

EC504 Algorithms and Data Structure

Rich Brower Tu-Tr 11:00- 12:45

EC504 Super Team Fall 2023

L

					
Burke, John, W	Chen, Huaizhen	Chen, Ziyan	Dhimoyee, Sumatra, Kamal	Harrington, William, Thomas	Jamias, Austin, Gabriel
					
Jin, Yuqi	Karabay, Emre	Kulkarni, Adwait, Deepak	Markowitz, Noah	Oztop, Beste	Panda, Kashyap
					
Roshandel, Alireza	Ruan, Luying	Shum, Hin Lui	Wang, Jiayu	Wang, Sen	Wang, Zhiqing
					
Wilkins, John, Charles	Xie, Eric, C	Xu, Yuzhe	Yang, Shining	Young, Cynthia, Marisol	Zhao, Aowei

Total Students: 24

EC504 Course Organization

- Why Algorithm ==> Data Structures
- CRSL text with Slide Summaries
- Scaling, Math and Empirical Analysis on Simple Cases.
- Use **GitHub** (EC504_2023F), **Slack** and **CCS** and **Unix Tools**
- HW's pencil and paper handed at CCS
- Software delivered CCS — Must run from Makefile.
- Basic Unix environment — useful for computer engineers to know!



Compile and Running Hello World on the SCC

On your laptop use a Unix shell (terminal) and your BU Kerberos password.)

