Announcements

- Program 6 will be released tonight
  - It will be open ended
  - I am still working on the due date
- Still working on grading everything
  - Midterm 2 and Program 4 should be finished sometime tomorrow
- Last lecture on new material is on 12/9
- The last class will be on 12/11 and will serve strictly as a review session
  - I highly recommend you go...

Algorithm Analysis

An **algorithm** is a sequence of steps for accomplishing a task.

An algorithms **runtime** is the time the algorithm takes to execute.

An algorithm typically uses a number of steps proportional to the size of the input, for example:

- For a list with 32 elements, linear search requires at most 32 comparisons: 1 comparison if the search key is found at index 0, 2 if found at index 1, and so on, up to 32 comparisons if the the search key is not found.

Algorithm Analysis (cont.)

- For a list with N elements, linear search thus requires at most N comparisons. The algorithm is said to require "on the order" of N comparisons.

Algorithm Analysis (cont.)

Upper and Lower bounds

An algorithm with runtime complexity T(N) has a lower bound and an upper bound

- T(N) is a function representative of an algorithms runtime as N grows
- **Lower bound**: A function f(N) that is **≤ the best case T(N)**, for all values of N ≥ 1.
- **Upper bound**: A function f(N) that is **≥ the worst case T(N)**, for all values of N ≥ 1.

Remember ‘N’ is the number of elements your algorithm might deal with...

Algorithm Analysis (cont.)

“Best case T(N)” means: “The behavior in which your algorithm experiences the smallest amount of runtime growth as ‘N’ gets bigger”

A less accurate (but easier) way to say this is: “The case in which the time it takes your algorithm to execute, is at its **smallest**”

Algorithm Analysis (cont.)

“Worst case T(N)” means: “The behavior in which your algorithm experiences the largest amount of runtime growth as ‘N’ gets bigger”

A less accurate (but easier) way to say this is: “The case in which the time it takes your algorithm to execute, is at its **largest**”

Algorithm Analysis (cont.)

In strict mathematical terms, upper and lower bounds do not relate to best or worst case runtime complexities.

In the context of algorithm complexity analysis, the terms are often used to collectively bound all runtime complexities for an algorithm.

Therefore, the terms are used in this section such that the upper bound is ≥ the worst case T(N) and the lower bound is ≤ the best case T(N).

Algorithm Analysis (cont.)

Let’s look at an example...

Algorithm Analysis (cont.)

An additional simplification can factor out the constant from a bounding function, leaving a function that categorizes the algorithm's growth rate.

Ex: Instead of saying that an algorithm's runtime function has an upper bound of $30N^2$, the algorithm could be described as having a worst case growth rate of N^2

Asymptotic notation is the classification of runtime complexity that uses functions that indicate only the growth rate of a bounding function.

Algorithm Analysis (cont.)

Three **asymptotic notations** are commonly used in complexity analysis:

- ***O notation*** (Big Oh) provides a growth rate for an algorithm's upper bound.
- ***Ω notation*** (Big Omega) provides a growth rate for an algorithm's lower bound.
- ***Θ notation*** (Big Theta) provides a growth rate that is both an upper and lower bound.

Let’s take a look at the formal definition

Algorithm Analysis (cont.)

Big O notation is a mathematical way of describing how a function (running time of an algorithm) generally behaves in relation to the input size.

In **Big O notation**, all functions that have the same growth rate (as determined by the highest order term of the function) are characterized using the same Big O notation.

In essence, all functions that have the same growth rate are considered **equivalent** in Big O notation.

Let's look at an example...

Algorithm Analysis (cont.)

One consideration in evaluating algorithms is that the efficiency of the algorithm is most critical for **large input sizes.**

Small inputs are likely to result in fast running times because N is small, so efficiency is less of a concern.

Let’s look at an example...

Algorithm Analysis (cont.)

Let’s look at some common Big O complexities...

Algorithm Analysis (cont.)

To analyze how runtime of an algorithm scales as the input size increases, we first determine how many operations the algorithm executes for a specific input size, N. Then, the big-O notation for that function is determined.

Algorithm runtime analysis often focuses on the **worst-case runtime** complexity.

The ***worst-case runtime*** of an algorithm is the runtime complexity for an input that results in the longest execution.

Other runtime analyses include best-case runtime and average-case runtime. Determining the average-case runtime requires knowledge of the statistical properties of the expected data inputs.

Algorithm Analysis (cont.)

Let’s look at an example of determining worst case runtime...

Algorithm Analysis (cont.)

Constant Time Operations

For algorithm analysis, the definition of a single operation does not need to be precise.

An operation can be any statement (or constant number of statements) that has a **constant runtime complexity**, O(1).

Since constants are omitted in big-O notation, any constant number of constant time operations is O(1).

So, precisely counting the number of constant time operations in a finite sequence is not needed.

Algorithm Analysis (cont.)

Let’s take a look at constant runtime analysis

Algorithm Analysis (cont.)

Runtime Analysis of nested loops

Runtime analysis for nested loops requires summing the runtime of the inner loop over each outer loop iteration.

The resulting summation can be simplified to determine the big-O notation.

Let’s look at an example...

Algorithm Analysis (cont.)

We can now look runtime complexity for the searching and sorting algorithms we previously covered...