EC504 Algorithms and Data Structure
Fall 2020 Tuesday & Thursday
11:00AM - 1:00PM

Rich Brower and Krishna Palle and Casey Berger

Course Organization

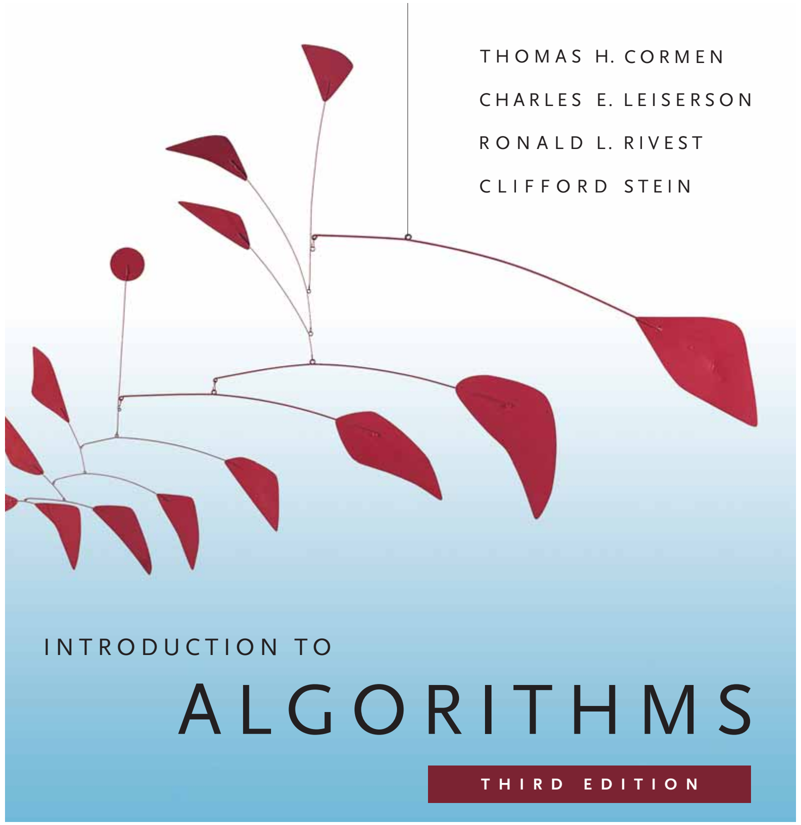
- Text:
  - Cormen, Leiserson, Rivest & Stein (CLRS), Fundamental text!

    “Introduction to Algorithms” 3rd Edition MIT Pres
- Keynote Slides guide to CLRS
- Reference:

- Wikepedia!
- Mark Allen Weiss “Data Structure and Algorithms in C”.
- UNIX, Makefiles, very basic C/C++ and gnuplot:

- Grading:

HW with Coning Programs:	30%
Zooming Class Partipation	10%++
Project	25%
Midterm (up to trees) Oct	15%
Final (comprehensive) Dec	20%



EC504 Course Organization

- Why Algorithm ==> Data Strutures
- CRSL text with Slide Summaries
- Scaling, Math and Empirical Analysis on Simple Cases.
- Use GitHub (EC405), Slack and CCS and Unix Tools
- HW's pencil and paper: pdf turned in HW# on CCS
- Software delivered CCS — Must run from Makefile.
- Basic Unix environment — useful for computer engineers to know!

Course Outline

- Algorithms Analysis CRSL 2-4 (5) HW1
  - Definition of Problem Class of Size N
  - Math for large N Asymptotics:
- I. 1-D Data Structures CRSL 6,7,8,9 HW2
  - Arrays, Lists, Stacks, Queues CRSL 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

INTRODUCTION

- 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 ($\Delta t_{step} < \text{max time}$)
  - Next step is unambiguous
  - Terminate in finite number of steps
- You know many examples:

