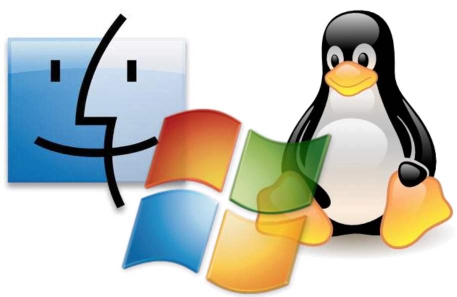
Design and Analysis of Algorithms

Fundamental Concepts

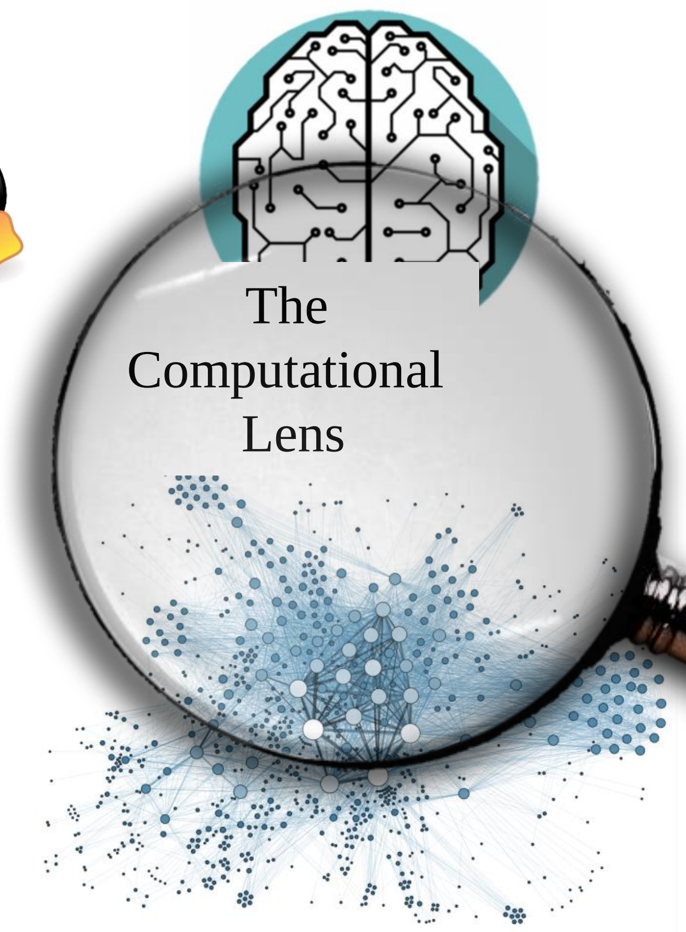
8th April, 2021

Dr. Ramesh Kumar
Associate Professor
Electronic Engineering Department, DUET Karachi