Step #1. `ssh [username]@scc1.bu.edu`

Step #2. `cd /projectnb/ec504rb/students/[username]`

Step #3. `git clone https://github.com/brower/EC504_2023F.git`

Step #4. `mkdir hello_working`

Step #5 `cp EC504_2023F>HelloWorld/* hello_working`

Step #6 `cd hello_working`

Step #7 `make -k`

Step #8 `./hello`

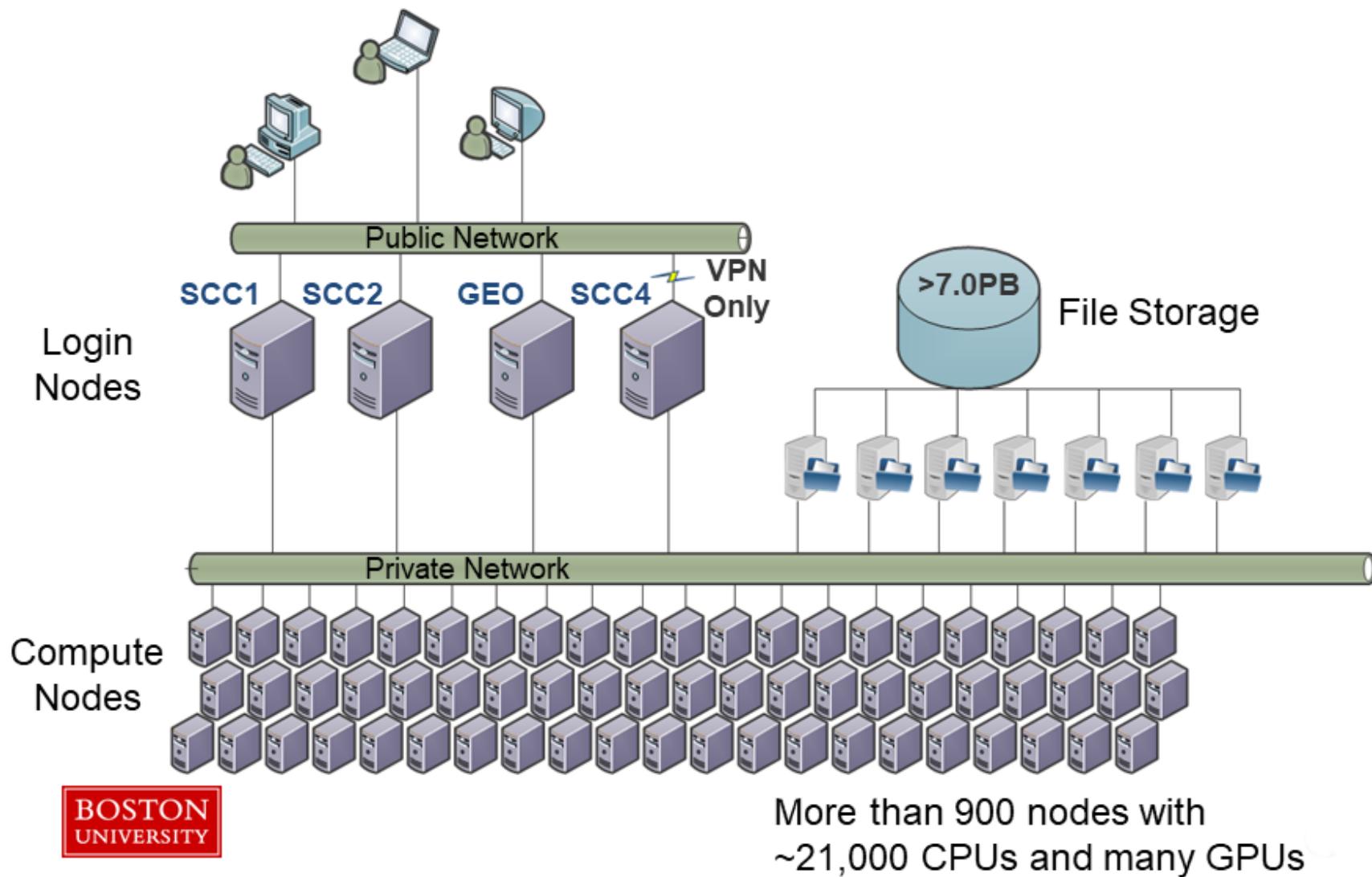


Massachusetts Green High Performance Computing Center

<https://www.bu.edu/tech/support/research/rcs/mghpcc/>



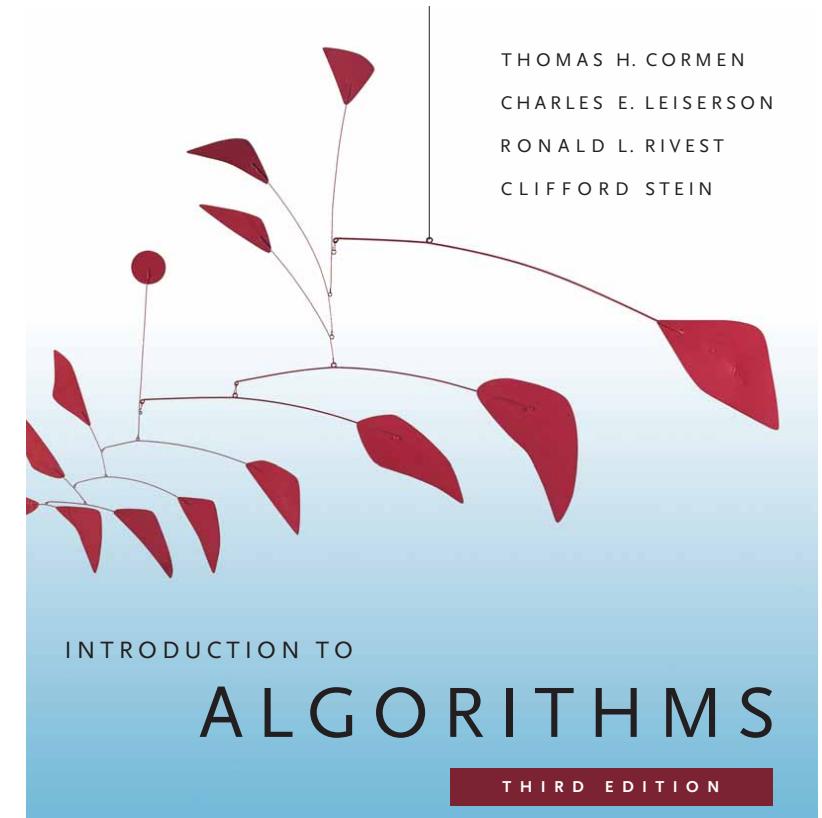
SCC Architecture



Course Organization

- **Text:**
 - Cormen, Leiserson, Rivest & Stein (CLRS), Fundamental text!
“Introduction to Algorithms” 3rd Edition MIT Pres
- **Keynote Slides guide to CLRS**
- **Reference:**
 - Wikipedia!
 - Mark Allen Weiss “Data Structure and Algorithms in C”.
 - UNIX, Makefiles, very basic C/C++ and gnuplot:
- **Grading:**

– HW with Coding Programs:	40%
– Project	20%
– Midterm (up to trees)	20%
– Final (comprehensive)	20%



Course Outline

- Algorithms Analysis CRS 2-4 (5) (HW1?)
 - Definition of Problem Class of Size N
 - Math for large N Asymptotics:
- I. 1-D Data Structures CRS 6,7,8,9 (HW2?)
 - Arrays, Lists, Stacks, Queues CRS 10
 - Searching, Sorting, String Matching, Scheduling
- II. 1.5 D Trees CRLS 12 -`14 (HW3?)
 - BST, AVL,
 - Coding, Union/Join CRLS 18-21, midterm (HW4?)
- III. 2D Graphs CRLS 22,23,24,25, (HW 5?)
 - Traversal, Min Spanning Tree, Shortest Path, Capacity, Min Flow CRLS 26, (HW6?)
- IV Selected Advanced Topics & Projects
 - Spatial Data Structures, FFT's, Complexity, Approx. Solutions, Quantum Computing etc

C vs C++ Advice — C++ compiler is C

1. To understand what is under the hood and how algorithms work go to C (The C++ compiler generates C!)
3. **Bottom up:** Simpler the better to see how the computer executes an algorithm and to optimized performance.
5. (KISS) Avoid **C++ sugar**. Adopt C++ism only when that add value.
7. Of course you can always use **C++ libraries!**
9. Standards are always better **Avoid re-inventing the wheel!**

INTRODUCTORY READING IN CRLS

- CRLS I.2
- CRLS I.3
- CRLS I.4
- CRLS 1.5 Just a bit of averaging!

What is an algorithm? An unambiguous list of steps (program) to transform some input into some output.



- Pick a Problem (set)
- Find method to solve
 1. Correct for all cases (elements of set)
 2. Each step is finite ($\Delta t_{\text{step}} < \text{max time}$)
 - Next step is unambiguous
 - Terminate in finite number of steps
- ◆ You know many examples:
GCD, Multiply 2 N bit integers, ...

Abu Ja'far Muhammad ibn Musa Al-Khwarizmi
Bagdad (Iraq) 780-850

Searching Sorted List :

- int a[0], a[1],a[2],a[3],.... a[m],..... a[2],a[N-1]

Three Algorithms:

- *Linear Search* → O(N)
(after Sorting)
- *Bisection Search* → O(log(N)).
- *Dictionary Search* → O(log[log[N]])



Euclid's Algorithm GCD

(325-265 BC in Egypt)

The Greatest Common Divisor (gcd) of positive integers p and q is the largest integer which divides p and q evenly.

Can assume $p > q$

If $p = n q + r$, then →
 $\text{gcd}(p,q) = \text{gcd}(q,r)$

```
int qcd(int p, int q)
{ int r;
  while(q!=0)
    { r = p%q;
      p=q; q=r;}
  return p; }
```

$$\text{gcd}(22,8): 22 = 2*8 + 6$$

$$\text{gcd}(8,6): 8 = 1*6 + 2$$

$$\text{gcd}(6,2): 6 = 3*2 + 0$$

Answer = 2

Complexity: $q = N$: $T(N) = \text{calls to gcd}$

Worst case: $T(N) < 1.44 \log_2(N)$

Average: $T(N) \gg (12 \ln 2 / \pi^2) \ln N$

Proof of Euclid's Algorithms

- With $r = p \% q$, $p = n q + r$ if $r > 0$ q is not a common factor must be small then $q \cdot k = \gcd(p, q)$ then
- Therefore $p = k p'$ and $q = k q'$
 -) $k p' = n k q' + r$
 -) so p , r & q have k as a common factor
 -) $\gcd(q, r) = K \geq k$
- BUT K can't be bigger than K common factor in p and q
 -) $p = n q + r = n K q'' + K r'$
 -) K is common factor in p and q
 -) so $K \leq k$ and $\gcd(p, q) = \gcd(q, r)$

Q.E.D (Quod Erat Demonstrandum)

Later: $p \% q < p/2$ implies $T(N) < 2 \log_2(N)$

Halting Problem: is this an algorithm?

Examples:

$x=1 \rightarrow 1$

$x=2 \rightarrow 2,1$

$x=3 \rightarrow 3,10,5,16,8,4,2,1$

...

