Program Complexity

Big-O Notation

CSCI 3700 — Data Structures and Objects

Department of Computer Science and Information Systems
Youngstown State University

Robert W. Kramer

Outline

1. Timing Programs
   - Bad Timing Measures
   - Counting Statements
2. Big-O Notation
   - Big-O Introduction

Bad Performance Measures

"Wall-clock" time

Sensitive to system load

CPU cycles

Varies from machine to machine

Machine instruction count

Varies by language and programming style

Counting Statements

Counting to ten

- Line 4:
  $1 + 11 + 10 = 22$
- Line 5: 10
- Line 7: 1
- Total: 33

```
1 int ten(int data[], int n) {
2   int j;
3
4   for (j=0;j<10;j++)
5     cout << j << endl;
6
7   return 0;
8 }
```

Counting Statements

Finding the largest element

- Line 4: 1
- Line 5: $2n$
- Line 6: $n-1$
- Line 7: $0 \ldots n-1$
- Line 9: 1
- Total: $3n+1 \ldots 4n$

```
1  int findMax(int data[], int n) {
2    int j,m;
3
4    m = 0;
5    for (j=1;j<n;j++)
6      if (data[j] > data[m])
7        m = j;
8
9    return m;
10 }
```

Counting Statements

Sorting a list

- Line 4: $2n$
- Line 5: $n-1$
- Line 6: $2\sum_{i=1}^{n-1}(i+1)$
- Line 7: $\sum_{i=1}^{n-1} i$
- Line 8: $0 \ldots \frac{n^2-n}{2}$
- Lines 10–12: $3(n-1)$ total
- Total: $1.5n^2 + 6.5n - 6 \ldots$ $2n^2 + 6n - 6$

```
1  void sort(int data[], int n) {
2    int s,j,m,tmp;
3
4    for (s=n-1;s>0;s--) {
5      m = 0;
6      for (j=1;j<=s;j++)
7        if (data[j] > data[m])
8          m = j;
9
10     tmp = data[m];
11     data[m] = data[s];
12     data[s] = tmp;
13   }
14 }
```

Why Bother?

- Counting statements is hard
  - Imagine counting a large function
- Counting statements is more accurate
  - Machine and language effects minimized
  - Better counting methods exist
- Additional benefits
  - Compare functions / programs

A Better Approach

- Big-O notation
  - Easily approximates count
  - Shows trends with large data sizes
  - Allows code / algorithm comparison
- How it works
  - Concept derived from discrete mathematics
  - Simple form needed here — focus on loops

Types Of Loops

Simple loops

- Linear loop
  - Key: ++ or --
  - $O(n)$
- Logarithmic loop
  - Key: cut data in half
  - $O(\lg n)$
    $\lg n = \log_2 n$

Nested loops

- Log-linear loop
  - Linear inside logarithmic (or vice versa)
  - $O(n \lg n)$
- Dependent quadratic loop
  - Linear inside linear
  - Inner counter depends upon outer counter
  - $O(n^2)$
- Quadratic loop
  - Linear inside linear
  - No dependency between loops
  - $O(n^2)$

Benefits and Caveats

Benefits

- Easy to determine
- Allows code / algorithm comparison
- Show trend as amount of data increases

Caveats

- Not accurate for small data size
- Not exact
- Can't distinguish two programs / algorithms with same $O(\cdot)$ value

Comparing Big-O Values

- Table shows common values
  - Fastest at top
- Each value significantly faster than values beneath it
- Two items, same big-O value
  - Neither significantly faster

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n \log n) = O(\lg(n!))$
- $O(n^2)$
- $O(n^3)$
- $O(n^4)$
- $O(2^n)$
- $O(3^n)$
- $O(n!)$
- $O(n^n)$

Summary

- Many bad timing measures
- Counting statements better, but hard
- Big-O notation provides easy approximation