Algorithm Concept

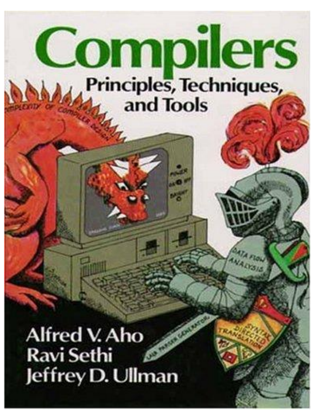


Operating Systems



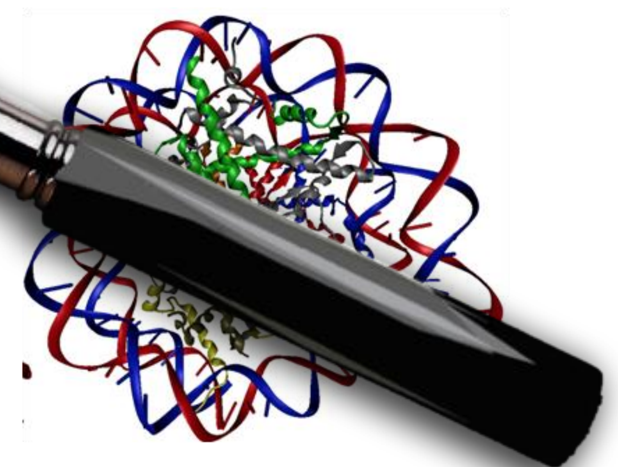
The Computational Lens

Cryptography



Compilers

Communication and Networks

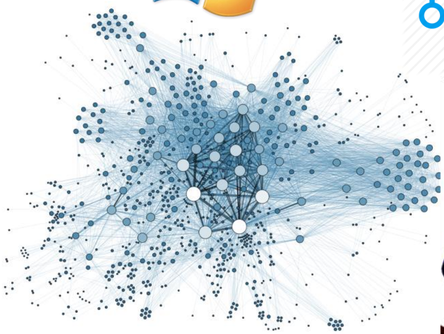


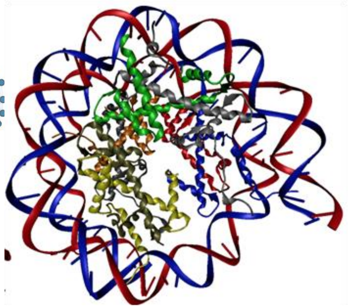
Computational Biology

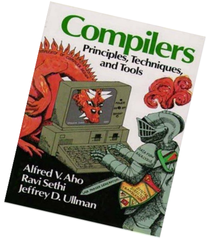
Algorithm Concept

Benefits

- All those things, without CS class numbers
- As we get more and more data and problem sizes get bigger and bigger, algorithms become more and more important.
- Will help you get a job.







Algorithm vs Program

Algorithm	Program
Design	Implementation
Domain-specific Knowledge	Programmer
English with Maths functions	Java, C++
H/W and OS	H/W and OS
Analyze	Testing

Priori Analysis

Language & Hardware independent

Posteriori Analysis

Language & Hardware dependent

Characteristics of Algorithm

- Major Characteristics are:
- Input
  - Should have 0 or more inputs
- Output
  - Should have at least one output
- Definiteness
  - Statements should be clear
- Finiteness
  - Should be finished/completed in finite time
- Effectiveness
  - Should not do the unnecessary statements

Multiplication of Integers

$$\begin{array}{r} 12 \\ \times\ 34 \\ \hline \end{array}$$

$$\begin{array}{r} 1234567895931413 \\ \times\ 4563823520395533 \\ \hline \end{array}$$

Multiplication of Integers

$$\overbrace{1233925720752752384623764283568364918374523856298}^{n}$$
$$\times\ 4562323582342395285623467235019130750135350013753$$

How would you solve this problem?
How long would it take you?

