CS302 - Introduction

David Kauchak

- "For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing." – Francis Sullivan
- What is an algorithm?
- Examples
  - sort a list of numbers
  - find a route from one place to another (cars, packet routing, phone routing, ...)
  - find the longest common substring between two strings
  - add two numbers
  - microchip wiring/design (VLSI)
  - solve sudoku
  - cryptography
  - compression (file, audio, video)
  - spell checking
  - pagerank
  - classify a web page
  - ...
- What properties of algorithms are we interested in?
  - does it terminate?
  - is it correct, i.e. does it do what we think it's supposed to do?
  - what are the computational costs?
  - what are the memory/space costs?

  - what happens to the above with different inputs?
  - how difficult is it to implement and implement correctly?

- Why are we interested? Most of the algorithms/data structures we will discuss have been around for a while and are implemented. Why should we study them?

  - For example, look at the java.util package
    * Hashtable
    * LinkedList
    * Stack
    * TreeSet
    * Arrays.binarySearch
    * Arrays.sort
  - Know what's out there/possible/impossible
  - Know the right algorithm to use
  - Tools for analyzing new algorithms
  - Tools for developing new algorithms
  - interview questions? :)
    * Describe the algorithm for a depth-first graph traversal.
    * Write a function f(a, b) which takes two character string arguments and returns a string containing only the characters found in both strings in the order of a. Write a version which is O(n^2) and one which is O(n).
    * You're given an array containing both positive and negative integers and required to find the sub-array with the largest sum (O(n) a la KBL). Write a routine in C for the above.
    * Reverse a linked list
    * Insert in a sorted list
    * Write a function to find the depth of a binary tree
    * ...
  - Personal experience: Understanding and developing new algorithms has been one of the most useful tools/skills for me.

- Pseodocode

- A way to discuss how an algorithm works that is language agnostic and without being encumbered with actual implementation details.
- Should give enough detail for a person to undersand, analyze and implement the algorithm.
- Conventions

  MYSTERY1(A)

  ```
  1  x ← −∞
  2  for i ← 1 to length[A]
  3          if A[i] > x
  4                  x ← A[i]
  5  return x
  ```

  MYSTERY2(A)

  ```
  1  for i ← 1 to ⌊length(A)/2⌋
  2          swap A[i] and A[length(A) − (i − 1)]
  ```

- Comments
  - array indices start at 1 not 0
  - we may use notation such as ∞, which, when translated to code, would be something like Integer.MAX_VALUE
  - use shortcuts for simple function (e.g. swap) to make pseudocode simpler
  - we'll often use $\leftarrow$ instead of $=$ to avoid ambiguity
  - indentation specifies scope

- Proofs
  - What is a proof?

    A deductive argument showing a statement is true based on previous knowledge (axioms)
  - Why are they important/useful?

    A proof should be an airtight argument/analysis of why something is true. Without a proof, we may guess that a statement is true, however, until we show it to be true rigorously, we cannot be sure.

    Proofs often come up in algorithm analysis since they allow us to prove properties about algorithms that may be useful.

- A basic proof to get us started: prove that the sum of two odd integers is even.

  The key thing for proofs is that you must be clear and precise!

  - First, let's state clearly what we're trying to prove: If n and m are odd integers, then $n + m$ is even.
  - For the proof, we walk through the set of logical steps:

    By the definition of "odd" if n and m are odd, then we know that there exists some integers j and k such that:
    $n = 2j + 1$
    $m = 2k + 1$

    This means that the sum of n and m is:
    $n + m = 2j + 1 + 2k + 1 = 2j + 2k + 2 = 2(j + k + 1)$

    By the definition of "even" (an even number is any number that is equal to 2 times some integer), given that j and k are integers then $j + k + 1$ is an integers and therefore $n + m$ is even.

- Proof by induction

  - What are some of the types of "proof techniques" you've heard of or used before?
  - What is a proof by induction?

    A proof by induction is a proof technique for proving something about a sequence of events by showing first proving that some starting case is true and then also proving that if a given event in the sequence were true then the *next* event would be true. By combining these two parts, we show that all of the events in the sequence are true.

