Course Notes for

CS 1501
Algorithm Implementation

By

John C. Ramirez

Department of Computer Science

University of Pittsburgh

- These notes are intended for use by students in CS1501 at the University of Pittsburgh and no one else
- These notes 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 Edition by Robert Sedgewick and Kevin Wayne
  - Introduction to Algorithms, by Cormen, Leiserson and Rivest
  - Various Java and C++ textbooks
  - Various online resources (see notes for specifics)

- CS 1501 Summer 2020 will be delivered online
  - Please review all course policies in the following location:
    - https://people.cs.pitt.edu/~ramirez/cs1501/
      - Click on the Course Info link
  - These policies will be discussed during the first synchronous lecture:
    - Monday, May 11, 2020, 10:20AM-11:15AM
  - After reading these policies, if you have any questions / concerns / problems / or need any accommodation, please contact me ASAP

- Office Hours:
  - Office hours will be held over Zoom
    - My office hours for the Summer Term will be:
      - Monday, 2:30PM-3:30PM
      - Tuesday, 10:00AM-12:00PM
      - Wednesday, 2:30PM-3:30PM
      - and by appointment
    - A virtual waiting room will be utilized and students will be seen one at a time
    - See email for Zoom link information
  - Your TA will also have office hours
    - See email for Zoom link information

- Text:
  - **Algorithms Fourth Edition** by Sedgewick and Wayne [Pearson]
    - See: http://www.pearsonhighered.com/educator/product/Algorithms/9780321573513.page
  - An e-copy should be available via inclusive access from Pitt on Canvas
    - You will automatically be billed for this e-copy
  - If you would prefer to get it on your own, you can opt out of this and will not have to pay.
    - This is done via Canvas in the e-book environment or by emailing the University Store: theuniversitystore@pitt.edu

Goals of Course

- Definitions:
  - Offline Problem:
    - We provide the computer with some input and after some time receive some acceptable output
    - No hard deadline for *when* the problem should be solved (as opposed to a real-time problem)
  - 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

- Goals of this course:
  1) To learn how to convert (nontrivial) algorithms into programs
     - Often what seems like a fairly simple algorithm is not so simple when converted into a program
     - Other algorithms are complex to begin with, and the conversion must be carefully considered
     - Many issues must be dealt with during the implementation process
       - We will get a few ideas in our interactive lecture

2) To see and understand differences in algorithms and how they affect the run-times of the associated programs

- Many problems can be solved in more than one way
- Different solutions can be compared using many factors
  - One important factor is program **run-time**
    - > Sometimes a better run-time makes one algorithm more desirable than another
    - > Sometimes a better run-time makes a problem solution feasible where it was not feasible before
  - However, there are **other factors** for comparing algorithms
    - > We will discuss some in the interactive lecture

- Asymptotic Analysis
  - First we should determine what we want to analyze
    - Usually it is time (as in run-time) but not always
    - What other resource may we want to keep track of when a program is run?
  - For time we determine a key instruction (or instructions) which drives the overall run-time
  - Determine a function that models the overall run-time behavior of the algorithm
    - Ex: $F(N) = 2N^2 + 6N + 100$

$F(N) = N^2$

- But we will ignore / remove
  - lower order terms
  - multiplicative constants
  - Why?
    - Discuss and see notes below this slide
- Use some standard measure for categorization
  - Do we know any measures?
    - Big O
  - You should be familiar with this from CS 0445
    - However, there are other ways to categorize asymptotic run-times

- Big O gives us an upper bound on the asymptotic performance
  - Think: $F(N) <= 2N^2 + 6N + 100 =< O(N^2)$
- Big Omega
  - Gives us a lower bound on the asymptotic performance
  - Think: $F(N) >= 2N^2 + 6N + 100 => \Omega(N^2)$
- Theta
  - Upper and lower bound on the asymptotic performance – exact bound
  - Think: $F(N) == 2N^2 + 6N + 100 => \Theta(N^2)$
- Why three different measures?
  - When analyzing algorithms we cannot always be precise
  - Sometimes we must make assumptions / simplifications

  - These make it difficult to determine an exact (Theta) bound
- We also may have different goals
  - Big Omega tells us "we cannot do better than this"
  - Big O tells us "we can definitely do this well or better"
  - Theta tells us "we can do exactly this well"
    - All of these are expressed within a constant factor

