Lecture 4: Analysis of Algorithms

BT 3051 – Data Structures and Algorithms for Biology

Karthik Raman
Department of Biotechnology
Bhupat and Jyoti Mehta School of Biosciences
Indian Institute of Technology Madras

Introduction

Which of these algorithms is better?

- Two different algorithms on two different computers ...

n (list size)	Computer A run-time (ns)	Computer B run-time (ns)
16	8	100,000
63	32	150,000
250	125	200,000
1,000	500	250,000

Searching for an element in a list

- Given a list of integers, find if an input integer belongs in the list

Linear Search

```
def linear_search(array,value):
    ''' (list,object) --> int
    Return the index of the first occurrence of an
    object in a list. Returns -1 if the number is
    absent in the list.
    >>> linear_search([1,2,3,4,5],1)
    0
    >>> linear_search([1,2,3,4,5],5)
    4
    >>> linear_search([1,2,3,4,5],6)
    -1
    '''
    n = len(array)

    for i in range(n):
        if array[i] == value:
            return i

    return -1
```

Binary Search

```
def binary_search(array,value):
    ''' (list,object) --> int
    Return the index of the first occurrence of an
    object in a sorted list. Returns -1 if the number
    is absent in the list.
    '''
    N = len(array)
    i = 0
    j = N  - 1
    while i != j+1:
        mid = (i+j)//2
        if array[mid] < value:
            i = mid+1
        else:
            j = mid-1

    if 0 <= i < N and array[i] == value:
        return i
    else:
        return -1
```

Comparison of Times

```
import time
from linear_search import *
from binary_search import *
def time_it(search_function, L, v):
    '''(function,object,value) --> number
    Time (in millseconds) how long it takes to run the
    search_function to find the value v in list L.
    '''
    t_begin = time.perf_counter()
    search_function(L,v)
    t_end = time.perf_counter()
    return int((t_end - t_begin) * 1000)

for prob_sz in [10**7]:
    test_list = list(range(prob_sz))
    for search_func in [linear_search,binary_search]:
        for test_value in [0,prob_sz//2, prob_sz]:
            t=time_it(search_func,test_list,test_value)
            print (search_func.__name__, prob_sz,
                                          test_value, t)
```

Analysis of Algorithms

- Empirical Analysis
- Mathematical Analysis

Analysis of Algorithms

Analysis of Algorithms:
Empirical Analysis

Empirical Analysis

- Involves measuring (timing) program performance ...
- Followed by 'curve fitting'!
- Measure things like $T(2n)/T(n)$
- Easy to conduct *experiments*
- Difficult to get precise measurements — many confounding factors e.g. hardware, software, state of the system etc.

Empirical Analysis

- Involves measuring (timing) program performance ...
- Followed by ‘curve fitting’!
- Measure things like $T(2n)/T(n)$
- Easy to conduct *experiments*
- Difficult to get precise measurements — many confounding factors e.g. hardware, software, state of the system etc.

Empirical Analysis

- Involves measuring (timing) program performance ...
- Followed by 'curve fitting'!
- Measure things like $T(2n)/T(n)$
- Easy to conduct *experiments*
- Difficult to get precise measurements — many confounding factors e.g. hardware, software, state of the system etc.

Empirical Analysis

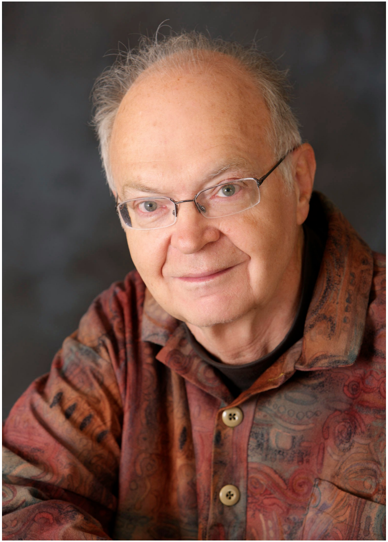
- Involves measuring (timing) program performance ...
- Followed by 'curve fitting'!
- Measure things like $T(2n)/T(n)$
- Easy to conduct *experiments*
- Difficult to get precise measurements — many confounding factors e.g. hardware, software, state of the system etc.

Empirical Analysis

- Involves measuring (timing) program performance ...
- Followed by 'curve fitting'!
- Measure things like $T(2n)/T(n)$
- Easy to conduct *experiments*
- Difficult to get precise measurements — many confounding factors e.g. hardware, software, state of the system etc.