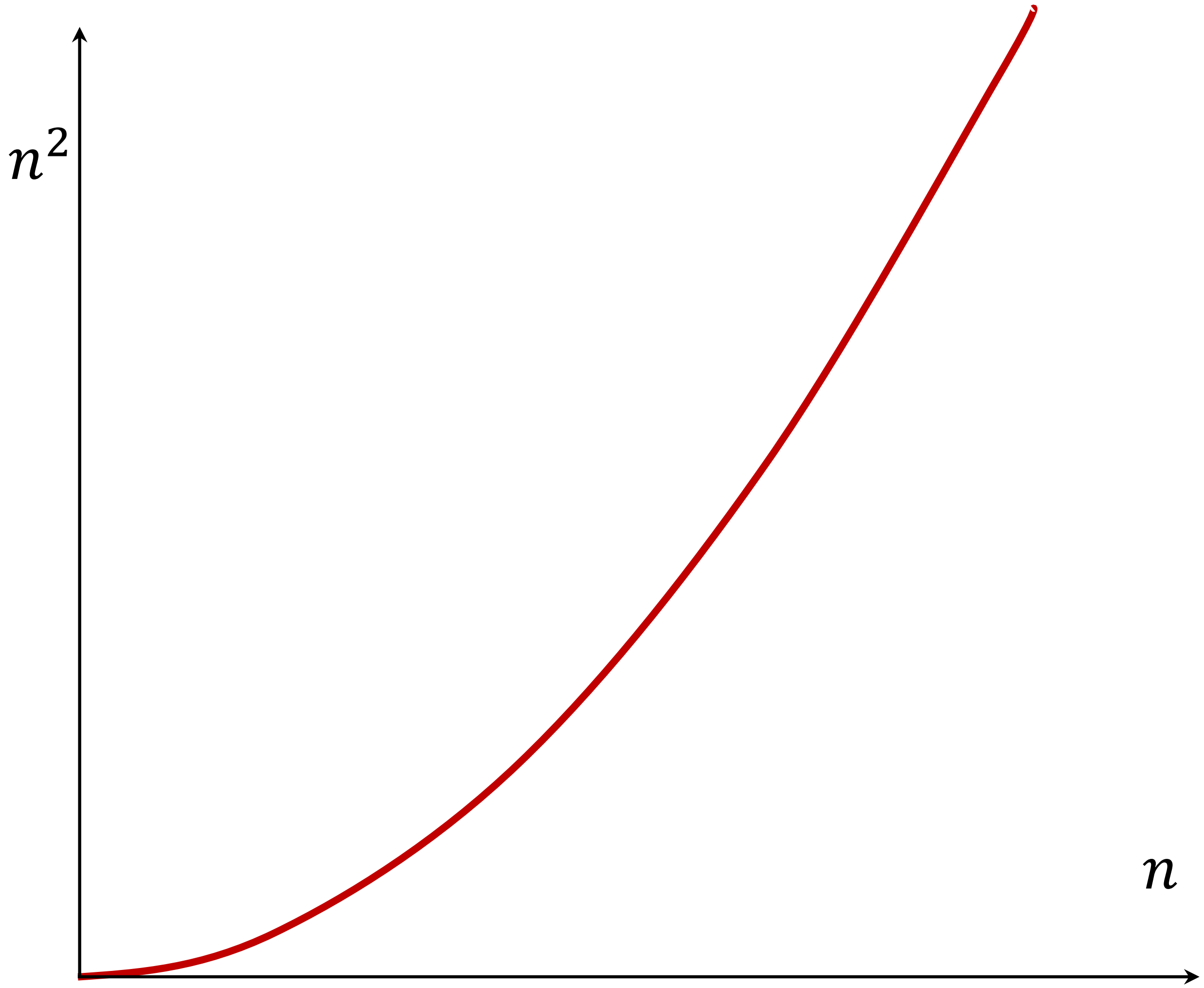
???

About n^2 one-digit operations

At most n^2 multiplications,
and then at most n^2 additions (for carries)
and then I have to add n different 2n-digit numbers…