$x=27 \rightarrow 27,82,41,124,62,31,94,47,142,71,214,107,322,161,484,$
 $242,121,364,182,91,274,137,412,206,103,310,155,466,233,$
 $700,350,175,526,263,790,395,1186,593,1780,890,445,1336,618,$
 $309,928,464,232,116,58,29,88,44,22,11,34,17,52,26,13,40,20,10,$
 $5,16,8,4,2,1$

The 3x + 1 Problem by L. Collatz (1937)

ENDS?(x):

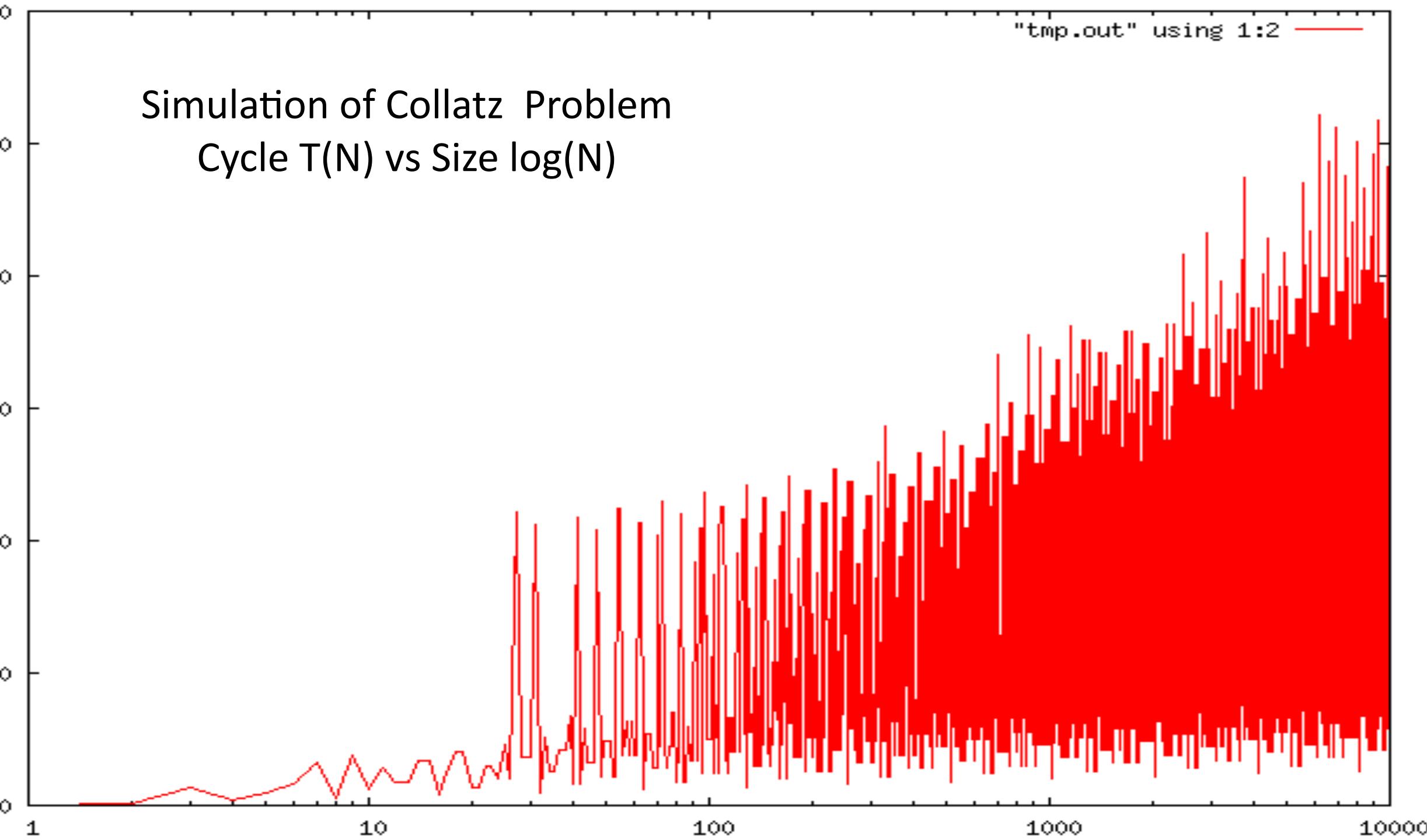
```
while  $x > 1$ :  
    print  $x$   
    if  $x$  is even  
        then  $x$  gets  $x/2$   
    else  $x$  gets  $3x+1$ 
```

```
print  $x$   
halt!
```

"tmp.out" using 1:2

Simulation of Collatz Problem

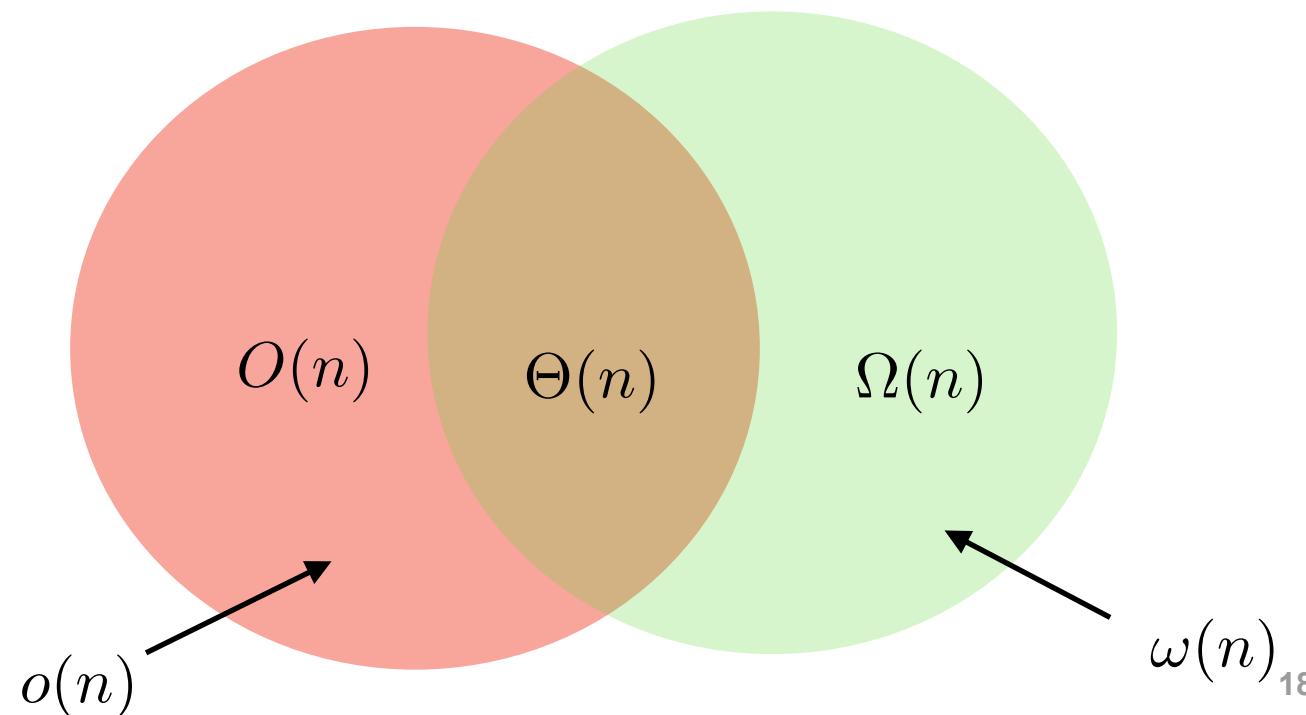
Cycle T(N) vs Size log(N)



