Facing recursion's big-O: *recurrence relationships*

Spam @ LACMA!

**This week:** the art and science of designing fast algorithms

Next CS 60 assignment...

**Sorting out big-O!**

Due *Tuesday, 4/15/14* ...

**Do less!**

*with big-O!*

Caught up? grading or other concerns? Let me know...
There may be a surprise for you in the fridge... or not. You'll never know if you don't look.
Thank you for being there! You are the best!
Please enjoy this can of SPAM. Open and close carefully.
The contents contain gluten, eggs, milk, soy, and lots of love!
we want the **fewest** coins that will produce this total
out of these denominations (we can use as many times as we want)

```
change(18, [1,5,9,10,12] )
```

Try filling out this table by hand – **this is the DP algorithm!**

### Fill in this table from 0 up to the actual amount of interest…
Each top cell has Min[amt], the minimum # of coins that can create amt

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>10</td>
<td>1</td>
<td>12</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

### In Java: dynamic programming!

### Watch out for the INITIAL value for each entry in these tables!
Step 1

check each src to each dst THROUGH 0

1 entry will change – which?

Step 1

Check each src to each dst through 0.

1 entry will change – which?

Intermediate nodes:

0

1

2

3

Source (src) "from"  

0 14 inf 100

inf 0 14 50

inf inf 0 14

10 inf inf 0

Destination (dst) "to"
Step 1  
check each src to each dst THROUGH 0
Step 2 check each src to each dst THROUGH

intermediate nodes:
Step 3

check each src to each dst THROUGH

2 entries will change – which?

intermediate nodes:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>inf</td>
<td>inf</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>inf</td>
<td>0</td>
<td>14</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>24</td>
<td>38</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dst</th>
<th>&quot;to&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>inf</td>
</tr>
</tbody>
</table>
Done! We now have all of the shortest-distances between all \((\text{src}, \text{dst})\) pairs!

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>14</td>
<td>28</td>
<td>42</td>
</tr>
<tr>
<td>1</td>
<td>38</td>
<td>0</td>
<td>14</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>38</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>24</td>
<td>38</td>
<td>0</td>
</tr>
</tbody>
</table>

 intermediate nodes: 0 1 2 3

What's the big-O runtime? And what about the paths?
We now have all of the shortest-distances between all (src, dst) pairs!

What's the big-O runtime?
And what about the paths?

How could we keep track of the paths as we go...?!
FW is popular in practice...

Thanks, Kevin!
Publishing success?

Looking to publish that first article? Consider Knuth's example...
THE POTREZBIE SYSTEM OF WEIGHTS AND MEASURES

This new system of measuring, which is destined to become the measuring system of the future, has decided improvements over the other systems now in use. It is based upon measurements taken by A. A. 12 at the Physics Lab. of Milwaukee Lutheran High School in Milwaukee, Wis., where the thickness of a MAD Magazine is determined to be 2.2923481

7438 1/2x14x16x3 mm. This length is the basis for the entire system, and is called one potrebite of length. The potrebite has also been standardized at 3515-3560 wave lengths of the red line in the spectrum of helium. A partial table of the Potrebite System, the measuring system of the future, is given below:

<table>
<thead>
<tr>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 potrebite = thickness of MAD Magazine</td>
</tr>
<tr>
<td>0.22923481</td>
</tr>
<tr>
<td>1 dp = 1 potrebite (p)</td>
</tr>
<tr>
<td>1 dp = 1 dekapotrebite (dp)</td>
</tr>
<tr>
<td>1 dp = 1 hectaropotrebite (hp)</td>
</tr>
<tr>
<td>1 dp = 1 kiliopotrebite (kp)</td>
</tr>
<tr>
<td>1 dp = 1 megalopotrebite (mp)</td>
</tr>
<tr>
<td>1 dp = 1 megapotrebite (mp)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VOLUME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 cubic dekapotrebite = 1 ngage (ng)</td>
</tr>
<tr>
<td>0.000001</td>
</tr>
<tr>
<td>1 dp = 1 ngage (ng)</td>
</tr>
<tr>
<td>1 dp = 1 dekangage (dg)</td>
</tr>
<tr>
<td>1 dp = 1 hectarangage (hg)</td>
</tr>
<tr>
<td>1 dp = 1 kiliangage (kg)</td>
</tr>
<tr>
<td>1 dp = 1 megalangage (mg)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ngage of helium = 1 bites (bt)</td>
</tr>
<tr>
<td>0.000001</td>
</tr>
<tr>
<td>1 dp = 1 bites (bt)</td>
</tr>
<tr>
<td>1 dp = 1 dekabites (db)</td>
</tr>
<tr>
<td>1 dp = 1 hectarabites (hb)</td>
</tr>
<tr>
<td>1 dp = 1 kiliabites (kb)</td>
</tr>
<tr>
<td>1 dp = 1 megalabites (mb)</td>
</tr>
</tbody>
</table>

*Helium is a form of gas, and it has a specific gravity of 2.2923481 and a specific heat of 3.1416.*

PICTURES BY WALLACE WOOD

TIME

Average rotation of the earth = 1 Clarke-sec (c)
10,000 sec = 1 hour (hr)
100 sec = 1 minute (min)
10 sec = 1 second (sec)
1 sec = 1 microsecond (usec)
1 usec = 1 nanosecond (ns)

DATE

October 3, 5992 (the 10th MAD was first published December 27th, 5712.)
The calendar for each monthly issue is given as 11. 51, 11. 52, 11. 53, 11. 54.

ENERGY AND WORK

1 banana = 1 watt-hr (wh)
100 bananas = 1 watt-hr (wh)
1000 bananas = 1 watt-hr (wh)

POWER

1 banana per hour = 1 kilowatt (kw)
100 bananas per hour = 1 kilowatt (kw)
1000 bananas per hour = 1 kilowatt (kw)

RADIOACTIVITY

The quantity of radium which is in equilibrium with any number of bunches of radium is 1. how many bananas are there in a bunch?

HEAT ENERGY

The quantity of heat necessary to raise one degree Kelvin is 1. how many bananas are there in a bunch?

BASIC CONVERSION FACTORS (ABBREVIATED TABLE)

1 banana = 1 watt-hr (wh)
100 bananas = 1 watt-hr (wh)
1000 bananas = 1 watt-hr (wh)
10000 bananas = 1 watt-hr (wh)
100000 bananas = 1 watt-hr (wh)

JAN KUNTZ 11 March 2011
THE POTRZEBIE SYSTEM

This new system of measuring, which is destined to become the measuring system of the future, has decided improvements over the other systems now in use. It is based upon measurements taken 6-9-12 at the Physics Lab. of Milwaukee Lutheran High School, in Milwaukee, Wis., when the thickness of MAD Magazine #26 was determined to be 2.263348651.

7.4881/12 = 0.6273 mm. This length is the basis for the entire system, and is called one potrzebie of length. The Potrzebie has also been standardized at 3515.3602 wave lengths of the red line in the spectrum of sodium. A partial table of the Potrzebie System, the measuring system of the future, is given below:

<table>
<thead>
<tr>
<th>TIME</th>
<th>FORCE</th>
<th>ENERGY AND WORK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 average rotation of the earth = 1 clarke (ad)</td>
<td>1 wavezone (ad)</td>
<td>10600 w = 1 horse (ad)</td>
</tr>
<tr>
<td>100 w = 1 minute (ad)</td>
<td>100 w = 1 word (ad)</td>
<td>100 w = 1 word (ad)</td>
</tr>
<tr>
<td>10 w = 1 digit (ad)</td>
<td>10 w = 1 digit (ad)</td>
<td>10 w = 1 digit (ad)</td>
</tr>
<tr>
<td>1 µ = 1 centesimal (bow)</td>
<td>1 µ = 1 centesimal (bow)</td>
<td>1 µ = 1 centesimal (bow)</td>
</tr>
</tbody>
</table>

DATE: October 1, 1920

*Note: The system is based on a unit of force (ad) and a unit of energy (ad). The system is designed to simplify and standardize measurements.*
… not necessarily stalking Don Knuth
MAN, YOU'RE BEING INCONSISTENT WITH YOUR ARRAY INDICES. SOME ARE FROM ONE, SOME FROM ZERO.