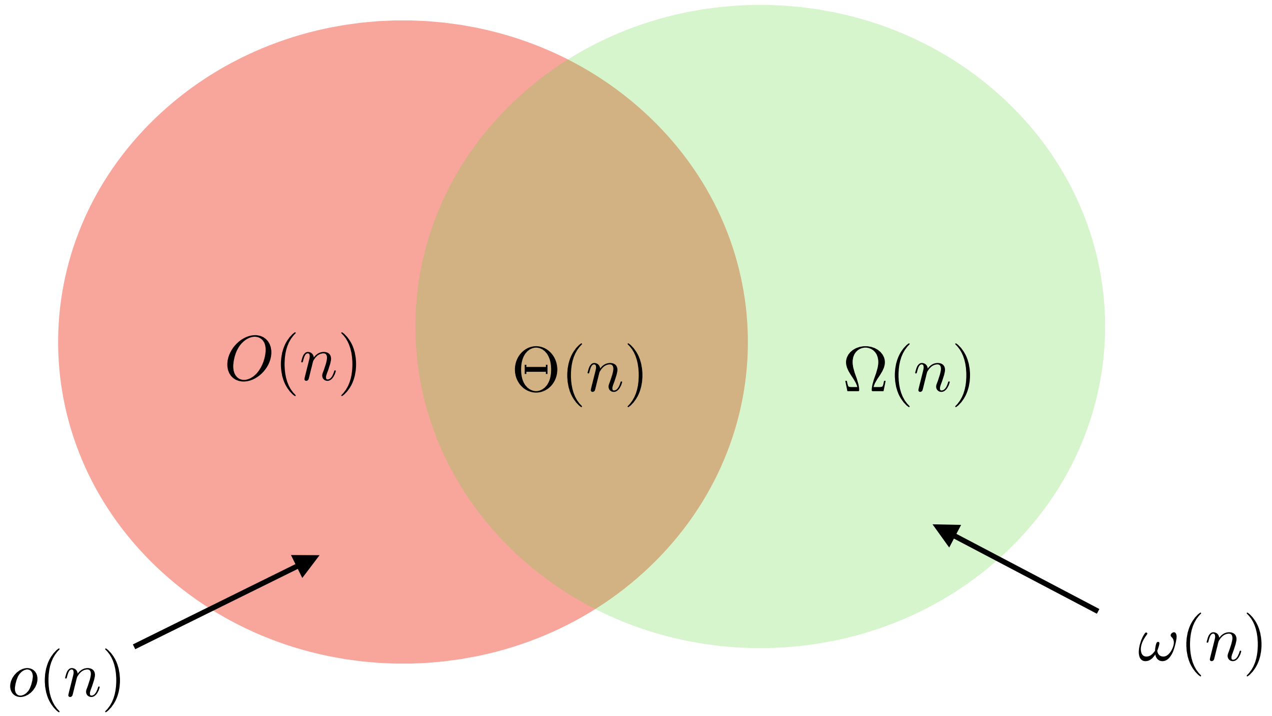
  GCD, Multiply 2 N bit integers, …

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

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) <= g(n) for large
  - e.g n^a log(n) exp[n] etc.



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 ➔ 1 the ratio log(N)/N ➔ 0

- Some common functions in increasing order:

1 log N √N N NlogN N^2 N^3 N^{100} 2^N 3^N N! N^N

Why is big-O important?

(processor doing ~1,000,000 steps per second)

input size \ time						
N	**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	$2x10^8$ centuries...	
n!	3sec	$7.8x10^8$ millennia ……				

Non polynomial algorithms are terrible!
Logs are great!

