

CMPSCI 611: “Advanced Algorithms”

Lecture 1

Andrew McGregor

Last Compiled: September 17, 2009

Outline

Introduction

Course Outline and Administrivia

Divide and Conquer

- Merge Sort

- Recurrences

- Matrix Multiplication

What's so great about algorithms?

- ▶ *Important to many areas of computer science and beyond:*
Being good at algorithms won't make you good at [insert subject here] but it'll help.

What's so great about algorithms?

- ▶ *Important to many areas of computer science and beyond:* Being good at algorithms won't make you good at [insert subject here] but it'll help.
- ▶ *Interesting in their own right:* Full of intellectual challenges, elegant mathematics, and the need to be ingenious!

What's so great about algorithms?

- ▶ *Important to many areas of computer science and beyond:* Being good at algorithms won't make you good at [insert subject here] but it'll help.
- ▶ *Interesting in their own right:* Full of intellectual challenges, elegant mathematics, and the need to be ingenious!

Goals of course:

- ▶ Learn some specific algorithms and specific techniques
- ▶ More importantly: Think algorithmically!

Abstraction: Makes it easier to study “efficiency”

- ▶ Making an algorithm efficient involves many considerations that are architecture specific. Let's ignore them!

Abstraction: Makes it easier to study “efficiency”

- ▶ Making an algorithm efficient involves many considerations that are architecture specific. Let's ignore them!
- ▶ RAM Model: We assume that any location in memory can be accessed a unit cost. Simple and reasonably realistic.

Abstraction: Makes it easier to study “efficiency”

- ▶ Making an algorithm efficient involves many considerations that are architecture specific. Let's ignore them!
- ▶ RAM Model: We assume that any location in memory can be accessed a unit cost. Simple and reasonably realistic.
- ▶ Don't measure running time in seconds, but rather “basic steps.” E.g.,
 1. pairwise arithmetic operation
 2. memory accesses

Abstraction: Makes it easier to study “efficiency”

- ▶ Making an algorithm efficient involves many considerations that are architecture specific. Let's ignore them!
- ▶ RAM Model: We assume that any location in memory can be accessed a unit cost. Simple and reasonably realistic.
- ▶ Don't measure running time in seconds, but rather “basic steps.” E.g.,
 1. pairwise arithmetic operation
 2. memory accesses
- ▶ Don't count steps exactly, consider basic steps $T(n)$ asymptotically as the size of the problem n grows. . .

Asymptotic Notation

- ▶ $f = O(g)$ means $\exists c, n_0 > 0$ such that $\forall n \geq n_0 : f(n) < cg(n)$
- ▶ $f = \Omega(g)$ means $\exists c, n_0 > 0$ such that $\forall n \geq n_0 : f(n) > cg(n)$
- ▶ $f = \Theta(g)$ means $f = O(g)$ and $g = O(f)$

Asymptotic Notation

- ▶ $f = O(g)$ means $\exists c, n_0 > 0$ such that $\forall n \geq n_0 : f(n) < cg(n)$
- ▶ $f = \Omega(g)$ means $\exists c, n_0 > 0$ such that $\forall n \geq n_0 : f(n) > cg(n)$
- ▶ $f = \Theta(g)$ means $f = O(g)$ and $g = O(f)$

- ▶ $f = o(g)$ means $\forall c > 0, \exists n_0$ such that

$$\forall n \geq n_0 : f(n) < cg(n)$$

- ▶ $f = \omega(g)$ means $\forall c > 0, \exists n_0$ such that

$$\forall n \geq n_0 : f(n) > cg(n)$$

Asymptotic Notation

- ▶ $f = O(g)$ means $\exists c, n_0 > 0$ such that $\forall n \geq n_0 : f(n) < cg(n)$
- ▶ $f = \Omega(g)$ means $\exists c, n_0 > 0$ such that $\forall n \geq n_0 : f(n) > cg(n)$
- ▶ $f = \Theta(g)$ means $f = O(g)$ and $g = O(f)$

- ▶ $f = o(g)$ means $\forall c > 0, \exists n_0$ such that

$$\forall n \geq n_0 : f(n) < cg(n)$$

- ▶ $f = \omega(g)$ means $\forall c > 0, \exists n_0$ such that

$$\forall n \geq n_0 : f(n) > cg(n)$$

Particularly interested in algorithms that run in polynomial time, i.e., the running time $T(n) = O(n^k)$ for some constant k .

Outline

Introduction

Course Outline and Administrivia

Divide and Conquer

- Merge Sort

- Recurrences

- Matrix Multiplication

Basic Stuff

Lectures are Tuesday and Thursday, 2:30 to 3:45 pm in MRST 211.