Growth of Algorithm with Size n

$$T(n) = O(g(n)) \quad \text{or} \quad T(n) \in O(g(n))$$

- $T(n)$ in set $O(g(n))$
 - like $T(n) \leq g(n)$ for large
 - e.g. n^a $\log(n)$ $\exp[n]$ etc.



Why is big-O important?

	time (processor doing ~1,000,000 steps per second)					
input size	10	20	30	40	50	60
$\log n$	3.3μsec	4.4μsec	5μsec	5.3μsec	5.6μsec	5.9μsec
n	10μsec	20μsec	30μsec	40μsec	50μsec	60μsec
n^2	100μsec	400μsec	900μsec	1.5msec	2.5msec	3.6msec
n^5	0.1sec	3.2sec	24.3sec	1.7min	5.2min	13min
3^n	59msec	48min	6.5yrs	385,500yrs	2×10^8 centuries...	
$n!$	3sec	7.8×10^8 millennia				

**Non polynomial algorithms are terrible!
Logs are great!**

All Logarithms are the “same”

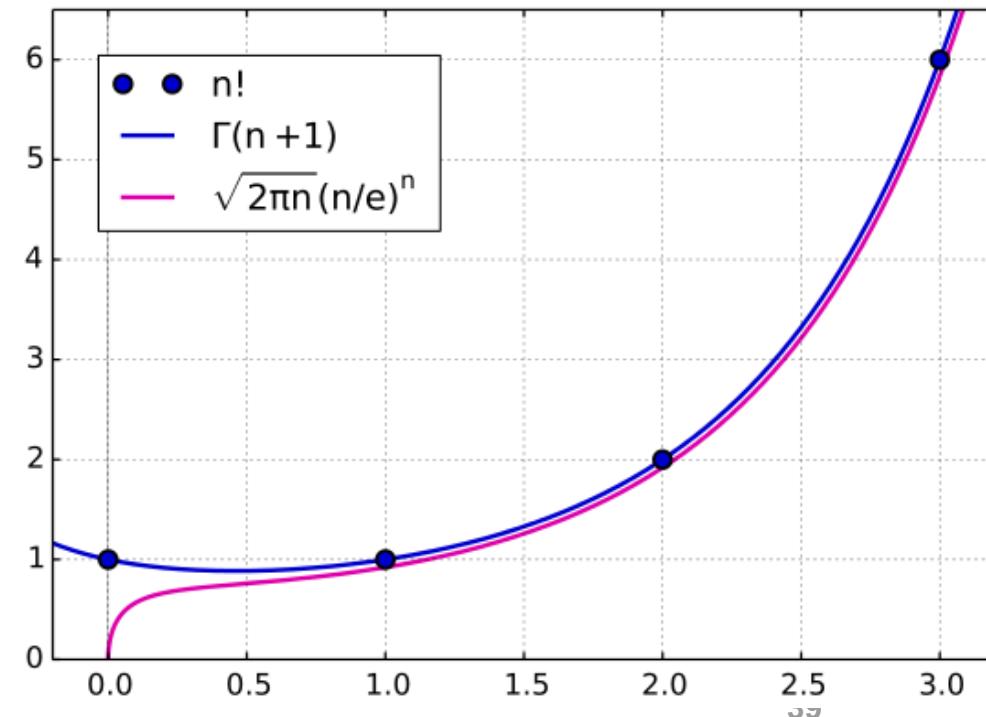
$$N = b^{\log_b(N)}$$

Therefore $\log_a(N) = \log_a(b^{\log_b(N)}) = \log_a(b) \log_b(N)$

Factorial: Worse than Exponential

Decisions $2^D > N!$ implies

$D > \log_2(N!) \simeq N \log N \dots$





Rules of thumb

- For polynomials, only the largest term matters.

$$a_0 + a_1 N + a_2 N^2 + \cdots + a_k N^k \in O(N^k)$$

- $\log N$ is in $o(N)$

Proof: As $N \rightarrow \infty$ the ratio $\log(N)/N \rightarrow 0$

- Some common functions in increasing order:

1 $\log N$ \sqrt{N} N $N \log N$ N^2 N^3 N^{100} 2^N 3^N $N!$ N^N

Insertion Sort --- Deck of Cards

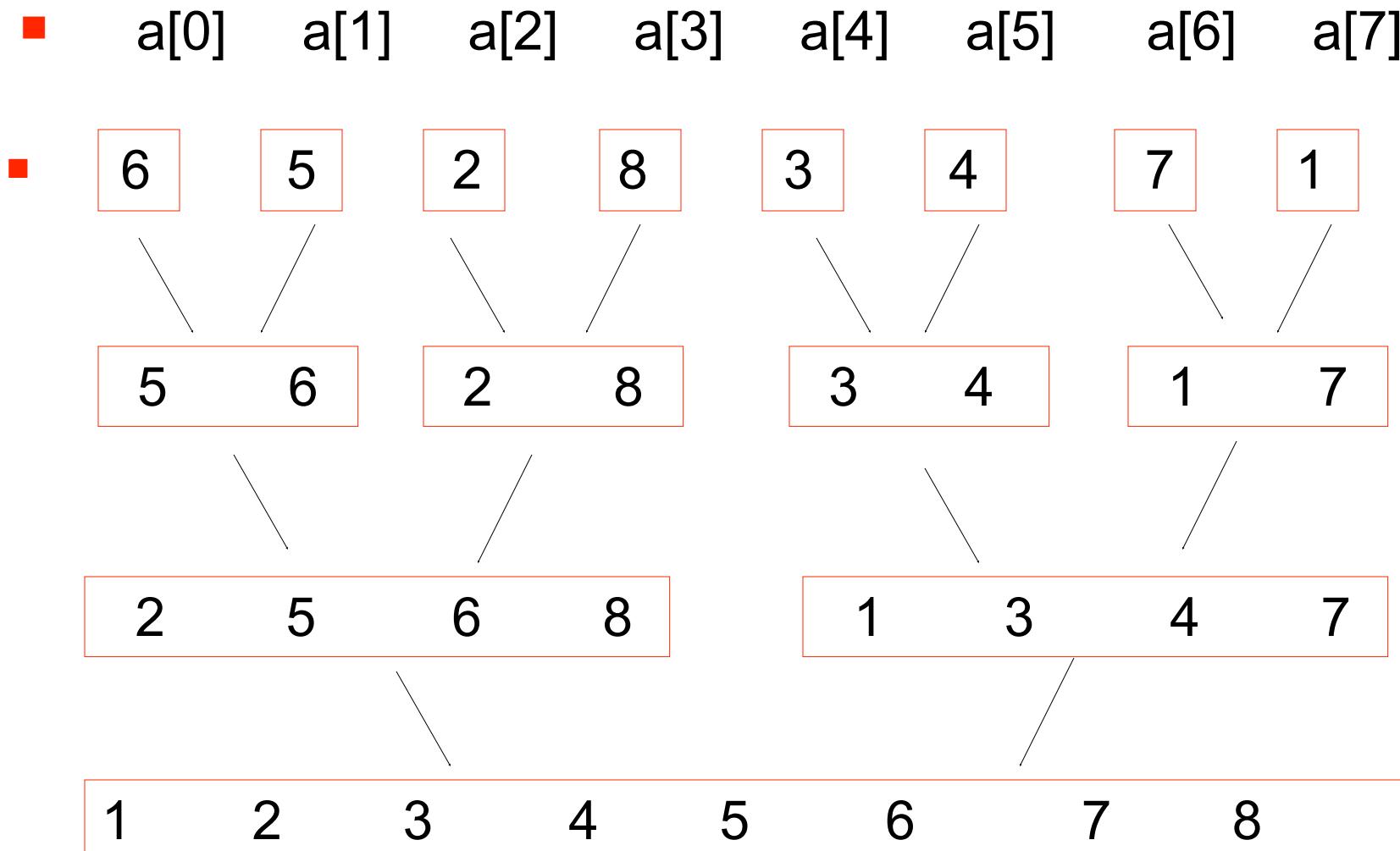
- Insertion Sort(a[0:N-1]):
 for ($i=1$; $i < n$; $i++$)
 for ($j = i$; $(j>0) \&\& (a[j]< a[j-1])$; $j--$)
 swap $a[j]$ and $a[j-1]$;

Worst case $\Theta(N^2)$ number of “swaps” (i.e. time)

Outer loop trace for Insertion Sort: $O(n^2)$

■	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	
									(Swaps)
■	6	5	2	8	3	4	7	1	(1)
	5 ←	→ 6							
■	5	6	2	8	3	4	7	1	
	(2)								
	2 ←	→ 6							
	2 ←	→ 5							
■	2	5	6	8	3	4	7	1	(0)
■	2	5	6	8	3	4	7	1	(3)
■	2	3	5	6	8	4	7	1	(3)
■	2	3	4	5	6	8	7	1	(1)
■	2	3	4	5	6	7	8	1	(7)

Merge Sort - Recursive $O(n \log(n))$



How do we find $T(n)$? What is big Oh ?

- Count the number of steps:
 - What is a step? RAM serial model.
 - Iterative loops: Sum series like

