CS 1501
bill-computer.science/1501

Introduction
These notes are intended for use by students in CS1501 at the University of Pittsburgh. They are provided free of charge and may not be sold in any shape or form.

These notes are NOT a substitute for material covered during course lectures. If you miss a lecture, you should definitely obtain both these notes and notes written by a student who attended the lecture.

Material from these notes is obtained from various sources, including, but not limited to, the following:

- *Algorithms in C++* by Robert Sedgewick
- *Algorithms, 4th Ed.* by Robert Sedgewick and Kevin Wayne
- *Data Structures and Abstractions with Java* by Carrano and Henry
- *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein
- Various Java and C++ textbooks
- Various online resources (see notes for specifics)
Classroom sites and technologies

- **Syllabus website:**
  - https://bill-computer.science/1501
  - Course policies, contact info, slide PDFs, readings

- **Canvas:**
  - Gradebook, office hours, announcements, assignment summary
  - Panopto (lecture recordings), Redshelf (textbook inclusive access)

- **GitHub:**
  - Programming project distribution and version control
  - Codespaces (develop from anywhere in a web browser)

- **Gradescope:**
  - Programming project submission and grading

- **Top Hat:**
  - In-lecture participation questions
Instructor Info

- Dr. Garrison (or bill)
  - Email: bill@cs.pitt.edu
  - Office: 6311 Sennott Square

- Static information on the syllabus website, day-to-day information on Canvas
Why should I take this course?

● To understand and utilize the **algorithms and data structures** that are fundamental to computer science
  ○ Trees, search trees, graphs, etc.
  ○ What are they and how do I use them?

● To understand **implementation issues** related to these data structures, and to see how they can be implemented in Java
  ○ Algorithms and data structures can be implemented in various ways, and these decisions have implications (run-time, code complexity, modifiability...).
1. **Read** the assigned chapters
   ○ We will not discuss all details in lecture
2. In **Lecture**, we discuss the abstract concepts and write some code together
3. **Recitation** provides more hands-on practice with code
   ○ Submitted each week
4. **Programming Projects** combine hands-on practice and evaluation
   ○ Completed individually. Expect many smaller problems to arise along the way.
5. **Pen-and-paper exams** demonstrate your ability to apply to new scenarios
   ○ Week 6 and Week 12
An aside about shared code...

- What is the role of shared code in “real-life”?
  - You may know of ways in which sharing code is beneficial and/or commonplace in some contexts
  - Why re-write code when someone else publishes theirs?
    - Consider licensing issues, greater efficiency of custom-built code, easier to modify/extend code you’re familiar with
    - Following code use policies is a necessary skill!
    - Research has found lots of security vulnerabilities and other bugs introduced via sloppy use of code from (e.g.) Stack Overflow answers
  - In this class, you are not permitted to share project code
    - Do not send code to a classmate, look for code online, etc.
Why is sharing code not allowed in this course?

- Project submissions aren’t a **functional deliverable**: I’m not asking you for this code for the sake of accomplishing a task
  - Assignments are both **practice** and **evaluation**
- **Practice**: By completing these projects **on your own**, you’re achieving structured learning objectives
  - Using external code would mean learning less!
- **Evaluation**: We’re trying to determine what you can do on your own / how much you’ve **learned**
  - You’re not on the job yet; this is still the **interview**!
  - The grade in this course is assigned for **individual** work

Shared code in either direction (distributing or receiving) is a violation of the Academic Integrity Policy

- Help each other long-lasting ways, not short-term
An aside about shared code...

- What will happen if I resort to shared code?
  - All submissions will be automatically processed for similarity to a corpus of hundreds of solutions
    - Your classmates’, former students’, found online, donated by other instructors, etc.
    - Starter code is included to reduce false positives
  - Comparison of the structure of the code
    - Minor changes will not prevent a match
    - If you know enough to re-write the code to avoid a match, you know enough to write it yourself to begin with!
  - I will manually inspect all non-trivial similarity
    - Those that indicate shared code will result in a report to the Dean’s Office and a sanction, up to F in the course

- Again, this includes all forms of sharing: Posting your code online, sending it to a classmate, submitting code that someone else wrote or helped write, reading solutions from GitHub or Chegg or StackOverflow or...
What can I do instead of sharing code, if I’m struggling on a project?

- Discuss the assignment at a high level with a classmate
  - Do this in the abstract, without having your code open, and without referring to specific details
- Attend office hours (mine or the TA’s)
- Re-read the assigned chapters
- Attempt a simplified version of the problem (reduction) or a subset of the problem (division)
  - General problem-solving strategies
- Get some rest and try again with a clear head
  - Only works if you start early!
- If a friend asks for help, suggest the above

An aside about shared code...
Let’s check out the syllabus website

● Website:
  ● https://bill-computer.science/1501

● Review the Course Information and Policies

● Assignments will not be accepted after the deadline
  ○ No late assignment submissions
  ○ If you do not submit an assignment by the deadline, you will receive a 0 for that assignment
Early programming classes focused on how you *could* solve a problem. In Algorithms and Data Structures, we’re starting to look at how you *should* solve a problem.
First some definitions:

- Offline problem
  - We provide the computer with some input and after some time receive some acceptable output