Basic Stuff

Lectures are Tuesday and Thursday, 2:30 to 3:45 pm in MRST 211.

Lecturer: Professor Andrew McGregor

- ▶ Email: mcgregor@cs.umass.edu
- ▶ Office: CMPS 334
- ▶ Office hours: Thursday 4:00 - 5:00, or by appointment.

Basic Stuff

Lectures are Tuesday and Thursday, 2:30 to 3:45 pm in MRST 211.

Lecturer: Professor Andrew McGregor

- ▶ Email: mcgregor@cs.umass.edu
- ▶ Office: CMPS 334
- ▶ Office hours: Thursday 4:00 - 5:00, or by appointment.

TA: Steve Murtagh

- ▶ Email: smurtagh@cs.umass.edu
- ▶ Office: TBA
- ▶ Office hours: TBA

Textbooks and Materials

Essential:

- ▶ M. Adler. Lecture Notes for 611 Advanced Algorithms

Textbooks and Materials

Essential:

- ▶ M. Adler. Lecture Notes for 611 Advanced Algorithms

Useful Background:

- ▶ T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms
- ▶ J. Kleinberg and E. Tardos. Algorithm Design

Textbooks and Materials

Essential:

- ▶ M. Adler. Lecture Notes for 611 Advanced Algorithms

Useful Background:

- ▶ T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms
- ▶ J. Kleinberg and E. Tardos. Algorithm Design

Useful Background:

- ▶ S. Dasgupta, C. Papadimitriou, U. Vazirani. Algorithms
- ▶ R. Motwani and P. Raghavan. Randomized Algorithms
- ▶ M. Mitzenmacher and E. Upfal. Probability and Computing
- ▶ V. Vazirani. Approximation Algorithms

Textbooks and Materials

Essential:

- ▶ M. Adler. Lecture Notes for 611 Advanced Algorithms

Useful Background:

- ▶ T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms
- ▶ J. Kleinberg and E. Tardos. Algorithm Design

Useful Background:

- ▶ S. Dasgupta, C. Papadimitriou, U. Vazirani. Algorithms
- ▶ R. Motwani and P. Raghavan. Randomized Algorithms
- ▶ M. Mitzenmacher and E. Upfal. Probability and Computing
- ▶ V. Vazirani. Approximation Algorithms

Other materials, including lecture slides, will be posted at:

www.cs.umass.edu/~mcgregor/courses/CS611/index.html .

Course Outline

- ▶ Preliminaries, Divide and Conquer (3 lectures)
- ▶ Greedy Algorithms, Matroids (4 lectures)
- ▶ Dynamic Programming, Shortest Paths, Network Flow (4 lectures)
- ▶ Randomized Algorithms (4 lectures)
- ▶ Approximation Algorithms, NP-Completeness (7 lectures)
- ▶ Linear Programming (3 lectures)

Assessment

- ▶ *Homeworks*: Approximately 5 assignments will contribute 40% to grade. Collaboration is allowed (in groups of size at most three) but solutions must be written separately and mention who you worked with.

Assessment

- ▶ *Homeworks*: Approximately 5 assignments will contribute 40% to grade. Collaboration is allowed (in groups of size at most three) but solutions must be written separately and mention who you worked with.
- ▶ *Exams*: There will be an in-class midterm and a final exam. Together the exams will contribute 50% to grade.

Assessment

- ▶ *Homeworks*: Approximately 5 assignments will contribute 40% to grade. Collaboration is allowed (in groups of size at most three) but solutions must be written separately and mention who you worked with.
- ▶ *Exams*: There will be an in-class midterm and a final exam. Together the exams will contribute 50% to grade.
- ▶ *Participation*: Remaining 10% of the grade will be based on class participation.

Outline

Introduction

Course Outline and Administrivia

Divide and Conquer

- Merge Sort

- Recurrences

- Matrix Multiplication

Outline

Introduction

Course Outline and Administrivia

Divide and Conquer

Merge Sort

Recurrences

Matrix Multiplication

Merge Sort

Problem: Given an unsorted list of n numbers, sort them!

Merge Sort

Problem: Given an unsorted list of n numbers, sort them!

Algorithm

1. *Divide list two halves.*
2. *Sort each half.*
3. *Merge the sorted halves.*

Merge Sort

Problem: Given an unsorted list of n numbers, sort them!