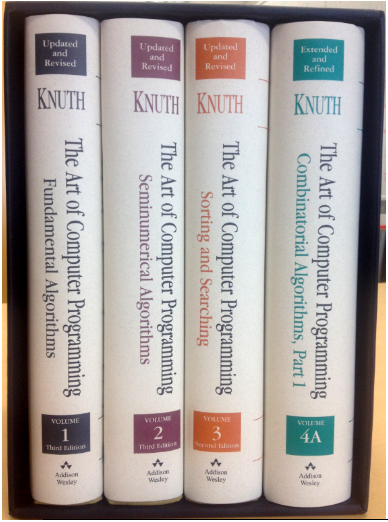
Analysis of Algorithms: Mathematical Analysis

Donald Ervin Knuth



The 1974 A.M. Turing Award was presented to Professor Donald E. Knuth of Stanford University for a number of major contributions to analysis of algorithms and the design of programming languages, and in particular for his most significant contributions to the "art of computer programming" through his series of well-known books. The collections of technique, algorithms, and relevant theory in these books have served as a focal point for developing curricula and as an organizing influence on computer science.

Donald Ervin Knuth



TAOCP emphasized a mathematical approach for comparing algorithms to find out how good a method is. Knuth says that when he decided this approach, he suggested to his publishers renaming the book The Analysis of Algorithms, but they said "We can't; it will never sell." *Arguably, the books established analysis of algorithms as a computer science topic in its own right.* ***Knuth has stated that developing analysis of algorithms as an academic subject is his proudest achievement.***

http://amturing.acm.org/award_winners/knuth_1013846.cfm
Image credit: http://www.preining.info/

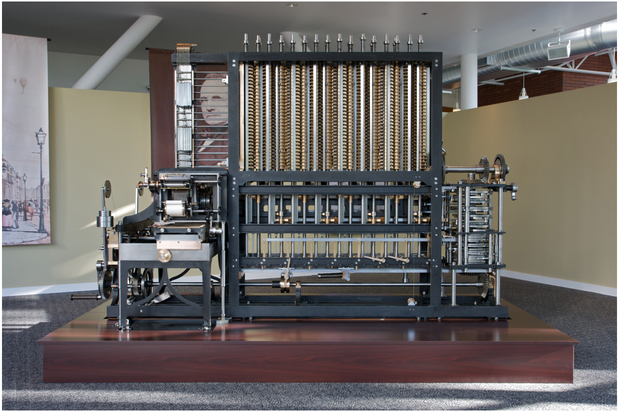
Mathematical Analysis

- $T_{\text{total}} = \sum_i \text{frequency}_i \cdot \text{cost}_i$
- Frequency depends on Algorithm/Program and the data input
- Cost depends on machine
- Involves careful analysis of algorithm performance
- Accurate mathematical models can be constructed in principle
- Important to break down a program into '*basic operations*'[a]

[a] http://en.wikipedia.org/wiki/MMIX

Order of Growth

Visionary Thinking



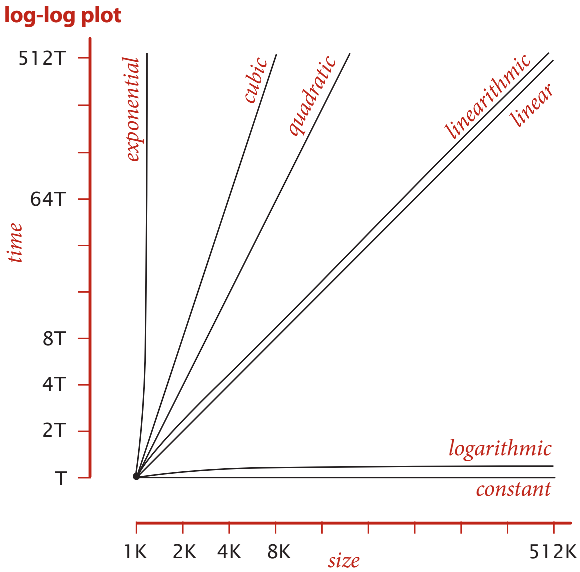
"As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise — by what course of calculation can these results be arrived at by the machine in the shortest time?"

— *Charles Babbage (1864)*

Image Courtesy: Wikipedia

Common Order of Growth Classifications

Reference: Robert Sedgewick



Common Order of Growth Classifications

Reference: Robert Sedgewick

order of growth	name	typical code framework	description	example	T(2N) / T(N)
1	constant	`a = b + c;`	statement	add two numbers	1
$\log N$	logarithmic	`while (N > 1) { N = N / 2; ... }`	divide in half	binary search	~ 1
N	linear	`for (int i = 0; i < N; i++) { ... }`	loop	find the maximum	2
$N \log N$	linearithmic	[see mergesort lecture]	divide and conquer	mergesort	~ 2
N^2	quadratic	`for (int i = 0; i < N; i++)` `for (int j = 0; j < N; j++)` `{ ... }`	double loop	check all pairs	4
N^3	cubic	`for (int i = 0; i < N; i++)` `for (int j = 0; j < N; j++)` `for (int k = 0; k < N; k++)` `{ ... }`	triple loop	check all triples	8
2^N	exponential	[see combinatorial search lecture]	exhaustive search	check all subsets	T(N)

Order of Growth: Why do we obsess?

