CPSC 221
Basic Algorithms and Data Structures

Introduction

Textbook References:
Koffman: Chapters P and 1

Hassan Khosravi
January - April 2015

Learning goal(s)

- Get an overview of the course

- Learning Goals for each chapter will be given
  - Please pay attention to them.
  - They're part of the lecture slides.
  - They make great exam checklists.

Course Personnel

- Your Instructor:
  - Hassan Khosravi
  - ICCS 241
  - hkhosrav@cs.ubc.ca
  - Office hours
    - Right after lectures in class
    - Mondays 3-4, and Tuesdays 11-12 in ICCS 241
    - By appointment; see my webpage for my work calendar.
- The other instructor
  - Alan Hu
- Teaching Assistants and office hours: See website

Course information

- The course home page http://www.ugrad.cs.ubc.ca/~cs221
- Read it carefully
  - Course info
  - Notes (Pre-lecture notes, Post-lecture notes)

- Texts: Epp Discrete Mathematics, Koffman C++
  - Exams are open book

Connect

- Quizzes and your grades are hosted on Connect
  - http://elearning.ubc.ca/connect/

- To access Connect you need a Campus-Wide Login (CWL). To register for a CWL account, visit www.cwl.ubc.ca.

Piazza

- The course bulletin (Piazza)
  - https://piazza.com/configure-classes/winterterm22014/cpsc221
    - **Enrollment code is…**
  - The Piazza bulletin board is required reading. It will be used for important material that may not be mentioned in class.
  - Problems with the CPSC 221 course contents (e.g. lecture, lab, textbook, assignments) can be posted on Piazza.
  - Students are encouraged to ask questions, and to respond to other students' questions.
  - Selected individuals that actively respond to postings on Piazza, may receive a bonus mark of up to 2%.
  - Only send email for personal issues. (ALWAYs add CPSC 221 in the title of your email)

Exams

- Midterm: Wed. February 25, 6:00pm – 8:30pm
  - Location to be announced
  - Potentially two stage exam
  - If you have a conflict let me know ASAP and no later than Thursday, January 8

- Final: To be determined by UBC.
  - Examination period begins: **Tuesday, April 14 2015**
  - Examination period ends: **Wednesday, April 29 2015**

Labs

- (roughly) every week. Please attend assigned lab section for now.
- Labs start Monday, January 12.
  - In room x350

- We'll try to have the labs posted by Thursday evenings.

Assignments

- No late work; may be flexible with advance notice
- Programming projects (~3) typically due 9PM on due date
  - All programming projects graded on Linux/g++

- Written homework typically due 5PM on due date
  - Assignment box 35

PeerWise

- Students use PeerWise to create and to explain their understanding of course related assessment questions, and to answer and discuss questions created by their peers.
  - https://peerwise.cs.auckland.ac.nz/docs/

- Making multiple-choice tests requires a much higher level of understanding than simply taking them!

PeerWise

- Encouraging students to "own their own learning" in a sharing and supportive environment. Posing good, thoughtful questions with supporting explanations is an excellent application of critical thinking skills.

- Answering questions and giving comments offers opportunities to improve one's communication and self-reflection abilities.

Clickers

- We will be using clickers in the section of the course, so please register your i>Clicker using UBC Connect
  - A typical pattern used in my lectures is: students vote on their own first; for more difficult questions students are encouraged to participate in group-discussions
  - You can participate in answering questions anonymously
  - Responses are projected, in real-time, so you can compare yourself with your peers
  - Reveals misconceptions and I can pace the presentation and explanations to fit the audience

Planned (tentative) assessments

• Labs	10%
• Theory Assns/Quizzes	15%
• Programming Projects	15%
• Midterm Exam	20%
• Final Exam	35%
• PeerWise	5%
• Piazza	2% (Bonus)

- To pass the course, you must obtain at least a 50% overall course mark (as per the above formula) and you must pass the final exam.

