

Algorithms & Data Structures

Dr. Jeffrey Blessing
Milwaukee School of Engineering

1

Agenda

- Syllabus
- On-line notes
- Weekly Quizzes
- Programs
- Homeworks
- Exams
 - Midterm (Wk. 6)
 - Final (Wk. 11)
- C++ Language
- Standard Library
 - STL now included
- MSVC++ 5.0
 - STL by P.J. Plauger
- STL originally by
 - Alex Stepanov, et. al.
 - HP, SGI
- Minor variations

2

Algorithms

- An algorithm is a function of the class of problems that it addresses.
- It consists of a finite number of steps which, when executed properly, yields a solution to the problem that is guaranteed to be correct.
- Time/Space complexity are measures used to compare the relative performance of algorithms that address the same problem.

3

Algorithm Measurement

- Time complexity is considered more important than space complexity.
- Comparing algorithms based on time is problematic because:

4

Algorithm Measurement

- Time complexity is considered more important than space complexity.
- Comparing algorithms based on time is problematic because:
 - machines vary in speed.

5

Algorithm Measurement

- Time complexity is considered more important than space complexity.
- Comparing algorithms based on time is problematic because:
 - machines vary in speed.
 - machines vary in architecture.

6

Algorithm Measurement

- Time complexity is considered more important than space complexity.
- Comparing algorithms based on time is problematic because:
 - machines vary in speed.
 - machines vary in architecture.
 - efficiency is a function of the programmer.

7

Algorithm Measurement

- Time complexity is considered more important than space complexity.
- Comparing algorithms based on time is problematic because:
 - machines vary in speed.
 - machines vary in architecture.
 - efficiency is a function of the programmer.
- A standard machine model is needed.

8

Scalar Machine Model

- Any fundamental operation (+, -, *, /, ^, =, <, <=, >, >=, <>, and, or, not, etc.) takes one time unit.

9

Scalar Machine Model

- Any fundamental operation (+, -, *, /, ^, =, <, <=, >, >=, <>, and, or, not, etc.) takes one time unit.
 - If necessary, count *, /, ^ separately.

10

Scalar Machine Model

- Any fundamental operation (+, -, *, /, ^, =, <, <=, >, >=, <>, and, or, not, etc.) takes one time unit.
 - If necessary, count *, /, ^ separately.
- Only unary or binary operations can be performed in unit time.

11

Scalar Machine Model

- Any fundamental operation (+, -, *, /, ^, =, <, <=, >, >=, <>, and, or, not, etc.) takes one time unit.
 - If necessary, count *, /, ^ separately.
- Only unary or binary operations can be performed in unit time.
- Only scalar operands can be handled in unit time.

12

Analysis Example

```
unsigned int sum (int n)
{
    unsigned int i,partial_sum;
    partial_sum = 0;
    for (i=1; i<=n; i++)
        partial_sum += i*i*i;
    return (partial_sum);
}
```

13

Analysis Example

```
unsigned int sum (int n)
{
    unsigned int i,partial_sum;
    partial_sum = 0;
    for (i=1; i<=n; i++)
        partial_sum += i*i*i;
    return (partial_sum);
}
```

Declarations count for no time.

14

Analysis Example

```
unsigned int sum (int n)
{
    unsigned int i,partial_sum;
    partial_sum = 0;
    for (i=1; i<=n; i++)
        partial_sum += i*i*i;
    return (partial_sum);
}
```

Simple statements = 1 time unit each.

15

Analysis Example

```
unsigned int sum (int n)
{
    unsigned int i,partial_sum;
    partial_sum = 0;
    for (i=1; i<=n; i++)
        partial_sum += i*i*i;
    return (partial_sum);
}
```

Two multiply, one add each iteration = 3 time units.

16

Analysis Example

```
unsigned int sum (int n)
{
    unsigned int i,partial_sum;
    partial_sum = 0;
    for (i=1; i<=n; i++)
        partial_sum += i*i*i;
    return (partial_sum);
}
```

One init, (n+1) tests, n increments = (2n+2) time units.

17

Analysis Example

```
unsigned int sum (int n)
{
    unsigned int i,partial_sum;
    partial_sum = 0;
    for (i=1; i<=n; i++)
        partial_sum += i*i*i;
    return (partial_sum);
}
```

Total = (6n+6) time units. Which is O(n).

18

Analysis Effort

- Detailed analysis
 - Line-by-line breakdown
 - Add up all contributions
- But details vary by machine, etc.
- Really only want the big picture
 - Is running time constant, linear, or ??

19

Big-Oh Notation

- “On the order of ...”
 - For large n , time grows as $f(n)$
 - $(5n+4) = O(n)$
 - Note special use of equal sign!
- Asymptotic behavior
 - Dominant term as problem size grows
- Variable n always represents the problem size.

20

Big-Oh Definition

$$T(n) = O(f(n))$$

if there are constants c
and n_0 such that

$$T(n) \leq cf(n)$$

when

$$n \geq n_0$$

21

Related Definitions

$O(f(n))$ Upper bound (\leq)

$\Omega(f(n))$ Lower bound (\geq)

$\Theta(f(n))$ Upper/lower bound

$o(f(n))$ Upper bound ($<$)

Use only big-oh?

