Lecture 9

CSE 331

Main Steps in Algorithm Design

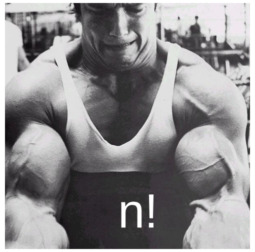
Problem Statement

NRMP
National Resident Matching Program

↓

Problem Definition

↓



Algorithm

↓

"Implementation"

↓

Analysis

Correctness Analysis

Definition of Efficiency

An algorithm is efficient if, when implemented, it runs quickly on real instances

Implemented where?



Platform independent definition

What are real instances?

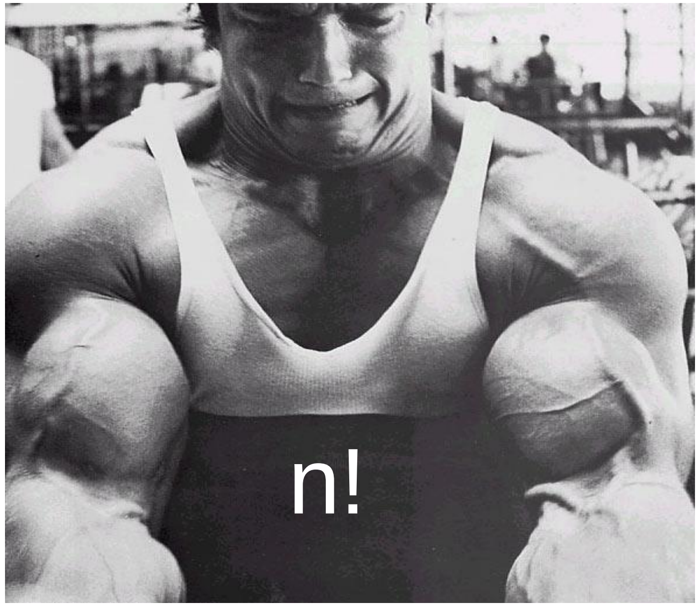
Worst-case Inputs

$N = 2n^2$ for SMP

Efficient in terms of what?

Input size N

Definition-II



Analytically better than brute force

How much better? By a factor of 2?

Definition-III

Should scale with input size

If N increases by a constant factor,
so should the measure

Polynomial running time

At most $c \cdot N^d$ steps ($c>0$, $d>0$ absolute constants)

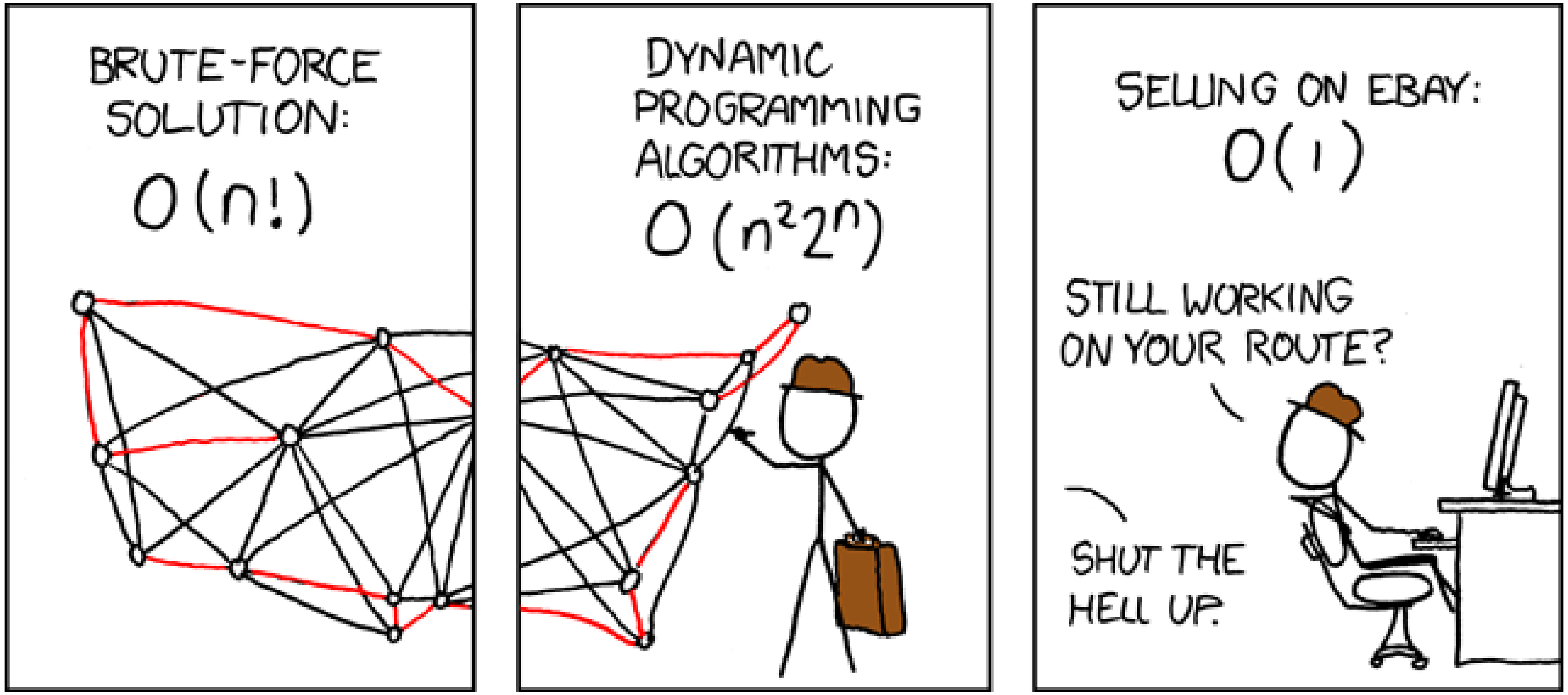
Step: “primitive computational step”