DIFFERENT TASKS CALL FOR DIFFERENT CONVENTIONS. TO QUOTE STANFORD ALGORITHMS EXPERT DONALD KNUUTH, "WHO ARE YOU? HOW DID YOU GET IN MY HOUSE?"

WAIT, WHAT?

WELL, THAT'S WHAT HE SAID WHEN I ASKED HIM ABOUT IT.
1. Biographical and Personal Information

Donald E. Knuth, born January 10, 1938, Milwaukee, Wisconsin; U. S. citizen. Chinese name 高德納 (pronounced Gao Denä or Ko Tokuno).

2. Academic History

Case Institute of Technology, September 1956–June 1960; B.S., summa cum laude, June, 1960; M.S. (by special vote of the faculty), June 1960.
Time matters...

But counting all steps is too precise.

New machines make code run faster...

... but not more efficiently!

To capture the efficiency of an algorithm...

... or the difficulty of a problem!

Big-O
Big O definition

\[ f \in O(g) \text{ means } \]

\[ (\exists c) \ (\exists N) \ (\forall n > N) \ f(n) < c \cdot g(n) \]

- There exists a coefficient \( c \)
- And a threshold \( N \)
- Such that, for all input sizes bigger than that threshold, \( N \)
- \( g \) is bigger than \( f \) up to a constant factor

To claim that \( 42n^2 + 100n \in O(n^2) \)

We need to find \( c \) and \( N \) such that for all \( n > N \),

\[ 42n^2 + 100n < cn^2 \]
Big O in practice

$f \in O(g)$ means

g is bigger than f up to a constant factor
Big \( O \) in practice

Ignore everything but the dominant term

\[
420n^2 + 3n^3 - 201
\]

\[
17 + 300\sqrt{n} + 2\log(n)
\]

\[
\frac{117}{n} + \frac{2^n}{3^n} + 1700
\]

Big-O is formally an upper bound, but in general we want the **BEST** upper bound...
## big-O hierarchy

<table>
<thead>
<tr>
<th>function</th>
<th>Sort by asymptotic size (O)</th>
<th>Match to algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N^3$</td>
<td></td>
<td>A. finding $x$ in an $N$-element balanced binary search tree?</td>
</tr>
<tr>
<td>$N!$</td>
<td></td>
<td>B. finding the <strong>rest</strong> of a length-$N$ list</td>
</tr>
<tr>
<td>$\sqrt{N}$</td>
<td></td>
<td>C. finding $x$ in an length-$N$ list</td>
</tr>
<tr>
<td>$\log(N)$</td>
<td></td>
<td>D. closest pair in $N$ unsorted #s</td>
</tr>
<tr>
<td>$N^N$</td>
<td></td>
<td>E. tautology checking with $N$ vars</td>
</tr>
<tr>
<td>$N^2$</td>
<td></td>
<td>F. is $N$ prime?</td>
</tr>
<tr>
<td>$2^N$</td>
<td></td>
<td>G. $N\times N$ matrix multiplication</td>
</tr>
<tr>
<td>$42$</td>
<td></td>
<td>H. shortest tour of $N$ cities</td>
</tr>
<tr>
<td>$N \cdot \log(N)$</td>
<td></td>
<td>I. sorting a list of $N$ elements</td>
</tr>
</tbody>
</table>

**running times…**

**largest? next? smallest?**

Name(s): ________________________
CS's challenge...