Logarithms

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

Therefore

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

Insertion Sort --- *Deck of Cards*

- Insertion Sort(a[0:N-1]):

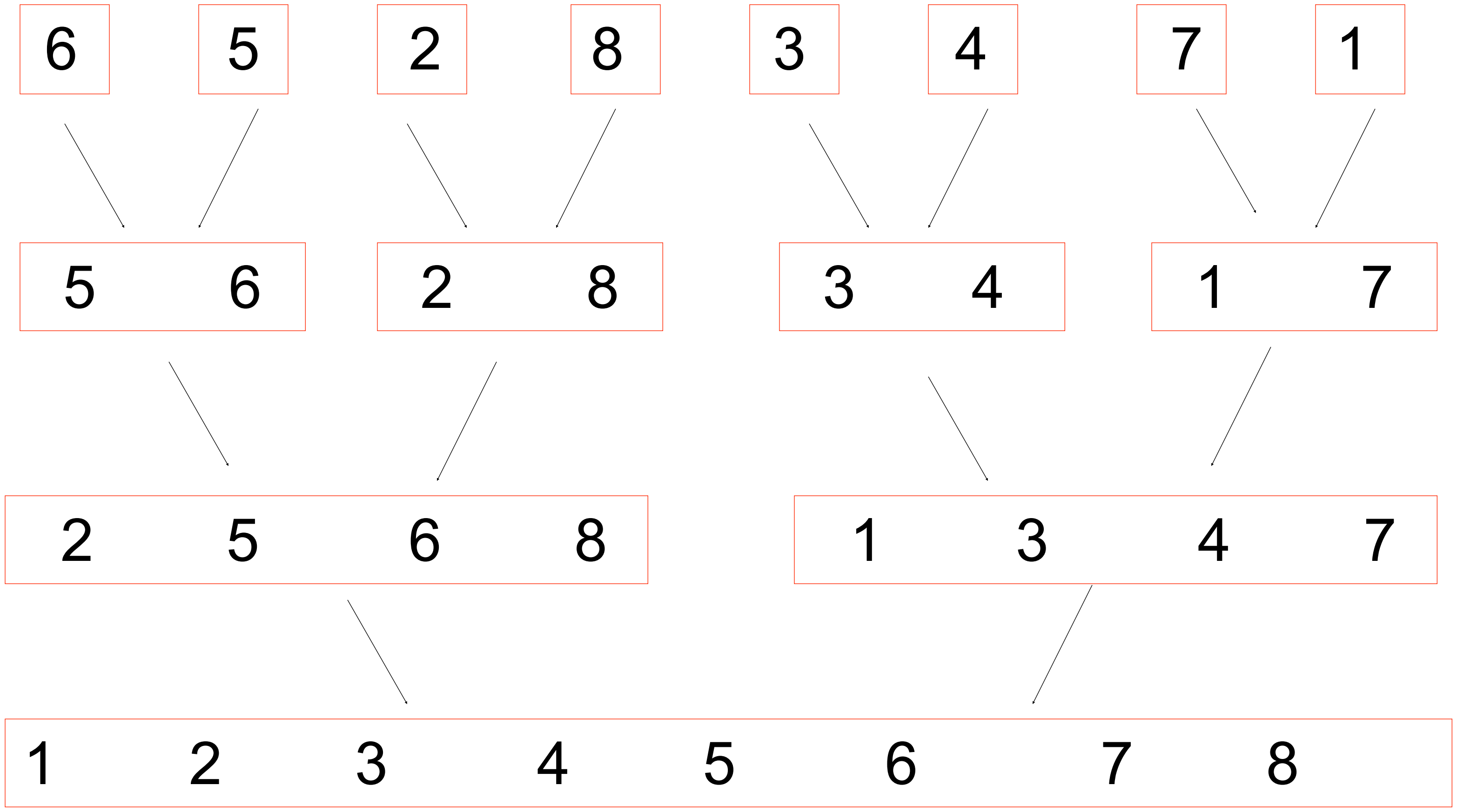
```
for (i=1;  i < n;  i ++)
      for (j = i;  (j>0) && (a[j]<a[j-1]);  j--)
            swap a[j] and a[j-1] ;
```

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)

- a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7]
- 

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

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

but k= -1 ➔ O(log(n))

  - Solve Recursive Relations:

T(n) = a T(n/b) + O(f(n))

Sums

- Cases:

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

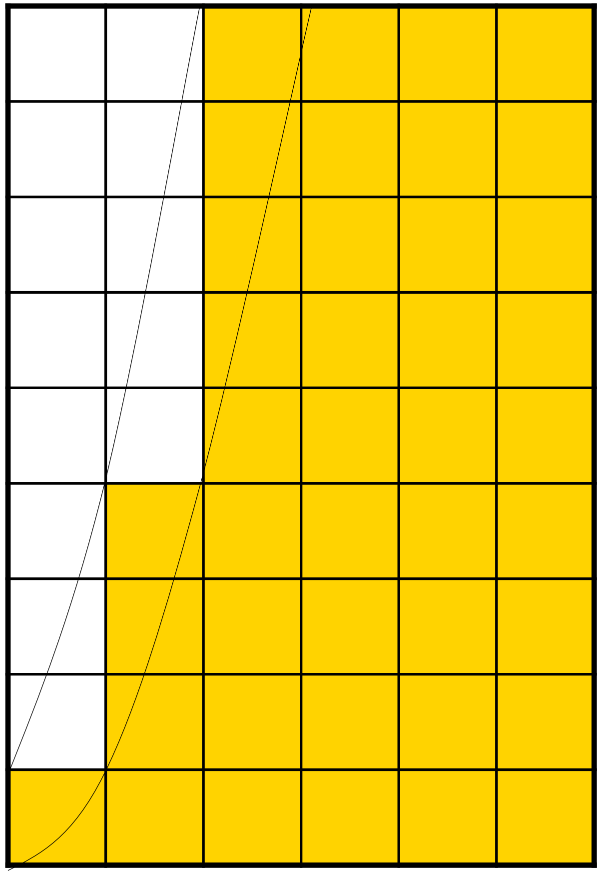
$$\sum_{i=1}^{N} i = \frac{1}{2} N(N+1) \approx \frac{1}{2} N^2$$

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

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

Prove this by Integration:

Estimating Sums



- Integral Bounds:

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

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

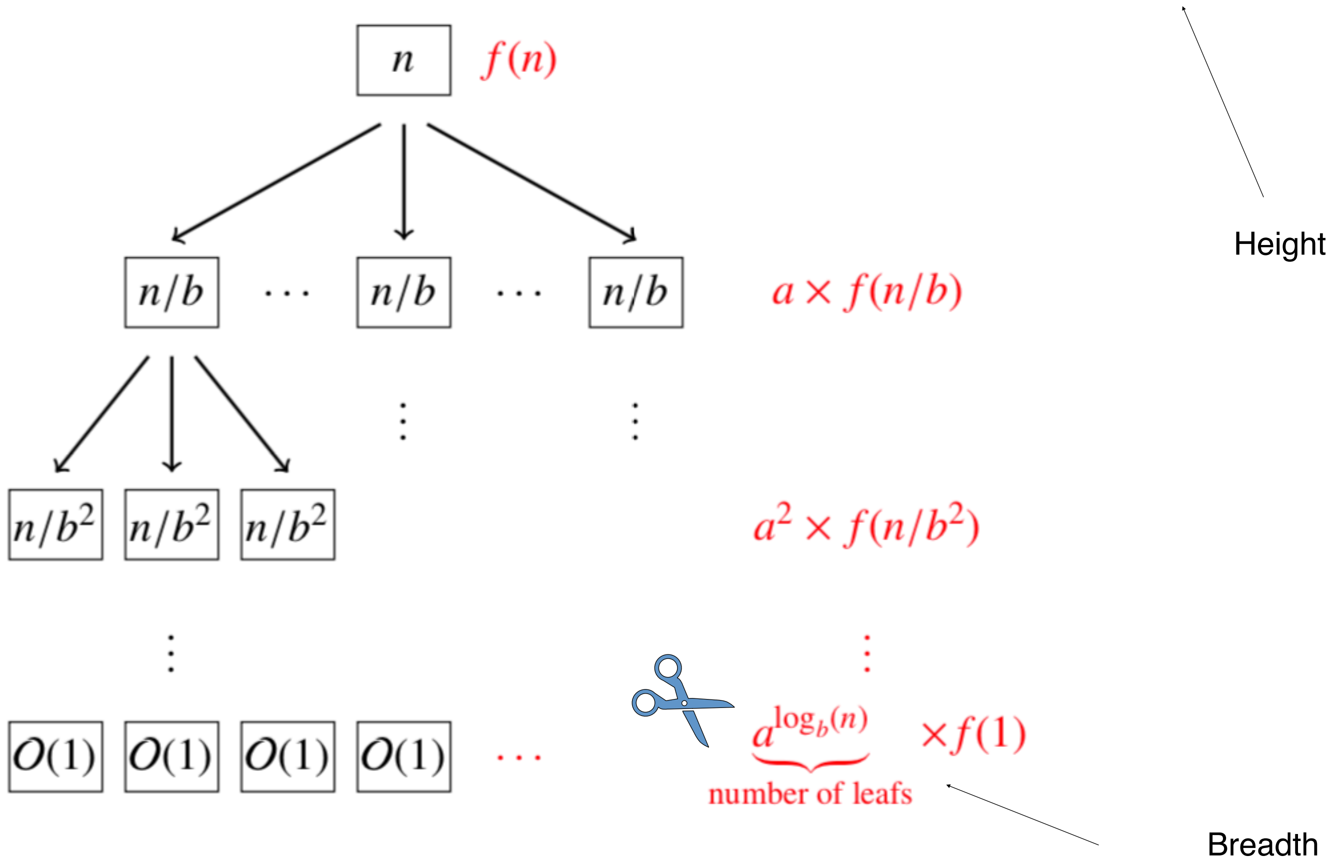
$$\underline{\int_0^N x^k dx} \leq S_k = \sum_{i=1}^{N} i^k \leq \underline{\int_0^N (x+1)^k dx}$$