Prerequisites

- One of CPSC 210, EECE 210
- One of CPSC 121, MATH 220
- If you don't have the prerequisites, you will be dropped from the course. Check with a CS Advisor if you have a problem

- The instructor cannot change your lab section or add you to the course.
  - Check with CS front office

Academic Concession

- Students registered with Access & Diversity must:
  - inform the instructor within seven (7) days of adding the course or the start of term (whichever is later) (email me please)
  - provide a copy of the letter from A&D granting accommodations (by email)
  - submit an accommodation form at least two weeks prior to each exam for which accommodation is requested

Collaboration

- Read the collaboration policy on the website. You have **LOTS** of freedom to collaborate! Use it to learn and have fun while doing it!
- Don’t violate the collaboration policy. There’s no point in doing so.
- Plagiarism occurs when you submit someone else's work as if it were your own.

Using your computer during class…

Multitasking on your computer during CPSC 221 lectures:

A. Helps me to learn the course materials because I'm not bored and has no effect on other people
B. Helps me to learn better, but distracts the people around me
C. Makes me learn worse, but has no effect on other people
D. Makes me learn worse, and distracts the people around me

Multi-tasking on laptop lowers your grade (Canadian study)

- "all the participants used laptops to take notes during a lecture on meteorology. But half were also asked to complete a series of unrelated tasks on their computers when they felt they could spare some time. Those tasks — which included online searches for information — were meant to mimic what distracted students might do during class."
- The students who were asked to multitask averaged 11% lower on the exam (difference between A- and B-)
- Their neighbors also did significantly worse than the average.

Some class rules

- Show up on time for class.
- Be respectful. Don't disrupt the class for your fellow students. Examples of behavior to be <u>avoided</u> during class:
  - Using a cell phone in class
  - Surfing the Web on your laptop, iPad, smartphone, etc. in class
  - Talking loudly with your friends when I'm lecturing.
    - Whisper please, and only talk briefly.

Acknowledgement

- Thanks to Steve Wolfman for the content of most of these slides with additional material from Alan Hu, Ed Knorr, Will Evans, and Kim Voll

Observation

- Most programs manipulate data
  - programs *process, store, display, gather*
  - data can be *numbers, images, sound* (information!)
- Each program must decide how to store and manipulate data
- Choice influences program at every level
  - execution speed
  - memory requirements
  - maintenance (debugging, extending, etc.)

How you structure your data matters to every program you create!

What this course is about

- Some Classic Algorithms
- Some Classic Data Structures
- Analysis Tools and Techniques for the Above
  - what's good or bad
  - what trade-offs are being made
- Some Basics of Parallelism and Concurrency

What’s an algorithm?

- Algorithm (Typical Definition)
  - A high-level, language-independent description of a step-by-step process for solving a problem

- Algorithm (Street Definition)
  - A smarter way to solve the problem!

What’s a Data Structure?

- Data Structure (Typical Definition)
  - A way to store data to facilitate its manipulation/ access
- Data Structure (Street Definition)
  - How to organize your data to get the results you want, along with the supporting algorithms

Why study classic examples?

- They are useful!
  - Like pre-packaged intelligence in a can!
  - Don’t have to work hard to come up with your own solution
- They let you abstract away details!
  - These are “power tools” for programming.
  - Let you focus on solving bigger problems, ignore details.
- You learn general solution ideas!
  - This will help you solve new, unexpected problems.
  - Great masters in any field study the classic examples from their field.

Goals of the Course

- Become familiar with some of the fundamental data structures and algorithms in computer science
- Improve ability to solve problems abstractly
  - data structures and algorithms are the building blocks
- Improve ability to analyze your algorithms
  - prove correctness
  - gauge, compare, and improve time and space complexity
- Become modestly skilled with C++ and UNIX, but this is ***largely on your own***!

Fun Example

- We'll look at a simple example, to see how different choices affect performance:
  - Fibonacci Numbers
- Does performance matter in practice?
  - Massive load on web applications: Anyone use Cuil instead of Google?
  - Efficient algorithms allow lower power, longer battery life, cheaper processors, etc.

