15-252
More Great Ideas in
Theoretical Computer Science
Lecture 3:
Power of Algorithms

September 15th, 2017
Poll

What is the running time as a function of input length?

- logarithmic
- linear
- log-linear
- quadratic
- exponential
- beats me

def isPrime(n):
    if (n < 2):
        return False
    for factor in range(2,n):
        if (n % factor == 0):
            return False
    return True
def isPrime(n):
    if (n < 2):
        return False
    for factor in range(2,n):
        if (n % factor == 0):
            return False
    return True

# iterations: $\approx n$

$n = 2^{\log_2 n} = 2^{\text{len}(n)}$
Algorithms with number inputs

Algorithms on numbers involve **BIG** numbers.

This is actually still small. Imagine having millions of digits.
Algorithms with number inputs

\[ B = 5693030020523999993479642904621911725098567020556258102766251487234031094429 \]

\[ B \approx 5.7 \times 10^{75} \quad (5.7 \text{ quattorvigintillion}) \]

**Definition:** \( \text{len}(B) = \# \text{ bits to write } B \)

\[ \eta \approx \log_2 B \]

For \( B = 5693030020523999993479642904621911725098567020556258102766251487234031094429 \)

\[ \text{len}(B) = 251 \]
def sum(A, B):
    for i from 1 to B do:
        A += 1
    return A

What is the running-time of this algorithm?
Integer Addition

\[
\begin{array}{c}
36185027886661311069865932815214971104 \\
+ 65743021169260358536775932020762686101 \\
\hline
101928049055921669606641864835977657205
\end{array}
\]

\[# \text{ steps to produce } C \text{ is } O(n)\]
Integer Multiplication

\[ A = 3618502788661311069865932815214971104 \]
\[ B = 5932020762686101 \]

\[ \begin{array}{c}
\times \\
\hline
A & 5932020762686101 \\
\hline
\end{array} \]

\[ C = 214650336722050463946651358202698404452609868137425504 \]

\# steps: \( O(\text{len}(A) \cdot \text{len}(B)) = O(n^2) \)
You might think:

Probably this is the best, what else can you really do?

A good algorithm designer always thinks:

**How can we do better?**

What algorithm does Python use?
# Integer Multiplication

Given:

\[
x = \begin{array}{cc}
a & b \\
5 & 6 & 7 & 8 \\
c & d \\
1 & 2 & 3 & 4
\end{array}
\]

\[
y = \begin{array}{cc}
a \cdot 10^{n/2} + b \\
c \cdot 10^{n/2} + d
\end{array}
\]

\[
x \cdot y = (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d)
\]

\[
= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd
\]

Use recursion!
Integer Multiplication

\[
x = \begin{bmatrix} 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \\ c & d \end{bmatrix}
\]

\[
x = a \cdot 10^{n/2} + b \\
y = c \cdot 10^{n/2} + d
\]

\[
x \cdot y = (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d)
\]

\[
= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd
\]

- Recursively compute \( ac, ad, bc, \) and \( bd. \)
- Do the multiplications by \( 10^n \) and \( 10^{n/2} \)
- Do the additions.

\[
T(n) = 4T(n/2) + O(n)
\]
# Integer Multiplication

## Level 0

- $n$

## Level 1

- $n/2$
- $n/2$
- $n/2$
- $n/2$

## Level 2

- $n/4$
- $n/4$
- $n/4$
- $n/4$

**# distinct nodes at level $j$:** $4^j$

**Work done per node at level $j$:** $c(n/2^j)$

**# levels:** $\log_2 n$

**Total cost:** $\sum_{j=0}^{\log_2 n} cn2^j \in O(n^2)$
Integer Multiplication

\[ x \cdot y = \left( a \cdot 10^{n/2} + b \right) \cdot \left( c \cdot 10^{n/2} + d \right) \]

\[ = ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd \]

Hmm, we don’t really care about \( ad \) and \( bc \).
We just care about their sum.
Maybe we can get away with 3 recursive calls.
Integer Multiplication

\[ x \cdot y = (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d) = ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd \]

\[(a + b)(c + d) = ac + ad + bc + bd\]

- Recursively compute \( ac, bd, (a+b)(c+d) \).
- Then \( ad + bc = (a+b)(c+d) - ac - bd \).

\[ T(n) \leq 3T(n/2) + O(n) \quad \text{Is this better??} \]
Integer Multiplication

Level 0

Level 1

Level 2

# distinct nodes at level j: \(3^j\)

work done per node at level j: \(c(n/2^j)\)

# levels: \(\log_2 n\)

Total cost: \(\sum_{j=0}^{\log_2 n} cn(3^j / 2^j)\)
Integer Multiplication

Level

0

1

2

\[ \text{Total cost: } \sum_{j=0}^{\log_2 n} cn \left( \frac{3^j}{2^j} \right) \leq C n \left( \frac{3^{\log_2 n}}{2^{\log_2 n}} \right) = C 3^{\log_2 n} = C n^{\log_2 3} \in O(n^{\log_2 3}) \]
You might think:

Probably this is the best, what else can you really do?

