# CS 305
# Design and Analysis of Algorithms

09 / 13 / 2021

Instructor: Michael Eckmann

# Today's Topics

- Questions/Comments about anything from 1st class or reading assignment?
- Course overview slides that I didn't have time to do last time
- Review Big O, Big Omega, Big Theta
- Another example proof
- Little o, little omega
- Arithmetic series
- Selection sort analysis
- Insertion sort pseudocode and analysis

# Syllabus

- Course overview
  - A study of techniques used to design algorithms for complex computational problems that are efficient in terms of time and memory required during execution. Students will also learn the techniques used to evaluate an algorithm's efficiency. Topics include advanced sorting techniques, advanced data structures, dynamic programming, greedy algorithms, amortized analysis, graph algorithms, network flow algorithms, and linear programming if time permits.

# Syllabus

- Students will
  - Be exposed to a variety of existing algorithms and techniques to develop algorithms
  - Learn to design / develop an algorithm for a given computational problem
    - Maybe by noticing similarity to a learned algorithm/problem or existing technique
  - Learn how to analyze its running time

# Asymptotic notation

- Which is upperbound, lowerbound, tight bound
- Big O
- Big Theta
- Big Omega

# Asymptotic notation

- Definitions of Big O, Big Omega and Big Theta on the board.
- Note that these define sets of functions, but we typically say “is” instead of “is in the set”
- Because we cannot do better than n for findMax, the overall (including best and worst cases) running time of findMax is Big Theta (n) – it is an asymptotically **tight bound**
  - Notice that we must compare each element to the maxSoFar and since there are n elements we cannot do better than n-1 compares

# Asymptotic notation

- Why use asymptotic behavior for efficiency?
  - In general ...
  - Ignores small inputs (small values of n) because we may be able to create special purpose algorithms if the input is expected to be small
  - Ignore constants
    - Faster machines can combat constant factors
    - Faster machines cannot combat poor asymptotics

# Asymptotic notation

- Analyze space of findMax
  - Inputs are the list and the size of the list
  - Additional space for maxSoFar and i
  - Only care about the additional space required beyond the inputs for space analysis
  - Space complexity of findMax is Big Theta (1)

# Asymptotic notation

- Let's use the definition of O and Big Omega and Big Theta in an example to determine the asymptotics of some particular function (other than the simple one for findMax)

# Asymptotic notation

- Idea of tight bound vs. not tight bound
- e.g.
- $2*n^2 = O(n^2)$ is asymptotically tight
- $2*n = O(n^2)$ is NOT asymptotically tight (but it is correct to say)
- So, O may or may not be asymptotically tight
- One can use little o to denote an upper bound that is NOT asymptotically tight
- Definition on the board
- Note: $2*n = o(n^2)$ BUT $2*n^2 != o(n^2)$

# Asymptotic notation

- Use little o for an upper bound that is not asymptotically tight
- Definition on the board

# Asymptotic notation

- Use little omega for lower bounds that are not asymptotically tight
- Definition on the board

# Arithmetic Series

- Let me introduce Arithmetic Series on the board.
- This is one of 3 sums you should memorize.
- The other two being Geometric and Harmonic.
- See Appendix A of our text (pgs. 1146, 1147)

# Arithmetic Series

Let's prove the sum is big $O(n^2)$ on the board

I leave you to prove that it is also Big Omega of $n^2$

Together, they mean that it is Big Theta of $n^2$

# Sorting algorithms

- Reminder of what Selection Sort and Insertion Sort do (online animation).
- Let's discuss Selection Sort.
- Keep current "first".
- Continually find min element index in list and swap with the current "first".
  - How long does 1 find min take?
  - How many times do we do find min?
  - Are there any best or worst case situtations? All the same?

# Sorting Algorithms

- Insertion Sort pseudocode on the board with run time analysis.
- Are there any best or worst case situtations? All the same?

# Sorting Algorithms

- Let's look at an instance of Merge Sort and see how it works.
- Are there any best or worst case situtations? All the same?
- Let's see a recursion tree for n elements of MergeSort.
  - How many elements processed in each level?
  - How many levels?