**Big-O Order**

- $n^n$
- $n!$
- $2^n$
- $n^3$
- $n^2$
- $n \log(n)$
- $n$
- $\sqrt{n}$
- $\log(n)$
- $1$

\[ \{
\begin{align*}
    n^n & \quad \text{(Intractable Problems)} \\
    n! & \quad \text{(Exponential)} \\
    2^n & \quad \text{(Exponential)} \\
    n^3 & \quad \text{(Tractable Problems)} \\
    n^2 & \quad \text{(Polynomial)} \\
    n \log(n) & \\
    n & \\
    \sqrt{n} & \\
    \log(n) & \\
    1 & \quad \text{(No Problems!)}
\end{align*}
\]

**Problem-solving strategy:**

Push a problem as far down the hierarchy as possible, then worry about constants... .
# Running Times
(or why people worry about algorithm complexity)

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Complexity</th>
<th>( n = 10 )</th>
<th>100</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>( \log^2 n )</strong></td>
<td>10.361</td>
<td>44.140</td>
<td>99.317</td>
<td>176.54</td>
<td>275.85</td>
<td>397.24</td>
<td></td>
</tr>
<tr>
<td><strong>( \sqrt{n} )</strong></td>
<td>3.162</td>
<td>10</td>
<td>31.622</td>
<td>100</td>
<td>316.22</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>10000</td>
<td>100000</td>
<td>1000000</td>
<td></td>
</tr>
<tr>
<td><strong>n \log n</strong></td>
<td>33.219</td>
<td>664.38</td>
<td>9965.8</td>
<td>132877</td>
<td>1.66*10^6</td>
<td>1.99*10^7</td>
<td></td>
</tr>
<tr>
<td><strong>n^{1.5}</strong></td>
<td>31.6</td>
<td>10^3</td>
<td>31.6*10^4</td>
<td>10^6</td>
<td>31.6*10^7</td>
<td>10^9</td>
<td></td>
</tr>
<tr>
<td><strong>n^2</strong></td>
<td>100</td>
<td>10^4</td>
<td>10^6</td>
<td>10^8</td>
<td>10^10</td>
<td>10^12</td>
<td></td>
</tr>
<tr>
<td><strong>n^3</strong></td>
<td>1000</td>
<td>10^6</td>
<td>10^9</td>
<td>10^12</td>
<td>10^15</td>
<td>10^18</td>
<td></td>
</tr>
<tr>
<td>2(^n)</td>
<td>1024</td>
<td>10^30</td>
<td>10^301</td>
<td>10^3010</td>
<td>10^30103</td>
<td>10^301030</td>
<td></td>
</tr>
</tbody>
</table>

factorial? don't even ask!

imagine the time units ~ nanoseconds: \(10^{**(-9)}\)
So, I have an algorithm...

What's the big-O runtime?
How many terms are here? What’s the big-O sum?

1 + 2 + 4 + 8 + ... + N/4 + N/2 + N

What’s the big-O sum of these values?

1 + 2 + 4 + 8 + ... + 2^{N-2} + 2^{N-1} + 2^N

What’s the big-O sum of these values?

1 + 2 + 3 + 4 + ... + (N-2) + (N-1) + N
How many terms are here? What’s the big-O sum?

1 + 2 + 4 + 8 + … + N/4 + N/2 + N

2N

log(N) terms
How many terms are here? What’s the big-O sum?

\[ 1 + 2 + 4 + 8 + \ldots + 2^{N-2} + 2^{N-1} + 2^N \]

\[ 2^{N+1} \]
How many terms are here? What’s the big-O sum?

\[ 1 + 2 + 3 + 4 + \ldots + (N-2) + (N-1) + N \]
Loop Counting

What's the big-O running time of each of these loops?

for ( int i=0 ; i<N ; ++i )
   1 step of work here...

for ( int i=N ; i>0 ; i/=2 )
   1 step of work here...