The Fibonacci Numbers

- The Fibonacci numbers are the numbers in the sequence:
  1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …
- The first two numbers are 1, and then each succeeding number can be generated by adding together the previous two numbers in the sequence. This leads to the following recursive definition:

```
int fib(int n){
    if (n==1)
        return 1;
    else if(n==2)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

  - Common example in CS
  - Brief appearance in Da Vinci Code
  - Some applications, pops up in unusual places (art, nature, algorithm analysis)

n=4

```
int fib(int n){
→if (n==1)
   return 1;
→else if (n==2)
   return 1;
→else
→  return fib(n-1)
   + fib(n-2);
}
```

n=3

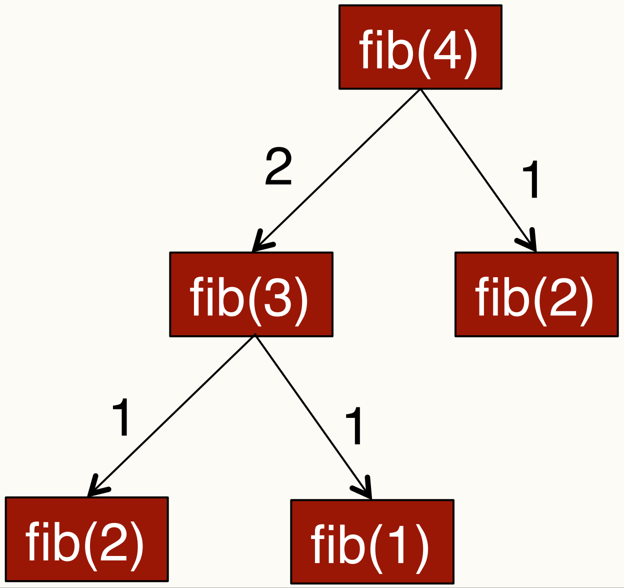
```
int fib(int n){
→if (n==1)
   return 1;
→else if (n==2)
   return 1;
→else
→  return fib(n-1)
   + fib(n-2);
}
```

n=2

```
int fib(int n){
→if (n==1)
   return 1;
→else if (n==2)
→  return 1;
 else
   return fib(n-1)
   + fib(n-2);
}
```

n=1

```
int fib(int n){
→if (n==1)
→  return 1;
 else if (n==2)
   return 1;
 else
   return fib(n-1)
   + fib(n-2);
}
```

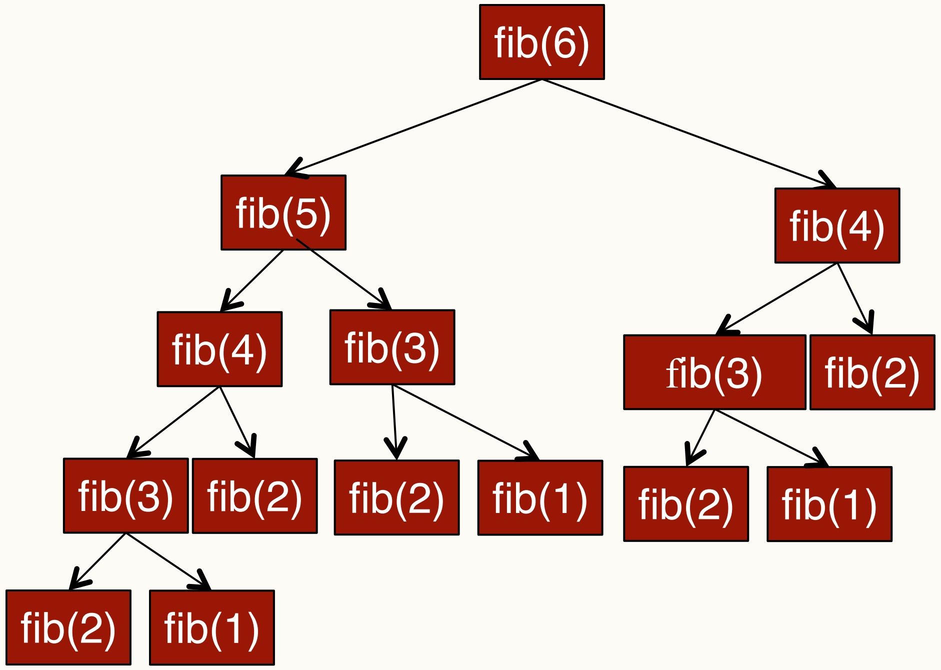


n=2