$$\sum_{i=0}^N i^k = 1 + 2^k + 3^k + \dots + N^k \sim O(N^{k+1})$$

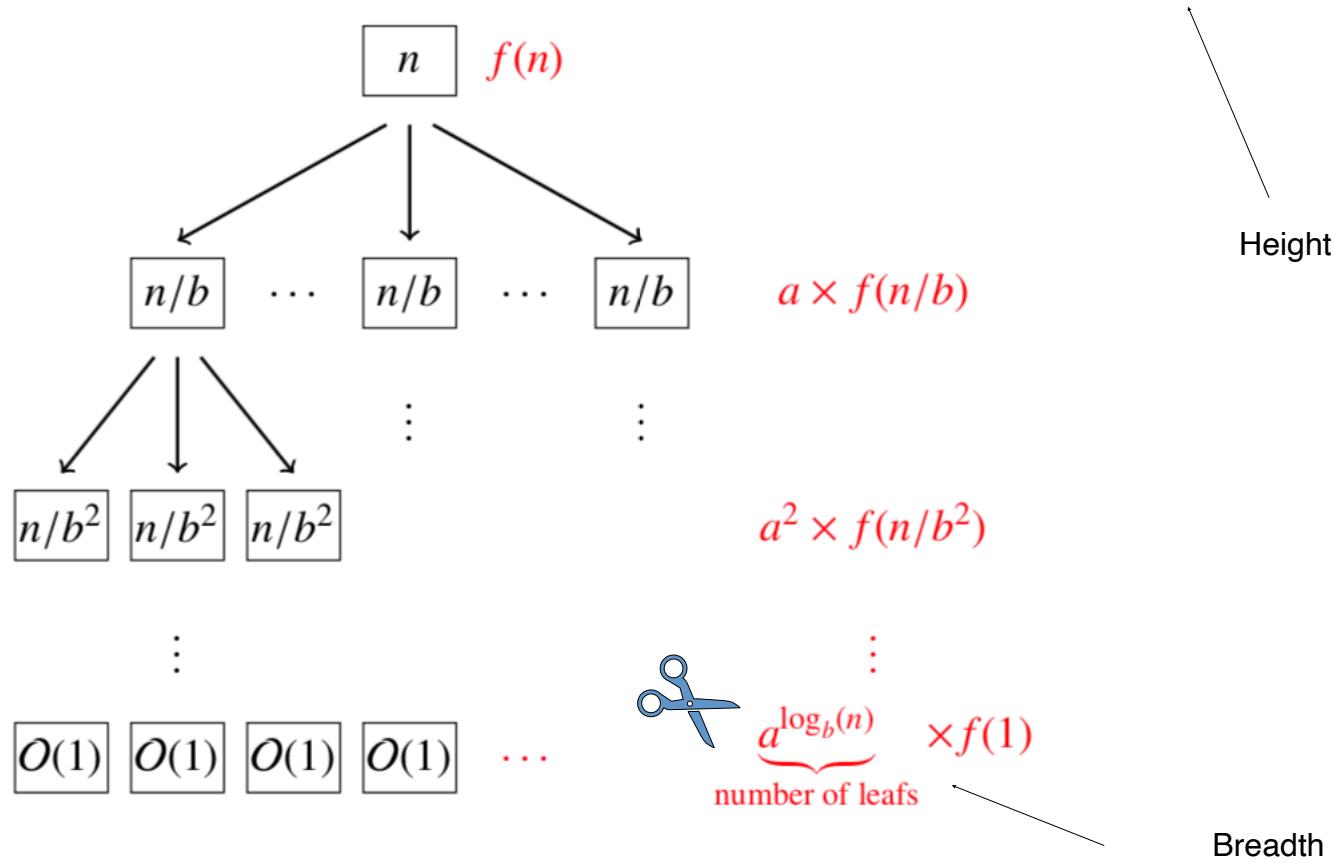
but $k = -1 \rightarrow O(\log(n))$

- Solve Recursive Relations: (Best method is parametrized guess)
 $T(n) = a T(n/b) + O(f(n))$
Homogeneous + Inhomogeneous form
- Run a program with a timer (or counter)!

Build Tree to Solve

$$T(n) = aT(n/b) + f(n)$$

$$n/b^h = 1 \implies h = \log_b(n)$$



$$T(n) = f(n) + af(n/b) + \dots + a^{\log_b(n)-1}f(b^2) + a^h T(1)$$

Master Equation:

$$T(N) = a T(N/b) + \Theta(g(N))$$

Theorem: The asymptotic Solution is:

- $T(N) \in \Theta(N^\gamma)$ if $g(N) \in O(N^{\gamma-\epsilon}) \forall \epsilon > 0$
- $T(N) \in \Theta(g(N))$ if $g(N) \in \Omega(N^{\gamma+\epsilon}) \forall \epsilon > 0$
- $T(N) \in \Theta(N^\gamma \log(N))$ if $g(N) \in \Theta(N^\gamma)$

where $a = b^\gamma$ or $\gamma = \log(a)/\log(b)$

L'Hospital's Rule

Limit for ratio is same as for ratio of derivatives!

$$\lim_{N \rightarrow \infty} \frac{f(N)}{g(N)} = \lim_{N \rightarrow \infty} \frac{\frac{df(N)}{dN}}{\frac{dg(N)}{dN}}$$

e.g.

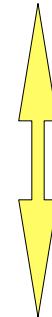
$$\lim_{N \rightarrow \infty} \frac{\log^2(N)}{N} =$$
$$\lim_{N \rightarrow \infty} \frac{2 \log(N)/N}{1} = \lim_{N \rightarrow \infty} \frac{2/N}{1} = 0$$

$$\gamma - k \rightarrow 0, \quad \text{where} \quad a = b^\gamma$$

$$T(N) = N^\gamma T(1) + c_0(N^\gamma - N^k)/(a/b^k - 1)$$

$$T(N) = N^\gamma T(1) + c_0 N^k \frac{N^{\gamma-k} - 1}{b^{\gamma-k} - 1}$$

Take derivative with respect to $x = \gamma - k$



$$T(N) = N^\gamma T(1) + c_0 N^k \log(N)/\log(b)$$

More useful stuff

- Logarithmic sum (Harmonic Series):