Track the Outer Loop index

<table>
<thead>
<tr>
<th>OL</th>
<th>IL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>...i...</td>
<td>...1...</td>
</tr>
<tr>
<td>N-2</td>
<td>1</td>
</tr>
<tr>
<td>N-1</td>
<td>1</td>
</tr>
</tbody>
</table>

Sum up the work Inside the Loop – *using big-O*...
Loop Counting

What's the big-O running time of each of these loops?

for ( int i=0 ; i<N ; ++i )
  for ( int j=0 ; j<N ; ++j )
    1 time step of work here…

for ( int i=1 ; i<=N ; ++i )
  for ( int j=1 ; j<=i ; ++j )
    1 time step of work here…

Track the Outer Loop index

<table>
<thead>
<tr>
<th>OL</th>
<th>IL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N</td>
</tr>
<tr>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>N</td>
</tr>
<tr>
<td>...i...</td>
<td>...N...</td>
</tr>
<tr>
<td>N-2</td>
<td>N</td>
</tr>
<tr>
<td>N-1</td>
<td>N</td>
</tr>
</tbody>
</table>

Sum up the work Inside the Loop – using big-O…
Loop Counting

In general, create a table of work done…

1 time step of work here…

```c
for ( int i=1 ; i<=N ; ++i )
    for ( int j=1 ; j<=i ; ++j )
        1 time step of work here…
```

**Outer Loop index**

**Inner Loop**

Compute the **Inner Loop**'s work

\[
\begin{array}{c|c|c}
\text{OL} & \text{IL} & \text{Compute the Inner Loop's work} \\
\hline
\text{i=1} & 1 & j \sim 1 \text{ to } 1 \\
\text{i=2} & 2 & j \sim 1 \text{ to } 2 \\
... & i & j \sim 1 \text{ to } i \\
\text{i=N} & N & j \sim 1 \text{ to } N \\
\hline
\end{array}
\]

Complicated nested loops may require additional thought for each of these rows…!

\[O(N^2)\]  

Ask Prof. Su for the sum of the series…
What about recursive code?
Recurrence Relations

def sum( L ):
    if L == []: return 0
    else: return L[0] + sum( L[1:] )

counting additions

\[
T(0) = 0 \quad \text{(additions)}
\]

\[
T(N) = \text{recurrence relation}
\]

T(N) is the running time of sum, with input size N
Recurrence Relations

def sum(L):
    if L == []: return 0
    else: return L[0] + sum(L[1:])

T(0) = 0
T(N) = 1 + T(N-1)

T(N) is the running time of sum with input size N

To solve: "Unwind" the relation until all of the terms are known.
Recurrence Unwinding

```python
def g(L):
    if L == []: return 1
    h = len(L)/2
    return sum(L), g(L[:h]), g(L[h:])
```

T(0) = 0

T(N) =

T(N) is now the running time of g, with input size N
Recurrence Unwinding

```python
def g(L):
    if L == []: return 1
    h = len(L)/2
    return sum(L), g(L[:h]), g(L[h:])
```

\[ T(0) = 0 \]
\[ T(N) = N + 2T(N/2) \]

\( T(N) \) is now the running time of \( g \), with input size \( N \).
Recurrence Unwinding

\[ T(0) = 0 \]

\[ T(N) = N + 2T(N/2) \]

\[ T(A) = T( ) = T( ) \]

\[ T(N/2) = \]
Recurrence Unwinding

\[ T(0) = 0 \]

\[ T(N) = N + 2T(N/2) \]

\[ N + 2\left( \frac{N}{2} + 2T(N/4) \right) \]

\[ T(A) = A + 2T(A/2) \]

\[ T(\text{\textvisiblespace}) = \text{\textvisiblespace} + 2T(\text{\textvisiblespace}/2) \]

