CS 624: Notes 01

Ryan Culpepper

September 7, 2022

These notes contain an outline of what I said in lecture (but only an outline), and they also contain interactive questions and exercises. The corresponding slides are in `slides01.pdf`.

1 Introduction

The course web page (not yet complete):

`https://www.cs.umb.edu/~ryanc/cs624/`

Other information:

- No programming, only math.
- Typed homework or *very* clearly handwritten.
- Attendance is not required, but highly encouraged.
  You are responsible for everything said in class.
- Don't be afraid to ask questions.

Question: How many of you have used LaTeX?
Answer: A few, not many.

(skip to slide 8)

2 Slide 8

What is the goal of algorithm analysis?

The programmer wants to know:

> How long will my program take?

That's very hard to answer, even for a program on one specific computer. Some reasons:

- Hardware (instruction time, memory latency, caches, etc)
- Operating system (scheduling and paging)

- Software (VM warmup, GC)
- Inputs

It's not a very universal answer. Some reasons:

- Hardware changes over time.
- Your inputs might look different from others.

So we abstract away these issues.

- **PRO:** It makes analysis tractable and "scalable". Programs grow. They rarely shrink.
- **CON:** The analysis doesn't give precise predictions.

3 Slide 11: Insertion Sort

```
InsertionSort(A) :=

for j ← 2 to length[A] do
  key ← A[j]
  i ← j - 1
  // Insert A[j] into sorted sequence A[1..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
```

4 Slide 13

What is the correctness theorem for this algorithm?

1. The code must terminate (without an error).
2. Output should be related to the input... somehow. How?
   - length should be the same
   - should have the same elements
   - should be sorted (each element is $\leq$ next)

In other words:

The array's elements should be

- a permutation (rearrangement) of the original array's elements

- sorted (ascending, $\leq$)

5 Slide 14

I don't consider this a good example of induction, but the "maintenence" (or "preservation") case is similar to the "inductive" case in induction.

Note: In this class, arrays are fixed-length. Their length cannot be changed (for example, by assigning past the end of the array).

Two parts:

- termination
  - The `for` loop terminates because we're just going from 2 to `length[A]`, and because the body terminates (but we need to show that!)
  - The `for` body terminates because:
    The `while` loop only continues while `i > 0`. But `i` decreases each time the `while` loop body is executed. An integer cannot decrease forever and stay positive.
  - And it terminates without an error.
    TODO: Prove that all of the references (eg, `A[j]`) and assignments to the array are "in bounds".
- loop invariant
  - initially true
  - maintained on each loop iteration
  - on termination, yields a useful property

Loop invariant:

> The numbers in `A[1 .. (j-1)]` are in sorted order, and they are a permutation of the original `A[1 .. (j-1)]`.

What is the property after loop termination?

What is the value of `j` after the `for` loop terminates?

```
j = length[A] + 1
```

Do substitution into loop invariant statement:

> The numbers in `A[1 .. (length[A])]` are in sorted order, and they are a permutation of the original elements.

That's the entire array!

The correctness of the entire algorithm is the same as the loop invariant after the termination of the loop! (In this case, because there is no code after the `for` loop.)

(Stopped here.)

6 Questions and Comments

One student recommends `https://www.overleaf.com/`, an online editor for LaTeX.

Question: Will there be review classes before the exams?
Answer: Yes, I'll schedule some time before the exams for review.