Ex: Comparison based sorting

- It has been proven that any comparison based sorting algorithm is $\Omega(N \lg N) \rightarrow (n \log(n))$
  - This tells us that any new algorithm that we develop to sort which compares values will require at least this much time

- Some sorting algorithms achieve this lower bound (ex: MergeSort always, QuickSort in average case)
  - We can say MergeSort is Theta(NlgN)
- Others do not

- Generally speaking
  - Big-O tends to be easier to determine than Big-Omega
    - And also Theta since Theta requires both lower and upper bounds
    - This is why Big-O is commonly used
  - However, if we can be precise enough to give a Theta bound it is better

- Tilde approximations and order of growth
  - Text introduces an alternative notation
    - Read text Section 1.4 for rationale for it
  - Practically speaking, tilde is like Theta but without dropping the multiplicative constants
  - Ex: $f(N) = 6N^2 + 18N - 50$
    - Theta(N^2) (or order of growth N^2)
    - $\sim 6N^2$
  - The constants can be useful for real analysis, but are somewhat implementation dependent
    - As discussed in previous slides

- So is algorithm analysis really important?
  - Yes! Different algorithms can have considerably different run-times
    - Sometimes the differences are subtle but sometimes they are extreme
    - Let's look at a table of growth rates on the next slide
      - Note how drastic some of the differences are for large problem sizes
      - Note especially the last two columns
        - Solutions with exponential and hyper-exponential run-times are not practical for even relatively small values of N

lg_2N	N	NlgN	N^2	2^N	N!
2	4	8	16	16	24
3	8	24	64	256	40320
4	16	64	256	64K	> 20T
5	32	160	1024	4G	$> 10^{35}$
…	…	…	…	…	…
10	1024	10240	1M	2^{1024}	PUNT
20	1M	20M	1T	2^{1M}	PUNT
30	1G	30G	$> 10^{18}$	2^{1G}	PUNT

- Consider 2 choices for a programmer
  1) Implement an algorithm, then run it to find out how long it takes
  2) Determine the asymptotic run-time of the algorithm, then, based on the result, decide whether or not it is worthwhile to implement the algorithm
  - Which choice would you prefer?
  - Discuss

  - The previous few slides should be (mostly) review of CS 0445 material

- Example:
  - ThreeSum example from text
    - Given a set of arbitrary integers (could be negative)
      - Find out how many distinct triples sum to exactly zero
    - Ex: A = 10 40 -20 25 -10 -15 30 5 -35
      - One answer is 25, -10, -15, another is -20, -10, 30
    - Simple solution:
      - Triple for loops (keeping track of indices)
        - For each i from 0 to N-1 in the list
          - For each j from i+1 to N-1 in the list
            - For each k from j+1 to N-1 in the list

              Count solution if A[i]+A[j]+A[k] = 0
        - See ThreeSum.java
      - Works but clearly is Theta(N^3)

Algorithm Analysis

- Better solution:
  - Sort the numbers first (using a fast sort)
    - How long?
    - Theta(NlgN)
  - Now we add two numbers in our initial two loops and
    - If we can find in the array a number with the opposite value of that sum, we will have a solution
    - We can do this using binary search
    - For each i from 0 to N-1
      - For each j from i+1 to N-1
        - Let sum = A[i] + A[j]
        - Let k = binarySearch(A, -sum)
        - if (k > j) answer is i, j, k
  - Ex: A[i] = -20   A[j] = -10
    - We can do a binary search for the value 30. If we find it, we have a solution, otherwise we do not

- We are still using two nested for loops to get A[i] and A[j], but we eliminate the third loop and replace it with a binary search
  - > Improve the run-time from Theta(N^3) to Theta(N^2lgN)
  - > BUT - we must add the time to sort, which we did not have to do before
    - > We must be careful not to ignore extra computation that we may introduce
  - > But we know a fast sort takes NlgN so it does not add to our asymptotic run-time
  - > This gives us Theta(NlgN + N^2lgN) => Theta(N^2lgN)
- See ThreeSumFast.java in text
  - > Note: Both versions assume no duplicates. How would duplicates complicate the solution?
    - > See comment below in notes