More on polynomial time

Problem centric tractability

Can talk about problems that are not efficient!

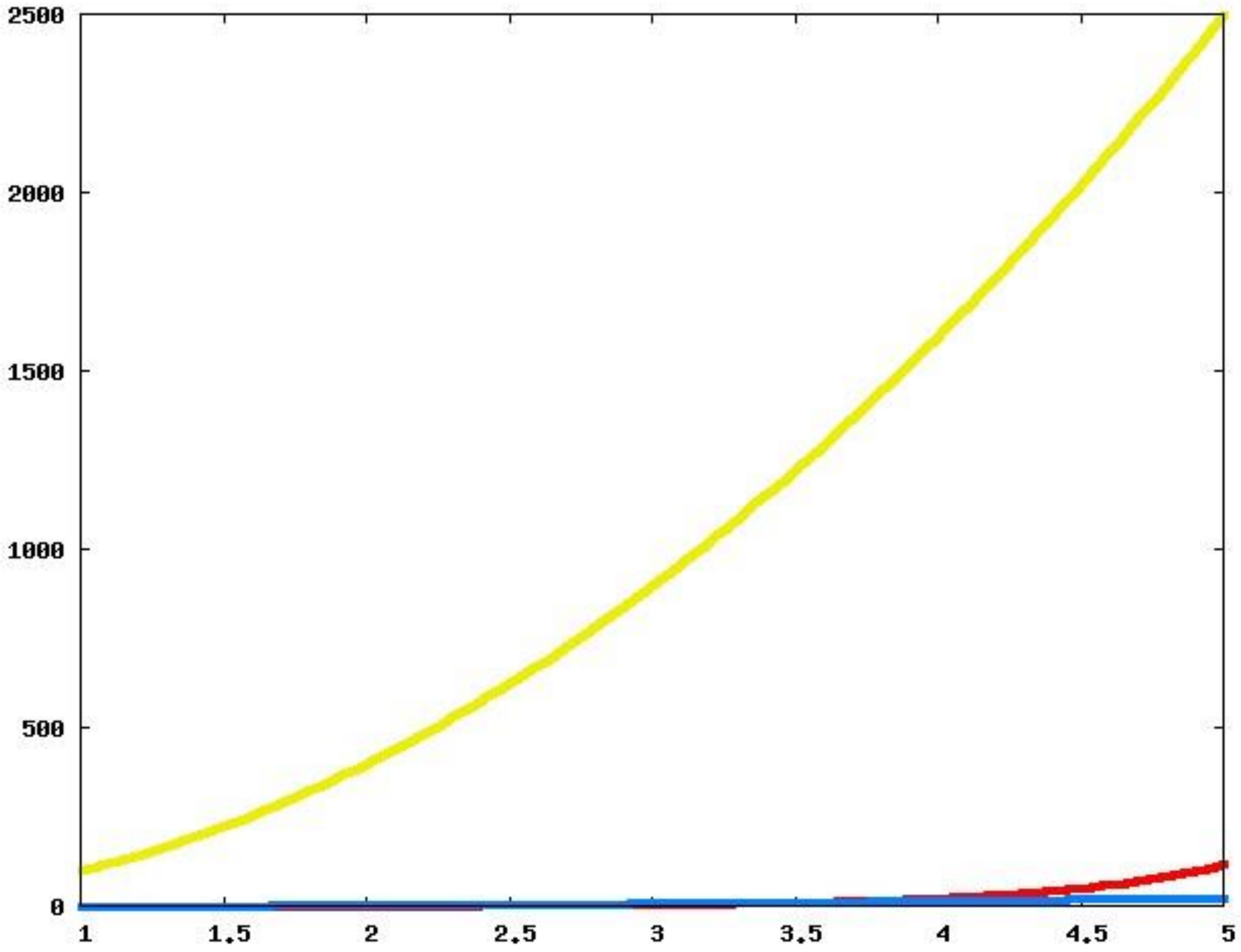