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

Build Tree to Solve

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

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



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

Master Equation (brute force): T(n) = aT(n/b) + f(n)

$$\begin{aligned} T(n) &= aT(n/b) + f(n) \\ aT(n/b) &= a^2T(n/b^2) + af(n/b) \\ a^2T(n/b^2) &= a^3T(n/b^3) + a^2f(n/b^2) \\ \cdots \quad & \quad \cdots \\ a^{h-2}T(b^2) &= a^{h-1}T(b) + a^{h-2}f(b^2) \\ a^{h-1}T(b) &= a^hT(1) + a^{h-1}f(b) \end{aligned}$$

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

$a^h = n^\gamma$ **using:** $n/b^h = 1 \implies h = \log_b(n)$

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

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

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

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

$$\cdots \qquad \cdots$$

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

$n/b^h = 1$

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

Therefore

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

$a^h = n^\gamma$ $\longrightarrow$

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

since $\quad 1 + a/b^k + (a/b^k)^2 + (a/b^k)^3 + \cdots + (a/b^k)^{h-1} = \dfrac{(a/b^k)^h - 1}{a/b^k - 1}$

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!

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

e.g.

$$\lim_{N\to\infty} \frac{\log^2(N)}{N} =$$

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

$$\gamma - k \to 0, \quad where \quad a = b^\gamma$$

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

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

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):

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

$$\gamma_{Euler} = 0.5772156649015328606065120900082$$

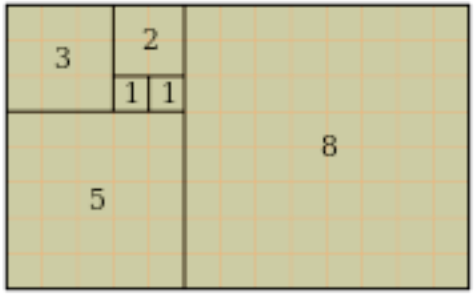
- Stirling's Approx: $N! \simeq \sqrt{2\pi N} N^N e^{-N} (1 + O(1/N))$