\[ T(N/2) = \left( \frac{N}{2} + 2T(\frac{N}{4}) \right) \]
Recurrence *Unwinding*

\[ T(0) = 0 \]

\[ T(N) = N + 2T\left(\frac{N}{2}\right) \]

\[ \quad N + 2\left( \frac{N}{2} + 2^{*}T\left(\frac{N}{4}\right) \right) \]
Recurrence Unwinding

\[ T(0) = 0 \]

\[ T(N) = N + 2T(N/2) \]

\[ N + 2\left( \frac{N}{2} + 2^2 T\left(\frac{N}{4}\right) \right) \]

\[ N + 2\left( \frac{N}{2} + 2\left( \frac{N}{4} + 2^3 T\left(\frac{N}{8}\right) \right) \right) \]

\[ N + N + N + N + \ldots \]

\[ O(N \log(N)) \]

T(N) is the running time of \( g \), with input size N
Try it! Finding big-O running times of looping and recursive code...
These are similar in spirit to problem #1 on the homework...

1. for (int i=1 ; i<=N ; i*=2)
   for (int j=0 ; j<i ; j++)
     // O(1) work here...

2. T(1) = 0
   T(N) = 2N + T(N/2)

3. hanoi(1,A,B) => move disk from A to B
   hanoi(N,A,B) => C is the "other peg",
                hanoi(N-1,A,C),
                hanoi(1,A,B),
                hanoi(N-1,C,B)

   T(1) =
   T(N) =

   Write and solve the hanoi recurrence relation, T(N)

4. for (int i=1 ; i<=N ; i*=2)
   for (int j=1 ; j<i ; j*=2)
     // O(1) work here...

   OL       IL
   Values of   i      j work

   i=1
   i=2
   i=4
   i
   i=N

   count each move as 1 operation
Finding big-O running times of looping and recursive code...

These are similar in spirit to problem #1 on the homework...

1. \[
\begin{align*}
T(1) &= 0 \\
T(N) &= 2N + T(N/2)
\end{align*}
\]

\[
\begin{align*}
= 2N + 2(N/2) + T(N/4)
= 2N + 2(N/2) + 2(N/4) + T(N/8)
= 2N + 2(N/2) + 2(N/4) + \ldots + 2(2) + T(1)
= 2N + N + N/2 + \ldots + 4 + 0
= 4N - 4
\end{align*}
\]

\[O(N)\]

2. \[
\begin{align*}
T(1) &= 0 \\
T(N) &= 2N + T(N/2)
\end{align*}
\]

\[
\begin{align*}
= 2N + 2(N/2) + T(N/4)
= 2N + 2(N/2) + 2(N/4) + T(N/8)
= 2N + 2(N/2) + 2(N/4) + \ldots + 2(2) + T(1)
= 2N + N + N/2 + \ldots + 4 + 0
= 4N - 4
\end{align*}
\]

\[O(N)\]

3. \[
\begin{align*}
hanoi(1,A,B) &\Rightarrow \text{move disk from A to B} \\
hanoi(N,A,B) &\Rightarrow C \text{ is the "other peg",} \\
&\quad hanoi(N-1,A,C), \\
&\quad hanoi(1,A,B), \\
&\quad hanoi(N-1,C,B)
\end{align*}
\]

\[
\begin{align*}
T(1) &= 1 \quad \text{(one move for one disk)} \\
T(N) &= 1 + 2T(N-1) \quad \text{(1 base case + 2 big recursions)}
\end{align*}
\]

\[
\begin{align*}
&= 1 + 2(1 + 2T(N-2)) \\
&= 1 + 2(1 + 2(1 + 2T(N-3))) \\
&= 1 + 2(1 + 2(1 + 2(\ldots 2(1 + 2T(1))\ldots))) \\
&= 1 + 2 + 4 + 8 + \ldots + 2^{N-1}
\end{align*}
\]

\[
= 2^{N-1} - 1
\]

\[O(2^N)\]