```
int fib(int n){
→if (n==1)
   return 1;
→else if (n==2)
→  return 1;
 else
   return fib(n-1)
   + fib(n-2);
}
```

Recursion and efficiency

- How many times is fib(3) calculated?

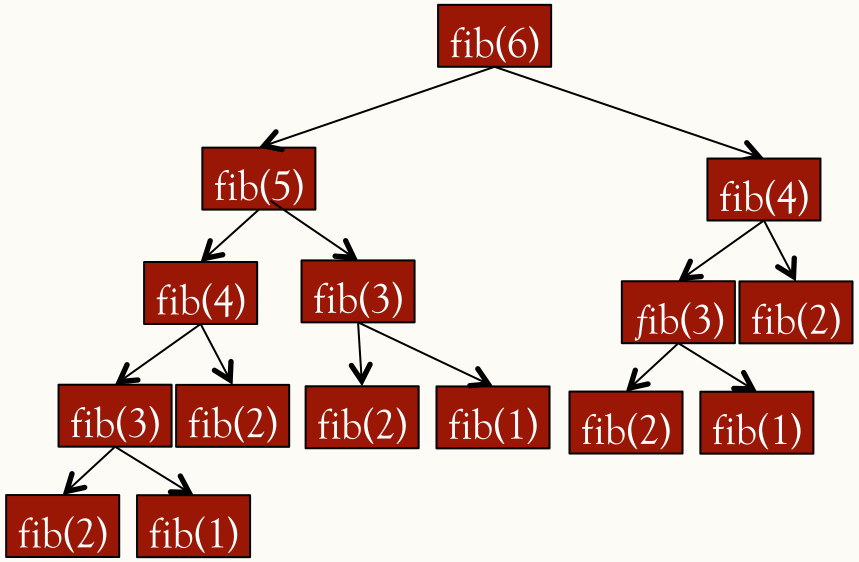


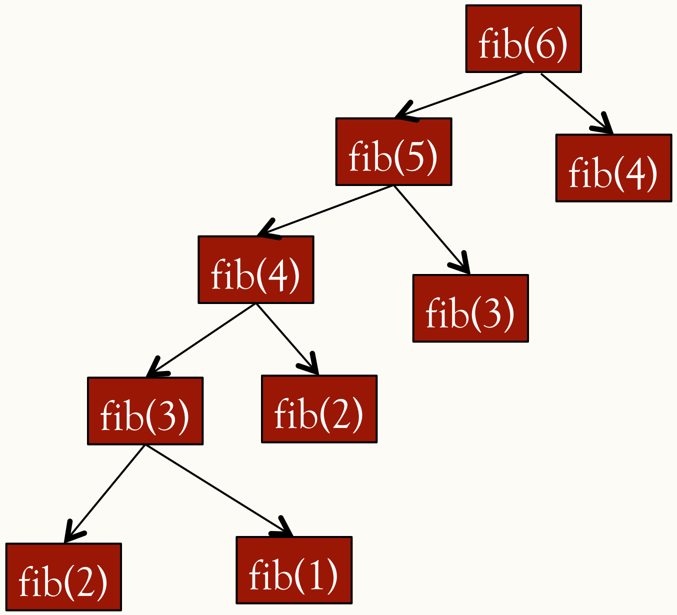
- Does it really need to be calculated multiple times?

Memoization

– After computing a solution, store it in a table before returning. (Leave a “memo” to yourself.)

– At start of the function, check if you’ve solved this case before. If so, return the already calculated solution.





n	F_n
6	8
5	5
4	3
3	2
2	1
1	1

Dynamic programming

- In dynamic programming you solve the problem "bottom up”, so you don’t risk blowing stack space.