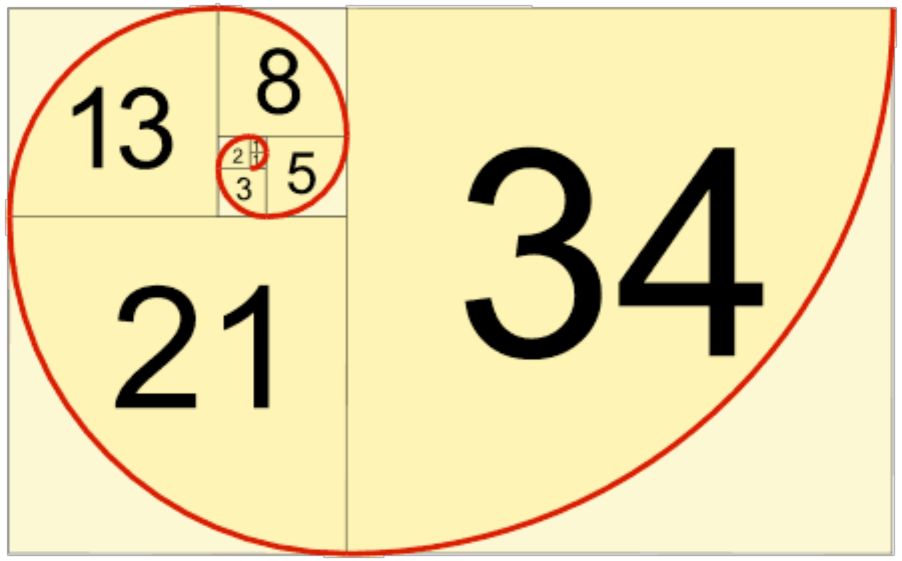
$$\log(N!) = N \log(N) - N \log(e) + \frac{\log(2\pi N)}{2} + \Omega(1/N)$$

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!