Algorithm

1. *Divide list two halves.*
2. *Sort each half.*
3. *Merge the sorted halves.*

Let running time of algorithm be $T(n)$. Observe $T(1) = \Theta(1)$ and, for $n > 1$,

$$T(n) = 2T(n/2) + \Theta(n)$$

Outline

Introduction

Course Outline and Administrivia

Divide and Conquer

Merge Sort

Recurrences

Matrix Multiplication

Solving Recurrences: Master Theorem

Theorem

Suppose $T(1) = \Theta(1)$ and $T(n) = aT(n/b) + \Theta(n^\alpha)$ for $n > 1$ where a, b are some constants. Then

$$T(n) = \begin{cases} \Theta(n^\alpha) & \text{if } \alpha > \beta \\ \Theta(n^\beta) & \text{if } \alpha < \beta \\ \Theta(n^\alpha \log n) & \text{if } \alpha = \beta \end{cases}$$

where $\beta = \log_b a$.

Solving Recurrences: Master Theorem

Theorem

Suppose $T(1) = \Theta(1)$ and $T(n) = aT(n/b) + \Theta(n^\alpha)$ for $n > 1$ where a, b are some constants. Then

$$T(n) = \begin{cases} \Theta(n^\alpha) & \text{if } \alpha > \beta \\ \Theta(n^\beta) & \text{if } \alpha < \beta \\ \Theta(n^\alpha \log n) & \text{if } \alpha = \beta \end{cases}$$

where $\beta = \log_b a$.

Therefore, Merge-Sort takes $\Theta(n \log n)$ time.

Proof (1/2)

- ▶ Assume n is a power of b but theorem holds in general.

Proof (1/2)

- ▶ Assume n is a power of b but theorem holds in general.
- ▶ Repeatedly expand $T(n)$ to get

$$\begin{aligned}T(n) &= aT(n/b) + \Theta(n^\alpha) \\ &= a^2T(n/b^2) + a\Theta((n/b)^\alpha) + \Theta(n^\alpha) \\ &= \dots \\ &= a^{\log_b n}\Theta(1) + a^{\log_b n-1}\Theta(b^\alpha) + \dots + a\Theta((n/b)^\alpha) + \Theta(n^\alpha)\end{aligned}$$

Proof (1/2)

- ▶ Assume n is a power of b but theorem holds in general.
- ▶ Repeatedly expand $T(n)$ to get

$$\begin{aligned}T(n) &= aT(n/b) + \Theta(n^\alpha) \\ &= a^2T(n/b^2) + a\Theta((n/b)^\alpha) + \Theta(n^\alpha) \\ &= \dots \\ &= a^{\log_b n}\Theta(1) + a^{\log_b n-1}\Theta(b^\alpha) + \dots + a\Theta((n/b)^\alpha) + \Theta(n^\alpha)\end{aligned}$$

- ▶ Substitute $r = a/b^\alpha = b^{\beta-\alpha}$ to get

$$T(n) = n^\alpha \Theta(r^{\log_b n} + r^{\log_b n-1} + \dots + r + 1)$$

Proof (2/2)

- ▶ For $r = a/b^\alpha = b^{\beta-\alpha}$ we have

$$T(n) = n^\alpha \Theta(r^{\log_b n} + r^{\log_b n - 1} + \dots + r + 1)$$

Proof (2/2)

- ▶ For $r = a/b^\alpha = b^{\beta-\alpha}$ we have

$$T(n) = n^\alpha \Theta(r^{\log_b n} + r^{\log_b n - 1} + \dots + r + 1)$$

- ▶ If $r = 1$, then $T(n) = \Theta(n^\alpha \log_b n) = \Theta(n^\alpha \log n)$

Proof (2/2)

- ▶ For $r = a/b^\alpha = b^{\beta-\alpha}$ we have

$$T(n) = n^\alpha \Theta(r^{\log_b n} + r^{\log_b n - 1} + \dots + r + 1)$$

- ▶ If $r = 1$, then $T(n) = \Theta(n^\alpha \log_b n) = \Theta(n^\alpha \log n)$
- ▶ If $r \neq 1$, then

$$T(n) = \Theta\left(n^\alpha \frac{r^{1+\log_b n} - 1}{r - 1}\right) = \begin{cases} \Theta(n^\alpha) & \text{if } r < 1 \\ \Theta(n^\alpha r^{\log_b n}) & \text{if } r > 1 \end{cases}$$

Proof (2/2)

- ▶ For $r = a/b^\alpha = b^{\beta-\alpha}$ we have

$$T(n) = n^\alpha \Theta(r^{\log_b n} + r^{\log_b n - 1} + \dots + r + 1)$$

- ▶ If $r = 1$, then $T(n) = \Theta(n^\alpha \log_b n) = \Theta(n^\alpha \log n)$
- ▶ If $r \neq 1$, then

$$T(n) = \Theta\left(n^\alpha \frac{r^{1+\log_b n} - 1}{r - 1}\right) = \begin{cases} \Theta(n^\alpha) & \text{if } r < 1 \\ \Theta(n^\alpha r^{\log_b n}) & \text{if } r > 1 \end{cases}$$

- ▶ Result follows since $r^{\log_b n} = b^{(\beta-\alpha) \log_b n} = n^{\beta-\alpha}$.