$$H_N = \sum_{n=1}^N \frac{1}{n} = \ln(N) + \gamma_{Euler} + \Theta(1/N)$$

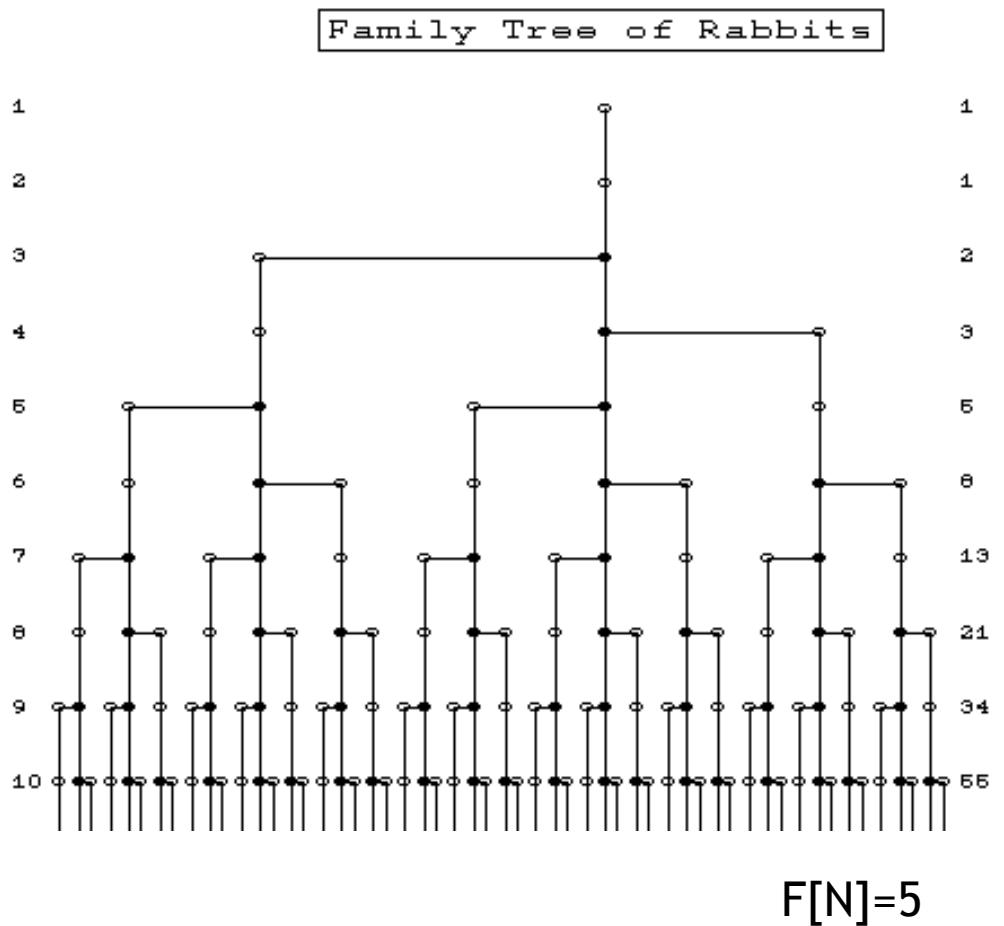
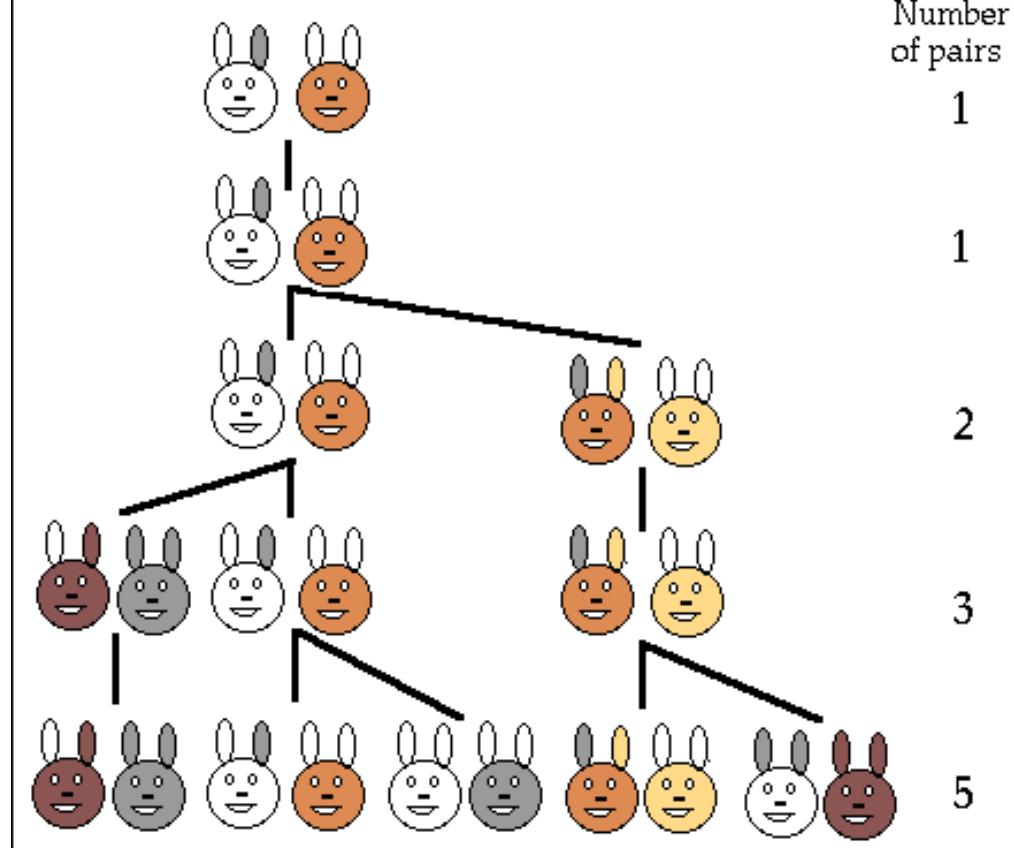
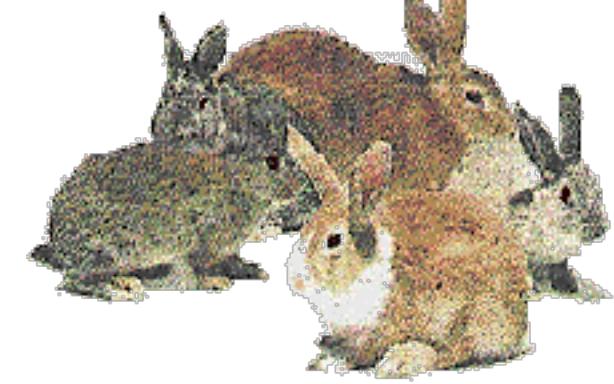
$$\gamma_{Euler} = 0.5772156649015328606512090082$$

- Stirling's Approx: $N! \simeq \sqrt{2\pi N} N^N e^{-N} (1 + O(1/N))$

$$\log(N!) = N \log(N) - N \log(e) + \frac{\log(2\pi N)}{2} + \Omega(1/N)$$

Rabbits

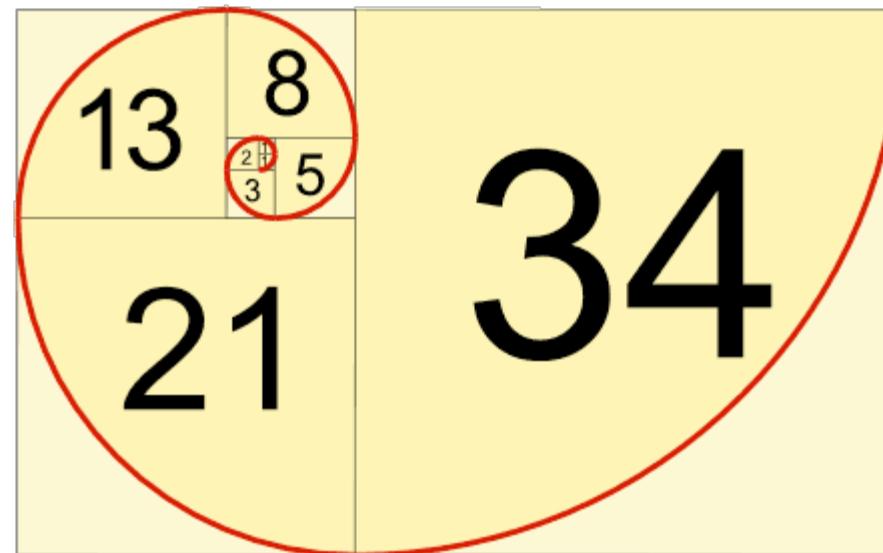
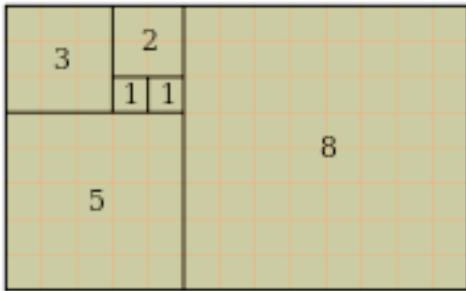
Fibonacci investigated (in the year 1202) was about how fast rabbits could breed in ideal circumstances.
Females take one month to mature: Pairs mate and produce a male and female in a month



Fibonacci: $F(N) = F(N-1) + F(N-2)$ →

0,1,1,2,3,5,8 , for $N = 0,1,2,3,....$

- Many examples in nature!



