

Algorithm Analysis

Yuck ☹️

Introduction

- ◆ Algorithm analysis is the practice of examining code to objectively decide which code is better.
- ◆ Better – Which code uses less resource.
- ◆ Resource – Something which computers have and users want, be it processor time, memory, etc.

How do you do this?

- ◆ There are a set of very formal rules for extremely precise algorithm analysis.
- ◆ In some cases this is necessary to determine whether one algorithm is better than another.
- ◆ For our purposes we will not need to do this.

big Oh

- ◆ We use a notation known as big Oh notation.
- ◆ This is a way to quantify the running time of an algorithm.
- ◆ The same rules can be applied to actual code.

Rules for big Oh Analysis

1. We assume an arbitrary time unit.
2. Running of each of the following type of statement takes time $T(1)$: [*omitting the arithmetic operators*]
 1. assignment statement
 2. I/O statement
 3. Boolean expression evaluation
 4. function return
3. Running time of a selection statement (if, switch) is $T(1)$ for the condition evaluation + the maximum of the running times for the individual clauses in the selection.

More Rules

4. Loop execution time is the time for the loop setup (initialization & setup) + the sum, over the number of times the loop is executed, of the body time + time for the loop check and update operations.
† Always assume that the loop executes the maximum number of iterations possible
4. Running time of a function call is $T(1)$ for function setup + the time required for the execution of the function body.
5. Running time of a sequence of statements is the largest time of any statement in the sequence.

Example

```
for (i = 0; i < n-1; i++) //What is the big Oh?
```

```
{  
  for (j = 0; j < i; j++)  
  {  
    array[i][j] = 0;  
  }  
}
```

The first loop is runs from 0 to n-1. The second loop runs from 0 to i. In general this has a big O of $O(n^2)$

General Rules

- ◆ A normal loop has big Oh, $O(n)$
- ◆ A doubly nested loop has big Oh, $O(n^2)$
- ◆ A triply nested loop has big Oh, $O(n^3)$
- ◆ You can get better times, e.g. $O(\log n)$
- ◆ How?
 - Binary Search is $O(\log n)$
 - Anytime anything is halved on each iteration, you usually get $O(\log n)$

What's Better