Outline

Introduction

Course Outline and Administrivia

Divide and Conquer

Merge Sort

Recurrences

Matrix Multiplication

Matrix Multiplication

Have two $n \times n$ matrices A and B and want to multiply them together to get C :

$$c_{ij} = \sum_{k \in [n]} a_{ik} b_{kj}$$

Naive algorithm works in $O(n^3)$ time.

Matrix Multiplication

Have two $n \times n$ matrices A and B and want to multiply them together to get C :

$$c_{ij} = \sum_{k \in [n]} a_{ik} b_{kj}$$

Naive algorithm works in $O(n^3)$ time.

- ▶ Divide A and B into four $n/2 \times n/2$ sub-matrices:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Matrix Multiplication

Have two $n \times n$ matrices A and B and want to multiply them together to get C :

$$c_{ij} = \sum_{k \in [n]} a_{ik} b_{kj}$$

Naive algorithm works in $O(n^3)$ time.

- ▶ Divide A and B into four $n/2 \times n/2$ sub-matrices:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

- ▶ And note

$$C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

Matrix Multiplication

Have two $n \times n$ matrices A and B and want to multiply them together to get C :

$$c_{ij} = \sum_{k \in [n]} a_{ik} b_{kj}$$

Naive algorithm works in $O(n^3)$ time.

- ▶ Divide A and B into four $n/2 \times n/2$ sub-matrices:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

- ▶ And note

$$C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

- ▶ **Bad News:** $T(n) = 8T(n/2) + \Theta(n^2)$ gives $T(n) = \Theta(n^3)$

Strassen's Algorithm

Break the problem into 7 sub-problems:

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_2 = (A_{21} + A_{22})(B_{11})$$

$$P_3 = (A_{11})(B_{12} - B_{22})$$

$$P_4 = (A_{22})(-B_{11} + B_{21})$$

$$P_5 = (A_{11} + A_{12})(B_{22})$$

$$P_6 = (-A_{11} + A_{21})(B_{11} + B_{12})$$

$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

Strassen's Algorithm

Break the problem into 7 sub-problems:

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_2 = (A_{21} + A_{22})(B_{11})$$

$$P_3 = (A_{11})(B_{12} - B_{22})$$

$$P_4 = (A_{22})(-B_{11} + B_{21})$$

$$P_5 = (A_{11} + A_{12})(B_{22})$$

$$P_6 = (-A_{11} + A_{21})(B_{11} + B_{12})$$

$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$\text{Then } AB = \begin{pmatrix} P_1 + P_4 - P_5 + P_7 & P_3 + P_5 \\ P_2 + P_4 & P_1 - P_2 + P_3 + P_6 \end{pmatrix}$$

Strassen's Algorithm

Break the problem into 7 sub-problems:

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_2 = (A_{21} + A_{22})(B_{11})$$

$$P_3 = (A_{11})(B_{12} - B_{22})$$

$$P_4 = (A_{22})(-B_{11} + B_{21})$$

$$P_5 = (A_{11} + A_{12})(B_{22})$$

$$P_6 = (-A_{11} + A_{21})(B_{11} + B_{12})$$

$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$\text{Then } AB = \begin{pmatrix} P_1 + P_4 - P_5 + P_7 & P_3 + P_5 \\ P_2 + P_4 & P_1 - P_2 + P_3 + P_6 \end{pmatrix}$$

Good: $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$ gives $T(n) = \Theta(n^{\log_2 7})$ where $\log_2 7 \approx 2.81$ by applying Master theorem.

Strassen's Algorithm

Break the problem into 7 sub-problems:

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_2 = (A_{21} + A_{22})(B_{11})$$

$$P_3 = (A_{11})(B_{12} - B_{22})$$

$$P_4 = (A_{22})(-B_{11} + B_{21})$$

$$P_5 = (A_{11} + A_{12})(B_{22})$$

$$P_6 = (-A_{11} + A_{21})(B_{11} + B_{12})$$

$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$\text{Then } AB = \begin{pmatrix} P_1 + P_4 - P_5 + P_7 & P_3 + P_5 \\ P_2 + P_4 & P_1 - P_2 + P_3 + P_6 \end{pmatrix}$$

Good: $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$ gives $T(n) = \Theta(n^{\log_2 7})$ where $\log_2 7 \approx 2.81$ by applying Master theorem.

Better: Improved to $O(n^{2.38})$ by Coppersmith and Winograd.

For Next Time...

- ▶ Get Adler notes from the Textbook Annex.
- ▶ Read first and second chapter.