Lecture 1: Introduction and Overview

CSCI 700 – Algorithms 1
What is an algorithm?

• A high level description of a process.
• Based on some input, an algorithm describes a method to generate output.
• An algorithm should solve some problem – captured by the input/output relationship
Example algorithm

- **Problem:** find the index of an element in an array
- **Input:** the array, A, and the element, x
- **Output:** the index in A of element x

```plaintext
function find(x, A)
    i ← 0
    while i < size(A)
        if A[i] = x
            return i
        end if
        i ← i + 1
    end while
end
```
Pseudocode

- Pseudocode allows you to describe an algorithm without the syntax and semantics of a specific programming language.

- Pseudocode can gloss over implementational details.
  - Pseudocode can include plain English sentences or phrases along with code.

- It can be convenient to think of the relationship between pseudocode and code as similar to that between an “outline” and a paper.
Why study algorithms?

• Efficiency is fundamental to successful software engineering.

• Designing efficient algorithms is essential.
• Identifying inefficiencies is just as important.
• Studying algorithms will help you write efficient software.

• Also, it will get you a job.
Why search and sort?

• **Search** is fundamental to many IO operations – find a file, book a flight, find an article, Google.

• Structuring information can make retrieval more efficient.

• **Sorting** is an easy to understand linear structuring of information.

• Sorting and searching is a case study for the interaction between structure and retrieval.

• Graphs and graph functions, hashing etc. are all more complicated examples of this structure/retrieval relationship.
Review: Data Structures

• Three data structures that we will be relying on heavily in this course.
  – Arrays
  – Trees
  – Graphs
Review: Arrays

• **Arrays** are linear series of data.

• Operations we assume to be available:
  – \([i]\) – access the ith element of the array
  – size – how many elements are in the array
Review: Trees

• Binary **Trees** contain a set of Nodes. Each Node can have a left or right child.

• Operations we assume to be available:
  – parent – get the parent node
  – left – get the left child
  – right – get the right child
Review: Graphs

• A **Graph** is defined by a set of Vertices or Nodes, and a set of Edges that connect pairs of Vertices.

• Operations we assume to be available:
  – Vertex::adjacent_vertices
  – Vertex::out_edges
  – Vertex::in_edges
  – Edge::source_vertex
  – Edge::dest_vertex

• Both Arrays and Trees can be viewed as special cases of Graph
Math we’ll use

• Exponents and Logarithms are used heavily. Log with no subscript is taken to be log base 2.

\[ x^n = y \]
\[ n = \log_x y \]
\[ \log xy = \log x + \log y \]
\[ \log \frac{x}{y} = \log x - \log y \]
\[ n \log x = \log x^n \]

• Summations

\[ \sum_{i=1}^{n} 1 = n \]
\[ \sum_{i=1}^{n} x = nx \]
\[ \sum_{i=1}^{n} i = \frac{i(i+1)}{2} \]

• Inductive proofs (tomorrow)
Example: Insertion Sort

• Sort an Array $A = [5, 2, 4, 6, 1, 3]$
• Take element $j=2$, move it to the right until $A[1..2]$ is correctly sorted.
• Take element $j=3$, move it to the left until $A[1..3]$ is sorted
• Continue up to $j=n$. 
Insertion Sort

Corman et al. p. 17
function insertionSort(A)
    for i « 2 to size(A) do
        key « A[j]
        i « j - 1
        while i > 0 and A[i] > key do
            i « i - 1
        end while
        A[i + 1] « key
    end for
end
Insertion Sort

function insertionSort(A)
    for i ← 2 to size(A) do
        key ← A[j]
        i ← j - 1
        while i > 0 and A[i] > key do
            A[i + 1] ← A[i]
            i ← i - 1
        end while
        A[i + 1] ← key
    end for
end
Insertion Sort

function insertionSort(A)
    for i « 2 to size(A) do
        key « A[j]
        i « j - 1
        while i > 0 and A[i] > key do
            i « i - 1
        end while
        A[i + 1] « key
    end for
end

• Total Runtime = \( n + n-1 + n-1 + n-1 + \sum_{j=2}^{n} j + \sum_{j=2}^{n} (j-1) + \sum_{j=2}^{n} (j-1) + n-1 \)
Insertion Sort

• Total runtime=

\[ n + n - 1 + n - 1 + n - 1 + \sum_{j=2}^{n} j + \sum_{j=2}^{n} (j - 1) + \sum_{j=2}^{n} (j - 1) + n - 1 \]

\[ \frac{n(n + 1)}{2} - 1 + 2 \left( \frac{n(n - 1)}{2} \right) + 5n - 4 \]

\[ \frac{n^2 + n}{2} - 1 + n^2 - n + 5n - 4 \]

\[ \frac{3}{2} n^2 + \frac{9}{2} n - 5 \]

• Insertion sort has “quadratic runtime” or \( \Theta(n^2) \)
Correctness of Insertion Sort

• We show insertion sort is correct using a construct called a **Loop Invariant**.

• Three properties of a Loop Invariant for an algorithm to be considered correct.
  1. **Initialization** – It is true at the start of the loop.
  2. **Maintenance** – It is true at the end of the loop.
  3. **Termination** – When the loop terminates, the Loop Invariant should show that the algorithm is correct.
Insertion Sort Loop Invariant

• **Loop Invariant:** At the start of each for loop iteration, A[1..j-1] contains the items initially in A[1..j-1] but in sorted order.

```
function insertionSort(A)
    for i « 2 to size(A) do
        key « A[j]
        i « j - 1
        while i > 0 and A[i] > key do
            i « i - 1
        end while
        A[i + 1] « key
    end for
end
```
Insertion Sort Loop Invariant

• **Loop Invariant:** At the start of each for loop iteration, $A[1..j-1]$ contains the items initially in $A[1..j-1]$ but in sorted order.

• **Initialization:** When $j=2$, the subarray $A[1..j-1] = A[1..1]=A[1]$ only contains one element, so it is sorted.
Insertion Sort Loop Invariant

• **Loop Invariant:** At the start of each for loop iteration, A[1..j-1] contains the items initially in A [1..j-1] but in sorted order.

• **Maintenance:** The body of the for loop moves A [j-1] down to A[1] one position to the right, until the correct position for A[j] is found. At which point A[j] is inserted.
  – Formally, we need to show that at the point where A [j] is inserted A contains only elements less than or equal to A[j] to the left, and only elements greater than A[j] to the right.
Insertion Sort Loop Invariant

- **Loop Invariant:** At the start of each for loop iteration, A[1..j-1] contains the items initially in A[1..j-1] but in sorted order.

- **Termination:** The loop ends when j=n+1. Therefore, A[1..n] contains the items initially in A[1..n] but in sorted order. Thus, the entire array is sorted.
Class Policies and Course Overview

• Course Website
Textbook

Bye.

• Next time:
  – Asymptotic Notation
  – Recursion
  – Inductive Proofs.

• Next Class:
  – Homework 1: Demographic Information Due.
  – Read Section 3.1