Reference: Robert Sedgewick

growth rate	problem size solvable in minutes			
	1970s	1980s	1990s	2000s
1	any	any	any	any
log N	any	any	any	any
N	millions	tens of millions	hundreds of millions	billions
N log N	hundreds of thousands	millions	millions	hundreds of millions
N^2	hundreds	thousand	thousands	tens of thousands
N^3	hundred	hundreds	thousand	thousands
2^N	20	20s	20s	30

Order of Growth: Why do we obsess?

Assumption: Processor that can execute 10^6 high-level instructions per second

Size	n	$n \lg n$	n^2	$n^2 \lg n$	n^3	1.5^n	2^n	$n!$
10	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	4 s
30	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	18 min	10^{19} y
50	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	11 min	36 y	10^{51} y
100	< 1 s	< 1 s	< 1 s	< 1 s	1 s	12,892 y	10^{17} y	∞
10^3	< 1 s	< 1 s	1 s	10 s	17 min	∞	∞	∞
10^4	< 1 s	< 1 s	2 min	23 min	12 d	∞	∞	∞
10^5	< 1 s	2 s	3 h	2 d	32 y	∞	∞	∞
10^6	1 s	20 s	12 d	231 d	31,710 y	∞	∞	∞
10^9	17 min	9 h	31,710 y	10^6 y	10^{14} y	∞	∞	∞
10^{12}	12 d	2 y	10^{11} y	10^{13} y	10^{23} y	∞	∞	∞

Adapted from Table 2.1 of *Algorithm Design* by Kleinberg and Tardos

Order of Growth: Why do we obsess?

Assumption: Processor that can execute 10^9 high-level instructions per second

Size	n	$n \lg n$	n^2	$n^2 \lg n$	n^3	1.5^n	2^n	$n!$
10	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
30	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	2 s	10^{16} y
50	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	14 d	10^{48} y
100	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	13 y	10^{14} y	∞
10^3	< 1 s	< 1 s	< 1 s	< 1 s	1 s	∞	∞	∞
10^4	< 1 s	< 1 s	< 1 s	2 s	17 min	∞	∞	∞
10^5	< 1 s	< 1 s	10 s	3 min	12 d	∞	∞	∞
10^6	< 1 s	< 1 s	17 min	6 h	32 y	∞	∞	∞
10^9	1 s	30 s	32 y	949 y	10^{11} y	∞	∞	∞
10^{12}	17 min	12 h	10^8 y	10^{10} y	10^{20} y	∞	∞	∞

Which of these algorithms is better?

- Two different algorithms on two different computers …

n (list size)	Computer A run-time (ns)	Computer B run-time (ns)
16	8	100,000
63	32	150,000
250	125	200,000
1,000	500	250,000
…		
1,000,000	500,000	500,000
4,000,000	2,000,000	550,000
16,000,000	8,000,000	600,000
…		
$63{,}072 \times 10^{12}$	$31{,}536 \times 10^{12}$ ns, or 1 year	1,375,000 ns, or 1.375 ms!

Which of these algorithms is better?

- Two different algorithms on two different computers ...

n (list size)	Computer A run-time (ns)	Computer B run-time (ns)
16	8	100,000
63	32	150,000
250	125	200,000
1,000	500	250,000
...		
1,000,000	500,000	500,000
4,000,000	2,000,000	550,000
16,000,000	8,000,000	600,000
...		
$63{,}072 \times 10^{12}$	$31{,}536 \times 10^{12}$ ns, or 1 year	1,375,000 ns, or 1.375 ms!

Which of these algorithms is better?

- Two different algorithms on two different computers ...

n (list size)	Computer A run-time (ns)	Computer B run-time (ns)
16	8	100,000
63	32	150,000
250	125	200,000
1,000	500	250,000
...		
1,000,000	500,000	500,000
4,000,000	2,000,000	550,000
16,000,000	8,000,000	600,000
...		
$63{,}072 \times 10^{12}$	$31{,}536 \times 10^{12}$ ns, or 1 year	1,375,000 ns, or 1.375 ms!

Moral of the Story

Why build better algorithms? Can't we use a super supercomputer instead?

> *"A faster algorithm running on a slower computer will always win for sufficiently large instances! Usually, problems don't have to get that large before the faster algorithm wins."*
>
> *– Steve Skiena*

Moral of the Story

Why build better algorithms? Can't we use a super supercomputer instead?

> *"A faster algorithm running on a slower computer will always win for sufficiently large instances! Usually, problems don't have to get that large before the faster algorithm wins."*
>
> – *Steve Skiena*