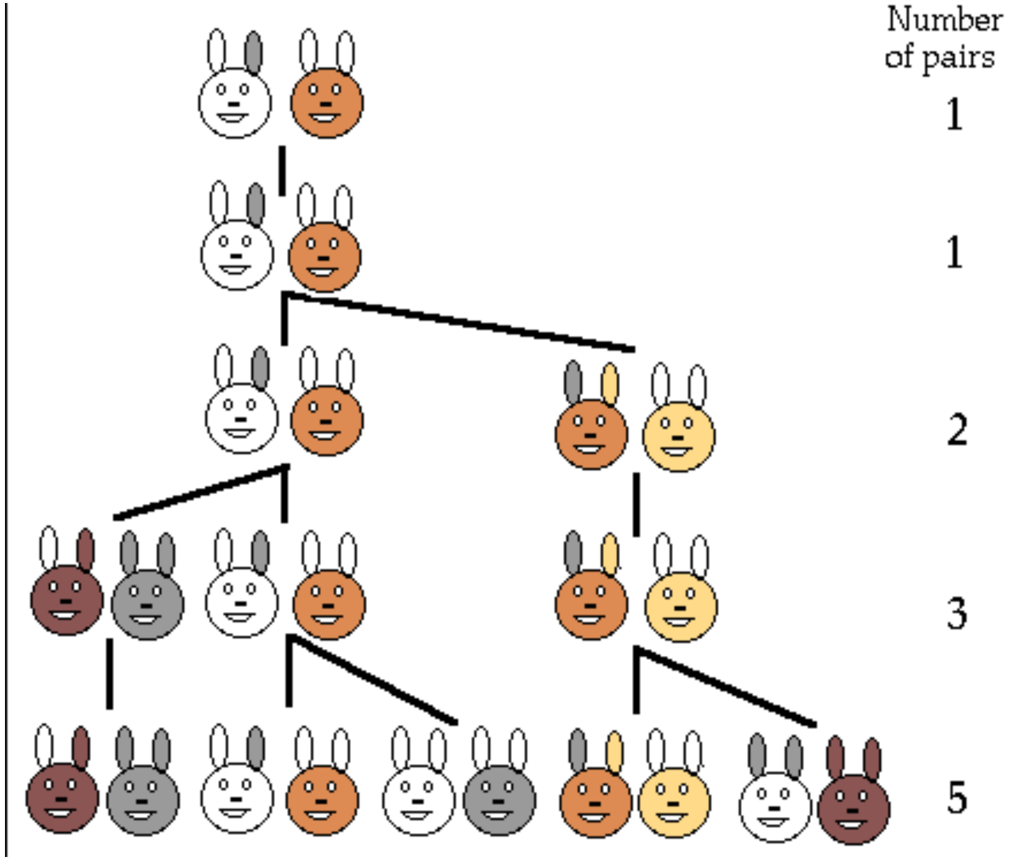


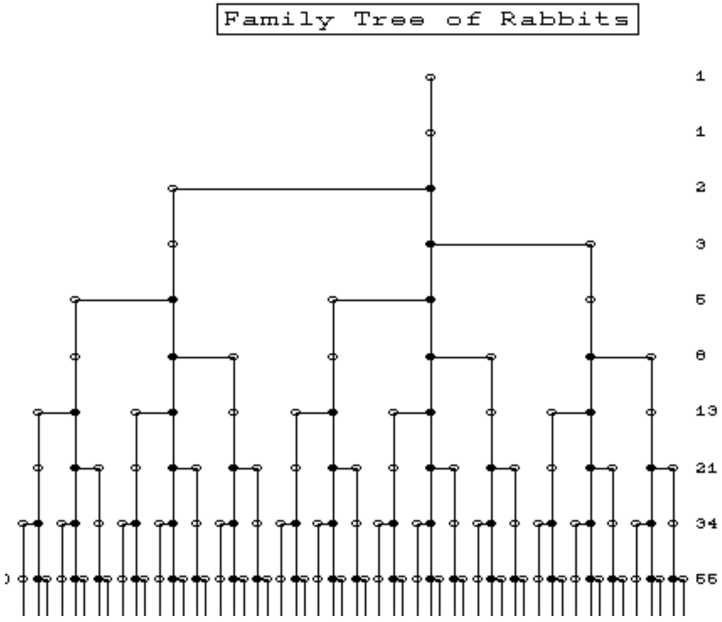


Rabits, Bees and Double Window Panes

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: $F_k = F_{k-1} + F_{f-2}$ ➔ *0,1,1,2,3,5,8 , ...*

Characteristic equation, try:

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

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

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

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

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

Maximum Subsequence Sum: CLRS 4.1

- Given a[0], a[1] ,..., a[N-1] find max $\boxed{\sum_{k=i}^{j} a[k]}$

  – Dumbest $\boxed{O(N^3)}$

  – Dumb $\boxed{O(N^2)}$

  – Smart $\boxed{O(N\log(N))}$

  – Smartest $\boxed{O(N)}$

i,j,k loops

```
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;
}
```

$$O(N^3)$$

$$\begin{aligned}
\Sigma_{k=i}^{j} 1 &= j - i + 1 \\
\Sigma_{j=i}^{N-1} (j - i + 1) &= \tfrac{1}{2}(N - i)(N - i + 1) \\
\tfrac{1}{2}\Sigma_{i=0}^{N-1} i(i + 1) &= \tfrac{1}{6}(N^3 + 3N + 2N)
\end{aligned}$$

i,j loops

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

$$O(N^2)$$

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

Recursion versus Single Pass

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

$\boxed{O(N\ \log(N))}$

Joining is 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;
                        }
```

$\boxed{O(N)}$

When you loose try and try again!

Searching and Sorting

- Searching
  - Linear O(N)
  - bisection O(log(N))
  - dictionary O(log(log(N
- Sorting
  - Insertion, bubble, selection $O(N^2)$ CRLS: 2.1
  - Merge, Quick, Heap O(N log (N)) CRLS: 2.2,
    Bin (Count), Radix, Bucket O(N) CLRS: 8
- Proof: $\Omega(N^2)$ near neighbor exchange
- Proof: $\Omega(N \log(N))$ Comparison search
- Median (or k quick selection) Problem CLRS: 9

Searching: *"Why Sort at All?"*

- 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]])

Bisection Search of Sorted List

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

i j