4. \[
\begin{align*}
\text{for (int i=1 ; i<=N ; i*2) } \\
\text{for (int j=1 ; j<i ; j*2) } \\
// O(1) work here...
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c|c}
\text{OL} & \text{IL} \\
\hline
\text{Values of } i & j \text{ runs} \\
\hline
i=1 & 1 \text{ time} \\
i=2 & 2 \text{ times} \\
i=4 & 4 \text{ times} \\
i=N & i \text{ times} \\
i=N & N \text{ times} \\
\hline
\end{array}
\end{align*}
\]

\[\text{sum this column}\]

\[
\begin{align*}
&= 2N - 1 \text{ work}
\end{align*}
\]

\[O(N)\]

\[
\begin{align*}
\text{for (int i=1 ; i<=N ; i*2) } \\
\text{for (int j=1 ; j<i ; j*2) } \\
// O(1) work here...
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c|c}
\text{OL} & \text{IL} \\
\hline
\text{Values of } i & j \text{ runs} \\
\hline
i=1 & 0 \text{ times} \\
i=2 & 1 \text{ times} \\
i=4 & 2 \text{ times} \\
\vdots & \ldots \text{log(N) times} \\
i=N & \text{log(N) times} \\
\hline
\end{align*}
\]

\[\text{sum up this column}\]

\[
\begin{align*}
&= \text{log}(N)(\text{log}(N)+1) / 2
\end{align*}
\]

\[O(\text{log}^2(N))\]
Strategies for fast-algorithm design...

**Brute Force** -- always consider it first!

How valuable is computer time?

**Divide and Conquer**

Divide the problem recursively and then reassemble the pieces

**Use data structures**

Remembering previous function calls
Look-up tables of partial results
Donald Knuth?

Slightly eccentric perfectionist + Fan of road signs =

Diamond Signs

During our summer vacation last year, my wife and I amused ourselves by taking leisurely drives in Ohio and photographing every diamond-shaped highway sign that we saw along the roadsides. (Well, not every sign; only the distinct ones.) For provenance, I also stood at the base of each sign and measured its GPS coordinates.

This turned out to be even more fun than a scavenger hunt, so we filled in some gaps when we returned to California. And we intend to keep adding to this collection as we drive further, although we realize that we may have to venture to New England in order to see 'FROST HEAVES'.

Here are the images of our collection so far.

my favorites...
Donald Knuth?

Slightly eccentric perfectionist + Fan of road signs =

Diamond Signs

During our summer vacation last year, my wife and I amused ourselves by taking leisurely drives in Ohio and photographing every diamond-shaped highway sign that we saw along the roadsides. (Well, not every sign; only the distinct ones.) For provenance, I also stood at the base of each sign and measured its GPS coordinates.

This turned out to be even more fun than a scavenger hunt, so we filled in some gaps when we returned to California. And we intend to keep adding to this collection as we drive further, although we realize that we may have to venture to New England in order to see 'FROST HEAVES'.

Here are the images of our collection so far.
Donald Knuth? www-cs-faculty.stanford.edu/~knuth

Slightly eccentric perfectionist + Fan of road signs =

Diamond Signs

During our summer vacation last year, my wife and I amused ourselves by taking leisurely drives in Ohio and photographing every diamond-shaped highway sign that we saw along the roadsides. (Well, not every sign; only the distinct ones.) For provenance, I also stood at the base of each sign and measured its GPS coordinates.

This turned out to be even more fun than a scavenger hunt, so we filled in some gaps when we returned to California. And we intend to keep adding to this collection as we drive further, although we realize that we may have to venture to New England in order to see 'FROST HEAVES'.

Here are the images of our collection so far.
Sorting out big-O!

Some fruit flies are less equal than others...