- Algorithm
  - A step-by-step procedure for solving a problem or accomplishing some end

- Program
  - An algorithm expressed in a language the computer can understand

An algorithm solves a problem if it produces an acceptable output on every input
More experience converting increasingly tricky *algorithms* into *programs*

- Many seemingly simple algorithms can become much more complicated as they are converted into programs
- Algorithms can also be very complex to begin with, and their implementation must be considered carefully
- Various issues will always pop up during *implementation*
  - Such as?...
Pseudocode for dynamic programming algorithm for relational query optimization

The optimizer portion of the PostgreSQL codebase is over 28,000 lines of code (i.e., not counting blank/comment lines)
To ensure you see and understand differences in algorithms and how they affect the run-times of the associated programs

- Different algorithms can be used to solve the same problem
- Different solutions can be compared using many metrics
  - Run-time is a big one
    - Better run-times can make an algorithm more desirable
    - Better run-times can sometimes make a problem solution feasible where it was not feasible before
- There are other metrics, though...
How to determine an algorithm’s performance

- Implement it and measure performance
  - Any problems with this approach?

- Algorithm Analysis
  - Determine resource usage as a function of input size
  - Measure asymptotic performance
    - Performance as input size increases to infinity
What does Big O mean?

- Upper bound on asymptotic performance
- Assuming that definition, if something is $O(n^3)$, then
  - Is it $O(n^4)$?
  - What about $O(n^5)$?
  - What about $O(3^n)$?
- We want to find the lowest upper bound
  - Get closest to proving the actual runtime of the algorithm
Big O isn’t the whole story

- Big Omega
  - Lower bound on asymptotic performance

- Theta
  - Upper and Lower bound on asymptotic performance
  - Exact bound
Resource Usage

\[ O(n^3) \]

\[ \Omega(n) \]

Actual Runtime (somewhere in here)

Input Size (n)
Resource Usage

\[ O(n^3) \]
\[ \Theta(n^3) \]
\[ \Omega(n^3) \]

Input Size (n)
f(x) is $O(g(x))$ if constants $c$ and $x_0$ exist such that:
- $|f(x)| \leq c \cdot |g(x)| \quad \forall x > x_0$

f(x) is $\Omega(g(x))$ if constants $c$ and $x_0$ exist such that:
- $|f(x)| \geq c \cdot |g(x)| \quad \forall x > x_0$

if $f(x)$ is $O(g(x))$ and $\Omega(g(x))$, then $f(x)$ is $\Theta(g(x))$
- $c_1, c_2,$ and $x_0$ exist such that:
  - $c_1 \cdot |g(x)| \leq |f(x)| \leq c_2 \cdot |g(x)| \quad \forall x > x_0$

May also see $f(x) \in O(g(x))$ or $f(x) = O(g(x))$ used to mean that $f(x)$ is $O(g(x))$
- Same for $\Omega$ and $\Theta$
Mathematically modelling runtime

- Runtime primarily determined by two factors:
  - Cost of executing each statement
    - Determined by machine used, environment running on the machine
  - Frequency of execution of each statement
    - Determined by program and input
For us to prove an algorithm is $\Theta(n^3)$, we would first need to prove that it has:

- Upper bound of $n^3$: $O(n^3)$
- Lower bound of $n^3$: $\Omega(n^3)$
- At that point, we would have determined that $n^3$ is an exact bound on the runtime of the algorithm: $\Theta(n^3)$

Tilde approximations?

- Introduced in section 1.4 of the text
- Basically Theta bounds with multiplicative constants
  - We won’t be using them in this class
    - (ever)
Common orders of growth

- Constant - 1
- Logarithmic - \( \log n \)
- Linear - \( n \)
- Linearithmic - \( n \log n \)
- Quadratic - \( n^2 \)
- Cubic - \( n^3 \)
- Exponential - \( 2^n \)
- Factorial - \( n! \)
Quick algorithm analysis

- Ignore multiplicative constants and lower terms
- Use standard measures for comparison
Bursting balloons

\[ a[i][0] \quad a[i][1] \]
public int findMinArrows(int[][] points) {
    Arrays.sort(points,
            (i, j) -> Integer.compare(i[1], j[1])
    );
    int cur = points[0][1];
    int arrows = 1;
    for (int i = 1; i < points.length; i++) {
        if (points[i][0] > cur) {
            arrows += 1;
            cur = points[i][1];
        }
    }
    return arrows;
}
def findMinArrows(self, points: List[List[int]]) -> int:
    points = sorted(points, key=lambda x: x[1])
    cur = points[0][1]
    arrows = 1
    for start, end in points[1:]:
        if start > cur:
            arrows += 1
            cur = end
    return arrows
pub fn find_min_arrows(mut points: Vec<Vec<i32>>) -> i32 {
    points.sort_unstable_by_key(|a| a[1]);
    let mut cur = points[0][1];
    let mut arrows = 1;
    for i in points.iter().skip(1) {
        let (start, end) = (i[0], i[1]);
        if start > cur {
            cur = end;
            arrows += 1;
        }
    }
    arrows
}
Topics for the term

- Trees
- Tries
- Heaps and Priority Queues
- Compression
- Graph Algorithms
- Dynamic Programming