

Principles of Algorithm Analysis

Biostatistics 615/815

Lecture 4

Homework Notes

- Provide a hard copy including
 - Your Answers
 - Your Code
- Write specific answer to each question
 - Supported by table or graph if appropriate
- Source code
 - Indented and commented, if appropriate

Office Hours

- Friday
 - 8:30 – 10:00 am
 - SPH II, M4132
- Alternatively, e-mail me at:
 - goncalo@umich.edu

Last Week

An Introduction to C

- Strongly typed language
 - Variable and function types set explicitly
- Functional language
 - Programs are a collection of functions
- Rich set of program control options
 - `for`, `while`, `do ...while`, `if` statements
- Compiling and debugging C programs
 - Windows, Linux

Today

- Strategies for comparing algorithms
- Common relationships between algorithm complexity and input data
- Compare two simple search algorithms

Objectives

- Framework for
 - Empirical Testing
 - Theoretical Analysis
- Highlight performance characteristics of algorithms

Specific Questions

- Compare two algorithms for one task
- Predict performance in a new environment
 - If we had a computer that was 10x faster and could store 10x more data, how would approach perform?
- Set values of algorithm parameters

Two Common Mistakes

- Ignore performance of algorithm
 - Shun faster algorithms to avoid complexity in program
 - Instead, wait for simple N^2 algorithms, when $N \log N$ alternatives exist of modest complexity available
- Too much weight on performance of algorithm
 - Improving a very fast program is not worth it
 - Spending too much time tinkering with code is rarely good use of time

Empirical analysis

- Given two algorithms ... which is better?
- Run both
 - Say, algorithm A takes 3 seconds
 - Say, algorithm B takes 30 seconds
- Empirical studies may not always be practical
 - Some algorithms may take too long to run!
 - Other algorithms may take too long to code...

Choices of Input Data

- Actual data
 - Measures performance in use
- Random data
 - Generic approach, may not be representative
- Perverse data
 - Attempt worst case analysis

Limitations of Empirical Analysis

- Quality of implementation
 - Is our favored implementation coded more carefully than another?
- Extraneous factors
 - Compiler
 - Machine
 - Computer system

Limitations of Empirical Analysis

- Requires a working program
- Theoretical analysis is an alternative
 - Estimate potential gains
- Predict effectiveness relative to new algorithms or computers (that may not yet exist)

Theoretical Analysis

- Predict performance of algorithm based on theoretical properties
- “Independent” of actual implementation
- Several constructs occur frequently in algorithm analysis

Limitations of Theoretical Analysis

- Efficiency can depend on compiler
- Efficiency may fluctuate with input data
- Some algorithms are not well understood

The idea...

- Given a code fragment

```
#Find parent of node i  
i = a[i];
```

- Consider how many times it is executed
- But not how long each execution takes

Two typical analyses

- Average-case for random input
- Worst-case
- Are these representative of real world problems?
 - Check with empirical predictions...

The Primary Parameter N

- Examples
 - Number of parameters to likelihood function
 - Number of items in dataset to be processed
 - Number of characters in a string
 - Size of file to be sorted
 - Some other abstract measure of problem size
- With multiple inputs, focus on one at a time, while holding the others constant

Running time as a function of N

$f(N)$	Description	Running time when N doubles...
1	<i>constant</i>	-
$\log N$	<i>logarithmic</i>	constant increase
N	<i>linear</i>	doubles
$N \log N$	<i>log-linear</i>	more than doubles
N^2	<i>quadratic</i>	increases fourfold
N^3	<i>cubic</i>	increases eightfold
2^N	<i>exponential</i>	running time squares

Running time as a function of N

- Multiple terms may be involved
 - e.g. $N + N \log N$
- Typically, we ignore
 - Smaller terms
 - Constant coefficient
 - Focus on inner loop
- In rare cases, smaller terms and constant coefficient will be important

Time to Solve Large Problem

operations per second	Problem Size $N = 1,000,000$		
	N	$N \log N$	N^2
10^6	seconds	minutes	months
10^9	instant	instant	hours
10^{12}	instant	instant	seconds

Time to Solve Huge Problem

operations per second	Problem Size $N = 1,000,000,000$		
	N	$N \log N$	N^2
10^6	hours	days	never
10^9	seconds	minutes	centuries
10^{12}	instant	instant	months

Application

- Analysis of two search algorithms
- Consider a set of items
 - Evaluate functions to decide whether a particular item is present...