```
unsigned long long dp_fib(int n) {
    unsigned long long answer;
    unsigned long long answerminus1 = 1;
    unsigned long long answerminus2 = 1;
    for (int i = 3; i <= n; i++) {
        answer = answerminus1 + answerminus2;
        answerminus2 = answerminus1;
        answerminus1 = answer;
    }
    return answer;
}
```

See course website for source code

Matrix Multiplication

- Consider this matrix equation:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + y \\ x \end{bmatrix}$$

- Hey! That's one iteration of Fibonacci!

Matrix Fibonacci

- Repeated matrix multiplication computes Fibonacci numbers…

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}=\begin{bmatrix}2\\1\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}2\\1\end{bmatrix}=\begin{bmatrix}3\\2\end{bmatrix}$$

$$\begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}3\\2\end{bmatrix}=\begin{bmatrix}5\\3\end{bmatrix}$$

$$\vdots$$

Repeated Multiplication is Exponentiation!

$$T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} fib(n) \\ fib(n-1) \end{bmatrix} = T^{n-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Multiplication is associative.

- Associative Law: (xy)z = x(yz)
- Therefore,

$$\begin{aligned} x^n &= x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot\ldots \\ &= ((((x\cdot x)\cdot x)\cdot x)\cdot x)\cdot x\cdot x\cdot x\cdot\ldots \\ &= (x\cdot x)\cdot(x\cdot x)\cdot(x\cdot x)\cdot(x\cdot x)\cdot\ldots \\ &= x\cdot((x\cdot x)\cdot(x\cdot x))\cdot x\cdot(x\cdot x)\cdot\ldots \\ &= \ldots \end{aligned}$$

Matrix Multiplication Is Associative

$$\begin{aligned}T^n &= T \cdot T \cdot T \cdot T \cdot T \cdot T \cdot T \cdot T \cdot T \cdot T \cdot T \cdot \ldots \\ &= T^2 \cdot T^2 \cdot T^2 \cdot T^2 \cdot T^2 \cdot \ldots \\ &= T^4 \cdot T^4 \cdot T^4 \cdot T^4 \cdot T^4 \cdot \ldots \\ &= \ldots\end{aligned}$$

Exponentiation by Iterative Squaring

- Imagine you have an old calculator and need to compute 2^{32}.
- You could type **`2 x 2 x 2 x 2 x`** ... .
- Or, you could type:
  - **2 x 2** **=** and get 2^2.
  - $\mathbf{2^2}$ **x** $\mathbf{2^2}$ **=** and get 2^4.
  - $\mathbf{2^4}$ **x** $\mathbf{2^4}$ **=** and get 2^8.
  - $\mathbf{2^8}$ **x** $\mathbf{2^8}$ **=** and get 2^{16}.
  - $\mathbf{2^{16}}$ **x** $\mathbf{2^{16}}$ **=** and get 2^{32}.

Iterative squaring works for matrices, too!

$$T^2 = T \cdot T$$

$$T^4 = T^2 \cdot T^2$$

$$T^8 = T^4 \cdot T^4$$

$$T^{16} = T^8 \cdot T^8$$

$$T^{32} = T^{16} \cdot T^{16}$$

$$T^{64} = T^{32} \cdot T^{32}$$

$$\vdots$$

Iterative Squaring Example

$$T^{100} = T^{64} \cdot T^{36}$$

$$= T^{64} \cdot T^{32} \cdot T^{4}$$

Do only 8 multiplications instead of 99!

Comparison of different fib implementations (in seconds)

	n=10	n=50	n=10,000	n=200,000	100,000,000
Plain, recursive	**$3 * 10^{-5}$**	**71**			
Memoized recursive	**$5 * 10^{-6}$**	**$7 * 10^{-6}$**	**$3 * 10^{-4}$**	Out of memory	
Dynamic programming	**$3 * 10^{-6}$**	**$5 * 10^{-6}$**	**$4 * 10^{-5}$**	**$7 * 10^{-4}$**	0.4
Matrix Multiplication	**$5 * 10^{-6}$**	**10^{-6}**	**$9 * 10^{-5}$**	**10^{-5}**	**$2 * 10^{-4}$**