22

Big-Oh of Aggregates

If:

$$T_1(n) = O(f(n))$$

$$T_2(n) = O(g(n))$$

Then:

$$T_1(n) + T_2(n) = \max(O(f(n)), O(g(n)))$$

$$T_1(n) * T_2(n) = O(f(n) * g(n))$$

23

Other Big-Oh Rules

If $T(n)$ is a polynomial in x of degree n :

$$T(n) = O(x^n)$$

For any constant k :

24

Logarithms
 $\log_k n = O(\log n)$

25

Logarithms
 $\log_k n = O(\log n)$
 for constants a, b and variable n:
 $\log_a n = \log_a b \cdot \log_b n$

26

Logarithms
 $\log_k n = O(\log n)$
 for constants a, b and variable n:
 $\log_a n = \log_a b \cdot \log_b n$
 $\frac{\log_a n}{\log_a b} = \log_b n$

27

Logarithms
 $\log_k n = O(\log n)$
 for constants a, b and variable n:
 $\log_a n = \log_a b \log_b n$
 $\frac{\log_a n}{\log_a b} = \log_b n$
 denominator is a constant!

28

Logarithmic Time

- Better than linear
- Constant-time operation ...
- Cuts problem size by fraction
- Example: binary search
 - Cut problem in half with each comparison

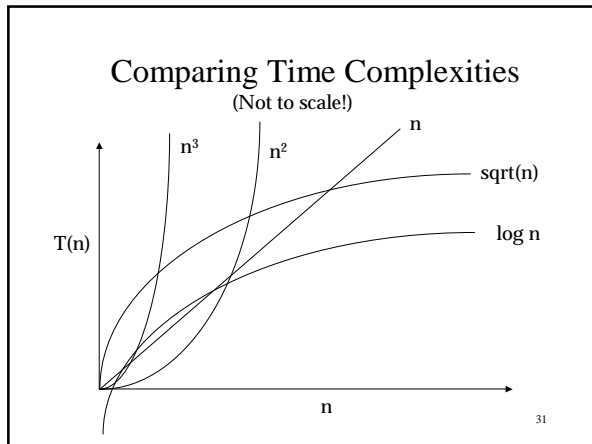
29

Logarithmic Time

- Better than linear
- Constant-time operation ...
- Cuts problem size by fraction
- Example: binary search
 - Cut problem in half with each comparison

$$T(n) = T(n/2) + C = O(\log n)$$

30



- ### Big-Oh Counting Rules
- FOR loops
 - (iterations) * (loop statement(s) time)
 - Nested loops
 - Product of loop sizes (iterations)
 - How many times body is executed
 - Don't forget test time (part of body)
- 32

- ### Big-Oh Counting Rules
- Consecutive Sequential Statements
 - Only the longest one counts
 - IF/ELSE
 - Test time (evaluate test expression)
 - Plus longer branch
 - "true" part or "false" part
- 33

- ### Some Big-Oh Categories
- Constant
 - Sublinear - faster than $O(n)$
 - $\log n$, \sqrt{n} .
 - Linear - $O(n)$
 - Polynomial - $O(n^k)$
 - Exponential - $O(a^n)$
- 34

Does It Really Matter?

$\log n$	1.2×10^{-5} seconds	From <i>Classic Data Structures in C++</i> , by Timothy A. Budd $n=100000$ $\Delta t=1\mu s$
\sqrt{n}	3.2×10^{-4} seconds	
n	0.1 seconds	
$n \log n$	1.2 seconds	
$n\sqrt{n}$	31.6 seconds	
n^2	2.8 hours	
n^3	31.7 years	
2^n	> 1 century	

35

- ### Running Time Bounds
- Worst case
 - Upper bound
 - Best case
 - Lower bound
 - Average
 - Sometimes can prove average value
- 36

Big-Oh Cautions

- Program run only once?
- Problem sizes known small?
- Complex algorithm desirable?
- Speed/space tradeoff?
- Accuracy/stability vs speed?

37

Find: Maximum Subsequence Sum

Find subsequence (in input sequence)
with largest sum.

2, -3, 1.5, -1, 3, -2, -3, 3

Find: Maximum Subsequence Sum

Find subsequence (in input sequence)
with largest sum.

2, -3, 1.5, -1, 3, -2, -3, 3

3.5

Maximum Subsequence Sum

Find subsequence (in input sequence)
with largest sum.

Four different algorithms:

- Simple-minded - $O(n^3)$
- Improved - $O(n^2)$
- Recursive - $O(n \log n)$
- Optimal - $O(n)$

40

$O(n^3)$ Max Subsequence Alg.

```
for (int i=0; i < size-1; i++)
{
  for (int j=i; j < size; j++)
  {
    sum = 0;
    for (int k=i; k <= j, k++)
      sum += seq[k];
    if (sum > max)
    {
      max = sum;
      record(sum, i, j);
    }
  }
}
```

41

$O(n^2)$ Max Subsequence Alg.

```
for (int i=0; i < size-1; i++)
{
  sum = 0;
  for (int j=i; j < size; j++)
  {
    sum += seq[j];
    if (sum > max)
    {
      max = sum;
      record(sum, i, j);
    }
  }
}
```

42