```
i= 0; j= N-1; m = N/2
 while(b!=a[m] && i!=j ){
          if(b>a[m])    i = m+1;
          if(b<a[m])    j = m-1;
          m =   (j-i)/2 + i;}
if(b==a[m])) "found it" else "not found"
```

Choose mid point

$T(N) = T(N/2) + c_0$ ➔ $T(N)$ » $Log(N)$

Dictionary: Sorted and Uniform

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

i j

m

Dictionary: Same code EXCEPT

estimate location of b

x = fractional distance (0<x<1)

x = (b-a[i])/(a[j] – a[i]) ;

m = x (j-i) + i ;

T(N) = T($N^{1/2}$) + c_0 ➔ T(N) » Log(Log(N))

$$N \to N^{\frac{1}{2}} \to N^{\frac{1}{4}} \to N^{\frac{1}{8}} \cdots \to N^{\frac{1}{2^n}} = 1 \quad \text{or} \quad n = log_2(log_2(N))$$

- *Extra Knowledge Helps: % Error » 1/$N^{1/2}$*

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

- 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)

Bubble Sort --- Sweep R to L

- *<u>Bubble Sort(a[0:N-1])</u>:*

  for i=0 to n-1

      for j = n-1 to i + 1

          if a[j]<a[j-1] then

              swap a[i] and a[j]

Worst case $\Theta(N^2)$ swaps (time)

Outer loop trace for Bubble Sort

a[0] (Swaps)	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	
6	5	2	8	3	4	7	1	(7)
1 |	6	5	2	8	3	4	7	(3)
1	2 |	6	5	3	8	4	7	(3)
1	2	3 |	6	5	4	8	7	(3)
1	2	3	4 |	6	5	7	8	(1)
1	2	3	4	5 |	6	7	8	(0)
1	2	3	4	5	6 |	7	8	(0)
1	2	3	4	5	6	7	8	(17)

- NOTE SAME # OF SWAPS? WHY?

Average Number of N(N-1)/4 swaps

- **Best Case:** sorted order → 0 swaps

- **Worst Case:** reverse order→N(N-1)/2 swaps
  since 1 + 2 + ...+N-1 = N(N-1)/2

- **Average Case:** Pair up each of the N! permutations with its reverse order →Every pair must swap in one or the other: Thus average is half of all swaps → (1/2) N(N-1)/2 q.e.d.

Selection Sort --- (Bubble only the index)

- <u>*Selection Sort(a[0:N-1])*</u>:

```
for i=1 to n-2
  { min = i
    for j = n-1 to i + 1
          if a[j]<a[min] then
                min = j;
      swap a[i] and a[min];
  }
```

worst case $\Theta(N)$ *swaps* + $\Theta(N^2)$ *comparisons*

Outer loop trace for Selection Sort

- 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)
  1 ← → 6
- 1 5 2 8 3 4 7 6
  (1)
- 2 ←→ 5
- 1 2 | 5 8 3 4 7 6
  (1)
- 1 2 3 | 8 5 4 7 6
  (1)
- 1 2 3 4 | 5 8 7 6
  (0)
- 1 2 3 4 5 | 6 7 8

Merge Sort: *Worst Case* $\Theta(N\log(N))$

```
void mergesort(int a[ ], int l, int r)
   if (r > l)  {
      m = (r+l)/2;
      mergesort(a, l, m);
      mergesort(a, m+1, r);
      for (i = l; i < m+1; I++)  b[i] = a[i];
      for (j = m; j < r; j++)     b[r+m-j] = a[j+1];    // reverse
      for (k = l; k <= r; k++)
               a[k] = (b[i] < b[j]) ? b[i++] : b[j--]; }
```

Outer loop trace for Merge Sort

- a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7]
- 6 5 2 8 3 4 7 1

5 6 | 2 8 | 3 4 | 1 7

2 5 6 8 | 1 3 4 7

1 2 3 4 5 6 7 8