Sequential Search

```
int search(int a[], int value, int start, int stop)
{
    // Variable declarations
    int i;

    // Search through each item
    for (i = start; i <= stop; i++)
        if (value == a[i])
            return i;

    // Search failed
    return -1;
}
```

Sequential Search Properties

- Algorithm:
 - Look through array sequentially, until we find a match
- Average cost
 - If match found: $N/2$
 - If match not found: N
- Actual cost depends on fraction of successful searches

Better Sequential Search

- If items are sorted...
- Stop unsuccessful search early, when we reach item with higher value
 - Cost for unsuccessful searches is now $N/2$
- Overall, algorithm is still $O(N)$

Binary Search

```
int search(int a[], int value, int start, int stop)
{
    while (stop >= start)
    {
        // Find midpoint
        int mid = (start + stop) / 2;

        // Compare midpoint to value
        if (value == a[mid])
            return mid;

        // Reduce input in half !...
        if (value > a[mid])
            { start = mid + 1; }
        else
            { stop = mid - 1; }
    }

    // Search failed
    return -1;
}
```

Binary Search Properties

- Algorithm:
 - Halve number of items to consider with each comparison
- Worst-case cost
 - Maximum cost is never greater than $\log_2 N$
- Much better than sequential search, but even better methods exist!

Sequential vs. Binary Search

N	<u>M = 1,000</u>		<u>M = 10,000</u>		<u>M = 100,000</u>	
	S	B	S	B	S	B
125	1	1	13	2	130	20
250	3	0	25	2	251	22
500	5	0	49	3	492	23
1250	13	0	128	3	1276	25
2500	26	1	267	3	*	28

Timings in seconds, for M searches in table of N elements

Big-Oh Notation

- Algorithm is $O(N)$ or $O(N \log N)$
 - Common statement
 - What does it mean?
- Summarizes performance for large N
- Focuses on leading terms of expression describing running time

Big-Oh Notation

- Consider function $g(N)$
- It is said to be $O(f(N))$
- If there exist c_0 and N_0 such that:
 - $N > N_0$ implies $c_0 f(N) > g(N)$

From N to Running Time...

- Common relationships
 - N^2
 - $\log N$
 - $N \log N$
 - N
- Describe examples of how these arise
- Cost of running program is C_N

$O(N^2)$

- Loop through input successively, eliminate one item at a time

$$\begin{aligned}C_N &= C_{N-1} + N \quad \text{for } N \geq 2, C_1 = 1 \\ &= C_{N-2} + (N-1) + N \\ &\dots \\ &= 1 + 2 + \dots + (N-1) + N \\ &= \frac{N(N+1)}{2}\end{aligned}$$

$O(\log N)$

- Recursive program, halves input in one step

$$C_{2^n} = C_{2^{n-1}} + 1 \quad \text{for } N \geq 2, C_1 = 1$$

$$= C_{2^{n-2}} + 1 + 1$$

$$= C_{2^{n-3}} + 3$$

...

$$= C_{2^0} + n$$

$$= n + 1$$

$$N = 2^n$$

$O(N \log N)$

- Recursive program, processes each item, splits input into two halves, examines each one...

$$C_N = 2C_{N/2} + N \quad \text{for } N \geq 2, C_1 = 0$$

$$C_{2^n} = 2C_{2^{n-1}} + 2^n$$

$$\frac{C_{2^n}}{2^n} = \frac{2C_{2^{n-1}} + 2^n}{2^n}$$

$$= \frac{C_{2^{n-1}}}{2^{n-1}} + 1$$

$$= \frac{C_{2^{n-2}}}{2^{n-2}} + 1 + 1$$

...

$$= n$$

$O(2N)$

- Halves input, must examine each item...

$$C_N = C_{N/2} + N \quad \text{for } N \geq 2, C_1=1$$

$$= N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \dots$$

$$\approx 2N$$

Summary

- Outline principles for analysis of algorithms
- Introduced some common relationships between N and running time
- Described two simple search algorithms

Further Reading

- Read chapter 2 of Sedgewick

Tip of the Day: Defensive Programming

- Document code and programs
 - Indicate intended purpose
 - Specify required inputs
 - Always indicate author
- Check for error conditions