A good algorithm designer always thinks:

**How can we do better?**

Cut the integer into 3 parts of length \( n/3 \) each. Replace 9 multiplications with only 5.

\[
T(n) \leq 5T(n/3) + O(n)
\]

\[
T(n) \in O(n^{\log_3 5})
\]

Can do \( T(n) \in O(n^{1+\epsilon}) \) for any \( \epsilon > 0 \).
Fastest known: \( n(\log n)2^{O(\log^* n)} \)  

Martin Fürer  
(2007)
Input: 2 $n \times n$ matrices $X$ and $Y$.

Output: The product of $X$ and $Y$.

(Assume entries are objects we can multiply and add.)
Matrix Multiplication

\[
\begin{pmatrix}
  a & b \\
  c & d \\
\end{pmatrix}
\times
\begin{pmatrix}
  e & f \\
  g & h \\
\end{pmatrix}
=
\begin{pmatrix}
  ae+bg & af+bh \\
  ce+dg & cf+dh \\
\end{pmatrix}
\]
Matrix Multiplication

\[ Z[i,j] = (i^{th} \text{ row of } X) \cdot (j^{th} \text{ column of } Y) \]

\[ = \sum_{k=1}^{n} X[i,k] \cdot Y[k,j] \]

Algorithm 1: \( \Theta(n^3) \)
Matrix Multiplication

Algorithm 2: recursively compute 8 products + do the additions.

$\Theta(n^3)$
Matrix Multiplication: Strassen’s Algorithm

Can reduce the number of products to 7.

\[
\begin{array}{cc}
AE+BG & AF+BH \\
\cdots \cdots \cdots \cdots \\
CE+DG & CF+DH \\
\end{array}
\]

\[Z = \]

\[
\begin{align*}
Q1 &= (A+D)(E+G) \\
Q2 &= (C+D)E \\
Q3 &= A(F-H) \\
Q4 &= D(G-E) \\
Q5 &= (A+B)H \\
Q6 &= (C-A)(E+F) \\
Q7 &= (B-D)(G+H) \\
AE+BG &= Q1+Q4-Q5+Q7 \\
AF+BH &= Q3+Q5 \\
CE+DG &= Q2+Q4 \\
CF+DH &= Q1+Q3-Q2+Q6
\end{align*}
\]
Matrix Multiplication: Strassen’s Algorithm

Running Time: \( T(n) = 7 \cdot T(n/2) + O(n^2) \)

\[ \Rightarrow T(n) = O\left(n^{\log_2 7}\right) = O\left(n^{2.81}\right) \]
Matrix Multiplication: Strassen’s Algorithm

Strassen’s Algorithm (1969)

Together with Schönhage (in 1971) did n-bit integer multiplication in time $O(n \log n \log \log n)$
The race for the world record

Improvements since 1969

1978: $O(n^{2.796})$ by Pan
1979: $O(n^{2.78})$ by Bini, Capovani, Romani, Lotti
1981: $O(n^{2.522})$ by Schönhage
1981: $O(n^{2.517})$ by Romani
1981: $O(n^{2.496})$ by Coppersmith, Winograd
1986: $O(n^{2.479})$ by Strassen
1990: $O(n^{2.376})$ by Coppersmith, Winograd

No improvement for 20 years!
The race for the world record

No improvement for 20 years!

2010: $O(n^{2.374})$ by Andrew Stothers (PhD thesis)

2011: $O(n^{2.373})$ by Virginia Vassilevska Williams (CMU PhD, 2008)
The race for the world record

2011: $O(n^{2.373})$ by Virginia Vassilevska Williams

(CMU PhD, 2008)

Current world record:

2014: $O(n^{2.372})$ by François Le Gall
Enormous Open Problem

Is there an \( O(n^2) \) time algorithm for matrix multiplication ??