## Algorithm Analysis

Chapter 2 Overview

- Definitions of Big-Oh and Other Notations
- Common Functions and Growth Rates
- Simple Model of Computation
- Worst Case vs. Average Case Analysis
- How to Perform Analyses
- Comparative Examples



1. Why do we analyze algorithms?

2. How do we measure the efficiency of an algorithm?

A. Time it on my computer.

B. Compare its time to that of another algorithm that has already been analyzed.

C. Count how many instructions it will execute for an arbitrary input data set.

Suppose there are **n** inputs.

We'd like to find a **time function T(n)** that shows how the execution time depends on **n**.

$T(n) = 3n + 4$  $T(n) = e^n$  $T(n) = 2$



## "Big-Oh"

T(N) = O(f(N)) if there are positive constants c and n0 such that $T(N) \le cf(N)$ when $N \ge n0$.

We say "T(N) has order f(N)."

We try to simplify T(N) into one or more common functions.

Ex. 1 $T(N) = 3N + 4$
T(N) is linear. Intuitively, f(N) should be N.

More formally,
$T(N) = 3N + 4 \le 3N + 4N, \ N \ge 1$
$T(N) \le 7N, \ N \ge 1$
So T(N) is of order N.



## Common Functions to Use

- O(1) constant
- O(log n) log base 2
- O(n) linear
- O(n log n)
- $O(n^2)$ quadratic
- $O(n^3)$ cubic
- $O(2^n)$ or $O(e^n)$ exponential

- O(n+m)
- O(n m)
- $O(n^m)$

Suppose we get $T(N) = 4N^2 + 3N + 6$.

Is T(N) $O(N^2)$?

Is T(N) $O(N^3)$?



Generally, we look for the smallest f(N) that bounds T(N).

We want a common function that is a least upper bound.

If $T(N) = c_k N^k + c_{k-1} N^{k-1} + \ldots + c_0$ .

$T(N) = O(N^k)$.

$N^k$ is the dominant term.



## Complexity Analysis

| | |
|---|---|
| Step 1. Counting | T(N) |
| Step 2. Simplifying | O(f(N)) |

```
int sumit( int v[ ],  int num)  {
  sum = 0;                                c1
  for (i = 0;  i  <  num;  i++)           c2*num
    sum = sum + v[i];                     c3*num
  return sum }                            c4
```

T(num) = (c2 + c3)* num + (c1 + c4)
= k1 * num + k2
= O(num)

```
int sumit(int v[ ], int num)
  if (num == 0)  return 0;                              c1
  else return(v[num-1] + sumit(v,num-1)) }               ?
                  |                |
                 c2            T(num-1)
```



Consecutive Loops:

```
for (i = 0; i< n; i++)    A[i] = 0;

for (j = 0; j < m; j++)  B[j] = 0;
```

Nested Loops:

```
for (i = 0; i< n; i++)
 for (j = 0; j < m; j++)
    A[i,j] = 0;
```



Try this one:

```
string t (int n)
  {
  if (n == 1) return '(1) ';
  else return '(' || n || t(n - 1) || t(n - 1) || ') '
  }
```

where || is the string concatenation operator



Average vs. Worst-Case Analysis

Usually we do worst-case analysis.

But average-case analysis can be useful, too.

Ex. Inserting a value in a list stored in an array of **n** elements.

How many elements must be moved?