This past journal illustration by T.C. Kaufman and colleagues from Indiana University shows how when the ‘pb’ (proboscipedia) gene in the fruit fly – the fruit fly’s homologue or equivalent gene to the Hoxa2 gene being studied by Nazarali, Doucette and Wolf – is mutated, it causes a transformation of the fruit fly’s mouth parts (slide A) into legs (slide B).

Sorting is the *drosophila* of computer science…
Sorting Algorithms

the *fruit flies* of complexity theory...

**List of length** $n$

- **PermutationSort**
  - **Check:**
    - 3 1 2
    - 3 2 1
    - 1 2 3
    - 1 3 2
    - 2 1 3
    - 2 3 1

- **PrologSort**
  - check all permutations until sorted

- **BogoSort!**
  - shuffle numbers
  - check if sorted
  - if not, repeat....

**List of length** $n$

- **Check:**
  - 3 1 2
  - running time ??
Sorting Algorithms

the *fruit flies* of complexity theory...

**PermutationSort**
- List of length n

**PrologSort**
- check all permutations until sorted

**BogoSort!**
- shuffle numbers
- check if sorted
- if not, repeat….

Check:
- 3 1 2
- 3 2 1
- 1 2 3
- 1 3 2
- 2 1 3
- 2 3 1

List of length n
- 3 1 2

Running time ??

**Slow:**
- 1 1 1
- 1 1 2
- 1 1 3
- 1 2 1
- 1 2 2
- 1 2 3
- 1 3 1
- 1 3 2
- 1 3 3
- 2 1 1
- 2 1 2
- 2 1 3
- 2 2 1
- 2 2 2
- 2 2 3
- 2 3 1
- 2 3 2
- 2 3 3
- 3 1 1
- 3 1 2
- 3 1 3
- 3 2 1
- 3 2 2
- 3 2 3
- 3 3 1
- 3 3 2
- 3 3 3
Polynomial sorting

def cs5sort(L):
    if len(L) < 2: return L

    if L[0] == min(L):
        return [L[0]] + cs5sort(L[1:]):

    return cs5sort(L[1:] + L[0])

cs5 sort
My favorite sorting algorithm!
MinSort + InsertionSort

keep finding the min...  

worst-case?
best-case?

keep inserting elements...  

better than $O(N^2)$?
Strategies for algorithm design

**Brute Force** -- always consider it first!

How valuable is computer time?

**Divide and Conquer**

Divide the problem recursively and then reassemble the pieces

**Use data structures**

Remembering previous function calls
Look-up tables of partial results
MergeSort

Recurrence relation?

even better than $O(N\log(N))$?
Can we sort faster than $N \log(N)$?

Order

- $n^n$
- $n!$
- $2^n$
- $n^3$
- $n^2$
- $n \log(n)$
- $n$
- $\sqrt{n}$
- $\log(n)$
- $1$

\begin{align*}
\text{Intractable Problems (Exponential)} & : n^n, n!, 2^n, n^3, n^2, n \log(n) \\
\text{Tractable Problems (Polynomial)} & : n
\end{align*}

\begin{align*}
\text{No Problems!} & : \sqrt{n}, \log(n), 1
\end{align*}

Problem-solving strategy:
Push a problem as far down the hierarchy as possible, then worry about constants... .
Can we sort faster than $N \log(N)$?

<table>
<thead>
<tr>
<th>Order</th>
<th>Intractable Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^n$</td>
<td></td>
</tr>
<tr>
<td>$n!$</td>
<td></td>
</tr>
<tr>
<td>$2^n$</td>
<td></td>
</tr>
<tr>
<td>$n^3$</td>
<td></td>
</tr>
<tr>
<td>$n^2$</td>
<td></td>
</tr>
<tr>
<td>$n \log(n)$</td>
<td></td>
</tr>
<tr>
<td>$n$</td>
<td></td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td></td>
</tr>
<tr>
<td>$\log(n)$</td>
<td></td>
</tr>
<tr>
<td>$1$</td>
<td>No Problems!</td>
</tr>
</tbody>
</table>