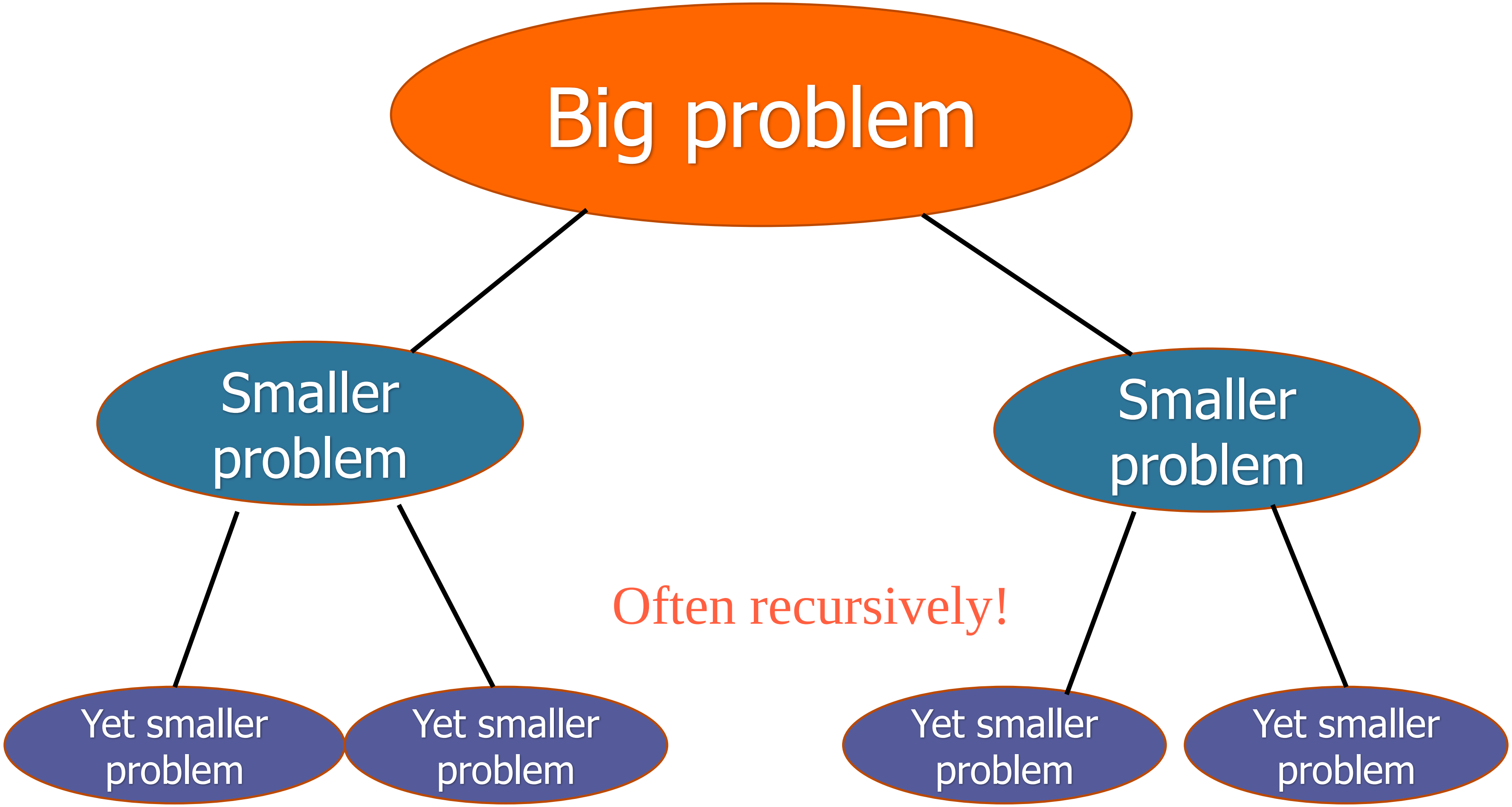
And I take 1 second to multiply two one-digit numbers and .6 seconds to add, so…

Multiplication of Integers



Divide and Conquer

Break problem up into smaller (easier) sub-problems



Algorithm Analysis

Break up an integer:

$$1234 = 12 \times 100 + 34$$

$$1234 \times 5678$$

$$= (12 \times 100 + 34)(56 \times 100 + 78)$$

$$= (\underbrace{12 \times 56}_{1})10000 + (\underbrace{34 \times 56}_{2} + \underbrace{12 \times 78}_{3})100 + (\underbrace{34 \times 78}_{4})$$

One 4-digit multiply

Four 2-digit multiplies

Algorithm Analysis

Break up an n-digit integer:

$$[x_1 x_2 \cdots x_n] = [x_1 x_2 \cdots x_{n/2}] \times 10^{n/2} + [x_{n/2+1} x_{n/2+2} \cdots x_n]$$

$$\begin{aligned} x \times y &= (a \times 10^{n/2} + b)(c \times 10^{n/2} + d) \\ &= \underbrace{(a \times c)}_{1} 10^{n} + (\underbrace{a \times d}_{2} + \underbrace{c \times b)}_{3} 10^{n/2} + \underbrace{(b \times d)}_{4} \end{aligned}$$

One n-digit multiply → Four (n/2)-digit multiplies

How to Analyze an Algorithm

Factors to consider

- Time
- Space
- Network
- Power Consumption

How to Analyze an Algorithm

```
Algorithm swap (a, b){
      temp=a;
      a=b;
      b=temp;
}

Algorithm expression (a, b){
      x=2*a + 4*b + a/b;
      }
```

How to Analyze an Algorithm

```
A[5]={2,4,6,8,10}

Algorithm sum (A,n){
        s=0;
        for (i=0; i<n; i++)
        {
        s=s+A[i];
        }
        return s;
}
```

How to Analyze an Algorithm

A[5]={2,4,6,8,10}

Space

$A \rightarrow n$

$n \rightarrow 1$

$s \rightarrow 1$

$i \rightarrow 1$

```
Algorithm sum (A,n){
        s=0;                    --------> 1
        for (i=0; i<n; i++)     --------> n+1
        {
        s=s+A[i];               --------> n
        }
        return s;               --------> 1
}
```

