ time.

Observe, any integer can be split into two parts

154517766 = 15451 * $10^4$ + 7766

Integer Multiplication: divide-and-conquer

\[
\text{num}_1 = x_1 \times 10^p + x_0 \\
\text{num}_2 = y_1 \times 10^p + y_0 \\
\text{num}_1 \times \text{num}_2 = x_1 \times y_1 \times 10^{2p} + (x_1 \times y_0 + x_0 \times y_1) \times 10^p + x_0 \times y_0
\]

The worst-case complexity: by the master theorem

\[T(n) = 4T(n/2) + O(n) \quad T(n) = \Theta(n^2)\]

Karatsuba’s Algorithm

\[
\text{num}_1 \times \text{num}_2 = x_1 \times y_1 \times 10^{2p} + (x_1 \times y_0 + x_0 \times y_1) \times 10^p + x_0 \times y_0
\]

The worst-case complexity: by the master theorem

\[T(n) = 3T(n/2) + O(n) \quad T(n) = \Theta(n^{1.58})\]

3-way splitting

The key idea is to divide a large integer into 3 parts (rather than 2) of size approximately $n/3$ and then multiply those parts.
This is similar to 3-way merging.

The worst-case: (x is unknown)

\[T(n) = x \cdot T(n/3) + O(n) \quad T(n) = \Theta(n^{\log_3 x}) = O(n^{1.58})\]

\[\log_3 x < 1.58 \quad x = 5\]

Thus we need to reduce 9 mults to 5

Is it possible to reduce a number of multiplications from 9 to 5?
3-way split
T. Cook (1966)

\[ Z_0 = x_0 y_0 \]
\[ Z_1 = (x_0 + x_1 + x_2) (y_0 + y_1 + y_2) \]
\[ Z_2 = (x_0 + 2x_1 + 4x_2) (y_0 + 2y_1 + 4y_2) \]
\[ Z_3 = (x_0 - x_1 + x_2) (y_0 - y_1 + y_2) \]
\[ Z_4 = (x_0 - 2x_1 + 4x_2) (y_0 - 2y_1 + 4y_2) \]

Further Generalization:

**k-way split**

<table>
<thead>
<tr>
<th>splits</th>
<th>Number of multiplications</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

\[ T(n) = (2k-1)T(n/k) + n \]
\[ T(n) = n^{\log_k(2k-1)} \]

Is it always possible to reduce \( k^2 \) multiplications to \( 2k-1 \)?

Multiplication of large integers of \( n \) digits can be done in time \( O(n \log n \log \log n) \) thanks to the Fast Fourier Transform.

Is it always possible to reduce \( k^2 \) multiplications to \( 2k-1 \)?

Consider \( k \)-way split

\[ \text{polyn}_1 = a_{k-1} x^{k-1} + a_{k-2} x^{k-2} + ... + a_1 x + a_0 \]
\[ \text{polyn}_2 = b_{k-1} x^{k-1} + b_{k-2} x^{k-2} + ... + b_1 x + b_0 \]
\[ \text{polyn}_1 \times \text{polyn}_2 = a_{k-1} b_{k-1} x^{2k-2} + ... + (a_0 b_0 + a_1 b_1 + a_2 b_2) x + a_0 b_0 \]

It has \( 2k-1 \) coefficients, which uniquely define a polynomial. Therefore, it requires \( 2k-1 \) new variables, thus we should have at least \( 2k-1 \) multiplications.