    Key parts to a proof by induction (*include each of these in your proofs by induction!*):

    1. **Base case:** Prove some starting case is true
    2. **Inductive case:** Assume some event is true and prove the next event is true
       (a) **Inductive hypothesis:** Assume the event is true (usually k or $k - 1$)

(b) **Inductive step to prove:** What you're trying to prove *assuming* the inductive hypothesis is true

(c) **Proof of inductive step**

- Example

  Prove that $\sum_1^n i = \frac{n(n+1)}{2}$

  1. **Base case:** Show that it is true for $n = 1$
     $\sum_{i=1}^1 i = 1 = \frac{1*2}{2}$
  2. **Inductive case:**
     (a) **Inductive hypothesis:** assume $n = k - 1$ is true, that is
     $\sum_{i=1}^{k-1} i = \frac{(k-1)k}{2}$
     (b) **Inductive step to prove:** show that
     $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$
     (c) **Proof:**

$$\begin{aligned}
\sum_{i=1}^{k} i &= k + \sum_{i=1}^{k-1} i \\
&= k + \frac{(k-1)k}{2} \\
&= \frac{2k}{2} + \frac{(k-1)k}{2} \\
&= \frac{2k + (k-1)k}{2} \\
&= \frac{k^2 + k}{2} \\
&= \frac{k(k+1)}{2}
\end{aligned}$$

  - Why has this proved what we wanted to show?

- Sorting

  Input: An array of numbers A
  Output: The array of numbers in sorted order, i.e. $A[i] \leq A[j]\ \forall i < j$

  - Insertion sort

INSERTION-SORT(A)

```
1  for j ← 2 to length[A]
2        current ← A[j]
3        i ← j − 1
4        while i > 0 and A[i] > current
5              A[i + 1] ← A[i]
6              i ← i − 1
7        A[i + 1] ← current
```

– Does it terminate?

– Is the algorithm correct?

Loop invariant: A statement about a loop that is true before the loop begins and after each iteration of the loop.

Upon termination of the loop, the invariant should help you show something useful about the algorithm.

INSERTION-SORT invariant: At the start of each iteration of the **for** loop of lines 1-7 the subarray $A[1..j-1]$ is the sorted version of the original elements of $A[1..j-1]$

To prove, need to show two things:

* Base case: invariant is true before the loop
* Inductive case: it is true after each iteration

Proof

* How does the loop invariant help us prove the correctness of the algorithm?
  When the loop finishes, we'll be at the "start" of the iteration for $length[A]+1$. Our loop invariant tells us then that $A[1..\,length[A]$ is sorted, i.e. that the whole array is sorted.

– Running time: How long does it take? How many computational "steps" will be executed?

What is our computational model? Turing machine? We'll assume a random-access machine (RAM) model of computation.

Examine costs for each step

Let n be the length of the array and t_j be the number of iterations of the while loop when examining element $A[j]$.

$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n}(t_j - 1) + c_6 \sum_{j=2}^{n}(t_j - 1) + c_7(n-1)$

* Best case: array is sorted
  $t_j = 1$
  $\sum_{j=2}^{n} = n$ - **Linear**
* Worst case: array is in reverse sorted order
  $t_j = j$
  $\sum_{j=2}^{n} = n + n - 1 + n - 2 + \cdot + 2 = \frac{n(n+1)}{2} - 1$ - **Quadratic**
* Average case: array is in random order
  The array up through j is sorted. How many entries on average will we have to analyze before in the sorted portion of the array to find the correct location for the current element?
  $t_j = j/2$
  $\sum_{j=2}^{n} = \frac{\frac{n(n+1)}{2}}{2} - 1/2$ - **Quadratic**
* Can we do better? What about if we used binary search to find the correct position?

– What is the memory usage of Insertion-Sort?
  Insertion-Sort only uses a bit more memory than the original array (*current*, i and j need to be allocated), but this is a fixed number.

These notes are adapted from material found in chapters 1 + 2 of [1].

References

[1] Thomas H. Cormen, Charles E. Leiserson Ronald L. Rivest and Clifford Stein. 2007. Introduction to Algorithms, 2nd ed. MIT Press.