Rabits, Bees and Double Window Panes

Fibonacci: $F_k = F_{k-1} + F_{k-2} \Rightarrow 0, 1, 1, 2, 3, 5, 8, \dots$

Characteristic equation, try:

$$F_k = \phi^k \implies \phi^k = \phi^{k-1} + \phi^{k-2}$$

$$\phi^2 - \phi - 1 = 0 \quad \phi = \frac{1}{2} \pm \frac{1}{2}\sqrt{5}$$

$$F_k = c_1 \left(\frac{1 + \sqrt{5}}{2} \right)^k + c_2 \left(\frac{1 - \sqrt{5}}{2} \right)^k$$

$$F_k = \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2} \right)^k - \left(\frac{1 - \sqrt{5}}{2} \right)^k \right]$$

Note that recursive Time
is homogeneous relation:

$$T(N) = T(N-1) + T(N-2)$$

Guess power form $T(N) \sim x^N$

$$ax^2 + bx + c = 0 \implies x = -b/2a \pm \sqrt{(b/2a)^2 - c/a}$$

Sums

- Cases:

$$\sum_{i=1}^N 1 = N \approx \frac{1}{1} N$$

$$\sum_{i=1}^N i = \frac{1}{2} N(N+1) \approx \frac{1}{2} N^2$$

$$\sum_{i=1}^N i^2 = \frac{1}{6} N(N+1)(2N+1) \approx \frac{1}{3} N^3$$

$$\sum_{i=1}^N i^k \approx \frac{1}{k+1} N^{k+1}$$

Prove this by Integration:

Let's be very careful for $f(n) = cn^k$

$$T(n) = aT(n/b) + c n^k$$

$$aT(n/b) = a^2T(n/b^2) + c an^k/b^k$$

$$a^2T(n/b^2) = a^3T(n/b^3) + c a^2n^k/b^{2k}$$

... ...

$$a^{h-2}T(b^2) = a^{h-1}T(b) + c a^{h-2}n^k/b^{(h-2)k}$$

$$a^{h-1}T(b) = a^hT(1) + c a^{h-1}n^k/b^{(h-1)k}$$

Therefore

$$T(n) = a^hT(1) + c n^k \frac{(a/b^k)^h - 1}{a/b^k - 1}$$

$$a^h = n^\gamma \quad \longrightarrow \quad = n^\gamma T(1) + c \frac{n^\gamma - n^k}{a/b^k - 1}$$

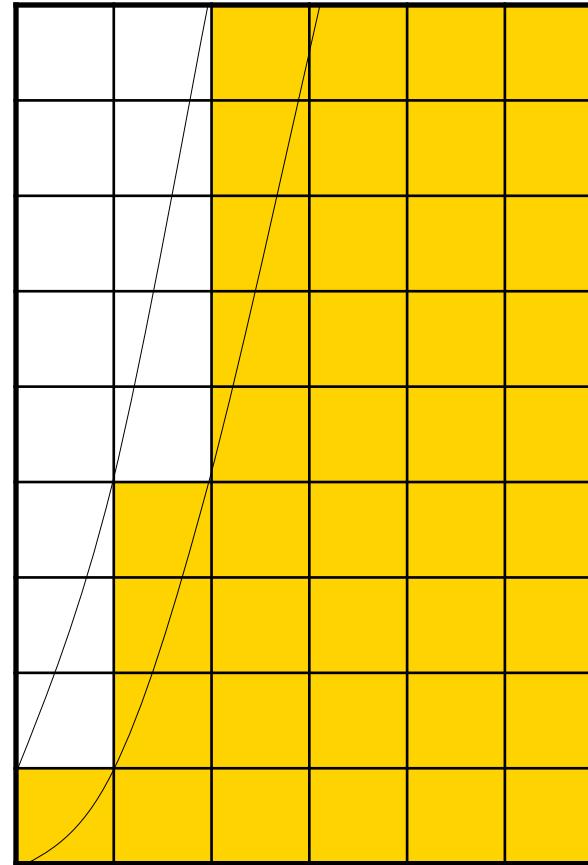
since $1 + a/b^k + (a/b^k)^2 + (a/b^k)^3 + \dots + (a/b^k)^{h-1} = \frac{(a/b^k)^h - 1}{a/b^k - 1}$

Estimating Sums

- Integral Bounds:

$$S_k = \sum_{i=1}^N i^k$$

Estimate by integrating $S_k(x) = x^k$



$$\int_0^N x^k dx \leq S_k = \sum_{i=1}^N i^k \leq \int_0^N (x+1)^k dx$$

$$\frac{1}{k+1} N^{k+1} \leq S_k \leq \frac{1}{k+1} ((N+1)^{k+1} - 1)$$

Maximum Subsequence Sum: CLRS 4.1

- Given $a[0], a[1], \dots, a[N-1]$ find max

$$\sum_{k=i}^j a[k]$$

- Dumbest

$$O(N^3)$$

- Dumb

$$O(N^2)$$

- Smart

$$O(N \log(N))$$

- Smartest

$$O(N)$$

```

for ( i = 0; i < N; i++)
    for(j = i; j < N; j++)
    { Sum = 0;
        for(k=i; k<j+1; k++)
            Sum += a[k];
        if(Sum > MaxSum)
            MaxSum = Sum;
    }

```

$O(N^3)$

$$\sum_{k=i}^j 1 = j - i + 1$$

$$\sum_{j=i}^{N-1} (j - i + 1) = \frac{1}{2}(N - i)(N - i + 1)$$

$$\frac{1}{2} \sum_{i=0}^{N-1} i(i + 1) = \frac{1}{6}(N^3 + 3N + 2N)$$

```

for ( i = 0; i < N; i++)
{ Sum = 0;
    for(j=i; j<N; j++)
        Sum += a[j];
    if(Sum > MaxSum)
        MaxSum = Sum;
}

```

$O(N^2)$

$$\begin{aligned}
\sum_{j=i}^{N-1} 1 &= N - i + 1 = N - i \\
\sum_{i=0}^{N-1} (N - i) &= N^2 - \left(\frac{N(N - 1)}{2}\right) \\
&= \frac{1}{2}(N^2 + N)
\end{aligned}$$

Recursion versus Single Pass

- $T(N) = 2 T(N/2) + c N$
 - Large left/right + sum to left and right for split screen.

$O(N \log(N))$

$O(N)$

- On line:
 - Quit when you are in debt and start over.

```
Sum = 0;  
for(j=0; j<N; j++){  
    Sum += a[j];  
    if(Sum > MaxSum)  
        MaxSum = Sum  
    else if (Sum < 0)  
        Sum = 0;  
    }  
}
```

Why? Sort of obvious if you draw the earning graph.