CSCI 241: Data Structures

Lecture 1
Introduction
Course Overview
Intro to Sorting
Today

1. About Me

2. Course Overview and a few notes on the syllabus

3. Insertion Sort and Selection Sort
About Me

Scott Wehrwein

📍 🍷
Computer Vision: Familiar Examples

In-Camera Face Detection

Autonomous Driving

Panorama Stitching

Image Search
Data Structures: Why?

Graph

Tree

Hash table
Syllabus Overview

Course website:

https://facultyweb.cs.wwu.edu/~wehrwes/courses/csci241_19w

Also linked from the Syllabus section on Canvas.
Goals

- Understand the range index convention a..b
- Know the definition of specification, precondition, postcondition, and invariant.
- Be able to execute insertion sort and selection sort on paper.
- Be able to implement insertion sort and selection sort.
Sorting Algorithms

Why?

- Arrays are the simplest and most ubiquitous data structure available to us.
- Sorting algorithms are a fundamental piece of knowledge for computer scientists.
- An entry point into the practice of developing, and analyzing algorithms.
Preliminaries: Tools for Talking about Algorithms
Range Indices

\[a..b\] denotes the range of consecutive integers from (and \textit{including}) \(a\) up to (but \textit{excluding}) \(b\).

Examples:

- 0..5 is the range 0, 1, 2, 3, 4
- A[4..6] denotes the 4th and 5th elements of A
- 7..8 is a range containing only 7
- 6..6 is a valid range but contains no elements
Range Indices

\[ a..b \] denotes the range of consecutive integers from (and including) \( a \) up to (but excluding) \( b \).

• How many elements are in the range \( a..b \)?

A. \( b - a - 1 \)

B. \( a - b - 1 \)

C. \( b - a + 1 \)

D. \( B - a \)
Range Indices

\(a..b\) denotes the range of consecutive integers from (and \textbf{including}) \(a\) up to (but \textbf{excluding}) \(b\).

- Recall that \(A\. \text{length}\) gives \(A\)'s length. What range denotes all elements of \(A\)?

  A. \(A[0..A\. \text{length}]\)

  B. \(A[0..A\. \text{length}-1]\)

  C. \(A[0..A\. \text{length}+1]\)

  D. \(A[1..A\. \text{length}-1]\)
Specification

```java
/** return the max value in A
 * precondition: A is nonempty
 * postcondition: max value of A is returned */

public int findMax(int[] A) {
    int max = A[0];
    // invariant: max is the largest value in A[0..i]
    for (int i = 1; i < A.length; i++) {
        if (A[i] > max) {
            Max = A[i];
        }
    }
    return max;
}
```

A method specification is a comment above the method that details the precise behavior of the method.
The precondition is true before method execution.
The postcondition is true after method execution.
/** return the max value in A
 * precondition: A is nonempty
 * postcondition: max value of A is returned */

public int findMax(int[] A) {
    int max = A[0];
    // invariant: max is the max of A[0..i]
    for (int i = 1; i < A.length; i++) {
        if (A[i] > max) {
            max = A[i];
        }
    }
    return max;
}

A loop invariant is true before, during, and after the loop.
(at the end of each iteration)
The loop invariant is true before, during, and after the loop.
Onward to sorting!
Insertion Sort
Insert $A[i]$ into the sorted sublist $A[0..i-1]$.

Selection Sort
Find the smallest element in $A[i..n]$ and place it at $A[i]$.
Insertion Sort

Insert $A[i]$ into the sorted sublist $A[0..i-1]$.

Invariant: $A$ sorted

 Selection Sort

Find the smallest element in $A[i..n]$ and place it at $A[i]$.

Invariant: $A$ sorted, $\leq A[i..n]$
insertionSort(A):
  i = 0;
  while i < A.length:
    // push A[i] to its sorted position by repeatedly
    //   swapping with the element to its left
    // increment i
  
Invariant: A sorted, <= A[i..n] ?

selectionSort(A):
  i = 0;
  while i < A.length:
    // find min of A[i..A.length]
    // swap it with A[i]
    // increment i
  
Invariant: A sorted, <= A[i..n] ?
// Sorts A using insertion sort
insertionSort(A):
    i = 0;
    while i < A.length:
        j = i;
        while j > 0 and A[j] > A[j-1]:
            swap(A[j], A[j-1])
            j--
        i++

// Invariant: A[0..i-1] is sorted