Time

S(n)=n+3

f(n)=2n+3

How to Analyze an Algorithm

A[]={} & B[]={}

Space

A → n^2
B → n^2
C → n^2
n → 1
i → 1
j → 1

$S(n)=3n^2+3$

```
Algorithm sumofTwoMatrices (A,B,n)
{
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
        {
        c[i,j]=A[i,j]+B[i,j];
        }
    }
}
```

Time

for (i=0; i<n; i++) → $\boxed{n+1}$

for (j=0; j<n; j++) → $\boxed{n*(n+1)}$

c[i,j]=A[i,j]+B[i,j]; → $\boxed{n*n}$

$f(n)=2n^2+2n+1$

How to Analyze an Algorithm

```
Algorithm matrixMultiplication (A,B,C,n)
{
      for (i=0; i<n; i++)
      {
            for (j=0; j<n; j++)
            {
                   c[i,j]=0;
                  for (k=0; k<n; k++)
                  {
                        c[i,j]=c[i,j]+A[i,k]*B[k,j];
                  }
            }
      }
}
```

How to Analyze an Algorithm

- Time Analysis

```
Algorithm Multiply(A,B,n)
{
    for(i=0; i<n; i++)                          --- n+1
    {
        for(j=0; j<n; j++)                      --- (n+1) n
        {
            c[i,j]=0;                           --- n×n
            for(k=0; k<n; k++)                  --- (n+1)×n×n
            {
                c[i,j]=c[i,j]+A[i,k]*B[k,j];    --- n×n×n
            }
        }
    }
}
```

$f(n)=2n^3+3n^2+2n+1$

How to Analyze an Algorithm

- Space Analysis

```
Algorithm Multiply(A,B,n)
{
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            C[i,j]=0;
            for(k=0; k<n; k++)
            {
                C[i,j]=C[i,j]+A[i,k]*B[k,j];
            }
        }
    }
}
```

Space

A — n^2
B — n^2
C — n^2
n — 1
i — 1
j — 1
k — 1

$S(n)=3n^2+4$

$O(n^2)$

Theoretical Analysis

- Uses a high-level description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, *n*.
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment

Counting Primitive Operations

- By inspecting the pseudo code, we can determine the maximum number of primitive/basic operations executed by an algorithm, as a function of the input size

Algorithm ***arrayMax***(***A***, ***n***)	# operations
currentMax ← *A*[0]	2
for (*i* =1; i<n; i++)	$2n$
(i=1 once, i<n n times, i++ (n-1) times: post increment)	
if *A*[*i*] > *currentMax* **then**	$2(n-1)$
currentMax ← *A*[*i*]	$2(n-1)$
return *currentMax*	1
Total	$6n-1$

Estimating Running Time

- Algorithm ***arrayMax*** executes $6\boldsymbol{n} - 1$ primitive operations in the worst case.

  Define:

  $\boldsymbol{a}$ = Time taken by the fastest primitive operation

  $\boldsymbol{b}$ = Time taken by the slowest primitive operation

- Let $\boldsymbol{T}(\boldsymbol{n})$ be worst-case time of ***arrayMax***. Then

$$\boldsymbol{a}\,(6\boldsymbol{n} - 1) \leq \boldsymbol{T}(\boldsymbol{n}) \leq \boldsymbol{b}(6\boldsymbol{n} - 1\,)$$

- Hence, the running time $\boldsymbol{T}(\boldsymbol{n})$ is bounded by two linear functions

Growth Rate of Running Time

- Changing the hardware/ software environment
  - Affects $T(n)$ by a constant factor, but
  - Does not alter the growth rate of $T(n)$
- The linear growth rate of the running time $T(n)$ is an intrinsic/basic property of algorithm ***arrayMax***

Function of Growth rate

Function	Name
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	N log N
N^2	Quadratic
N^3	Cubic
2^N	Exponential

Functions in order of increasing growth rate

Big-Oh Notation

- To simplify the running time estimation, for a function *f(n),* we ignore the constants and lower order terms.

Example: $10n^3+4n^2-4n+5$ is $O(n^3)$.