Asymptotic Analysis



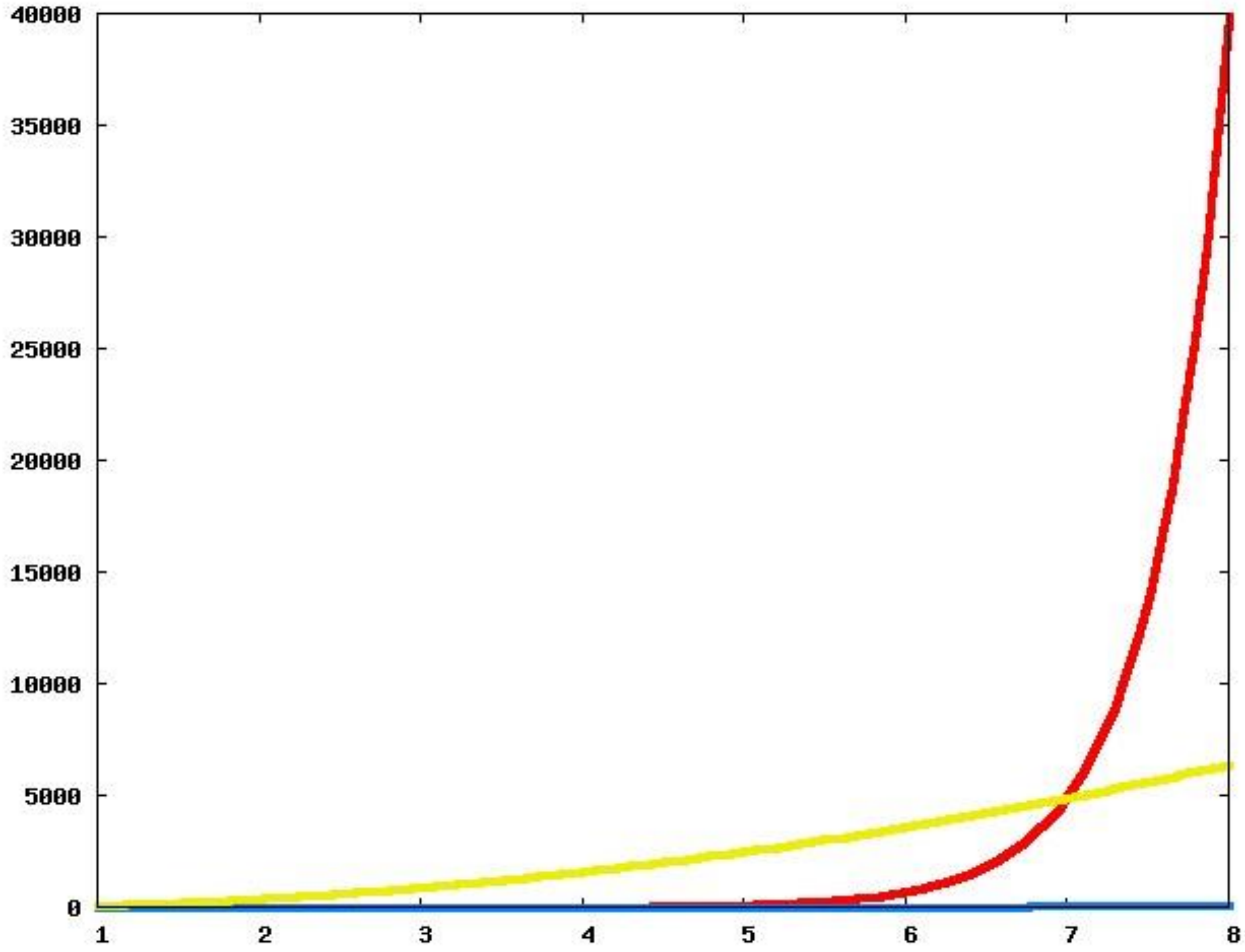
Travelling Salesman Problem

(http://xkcd.com/399/)

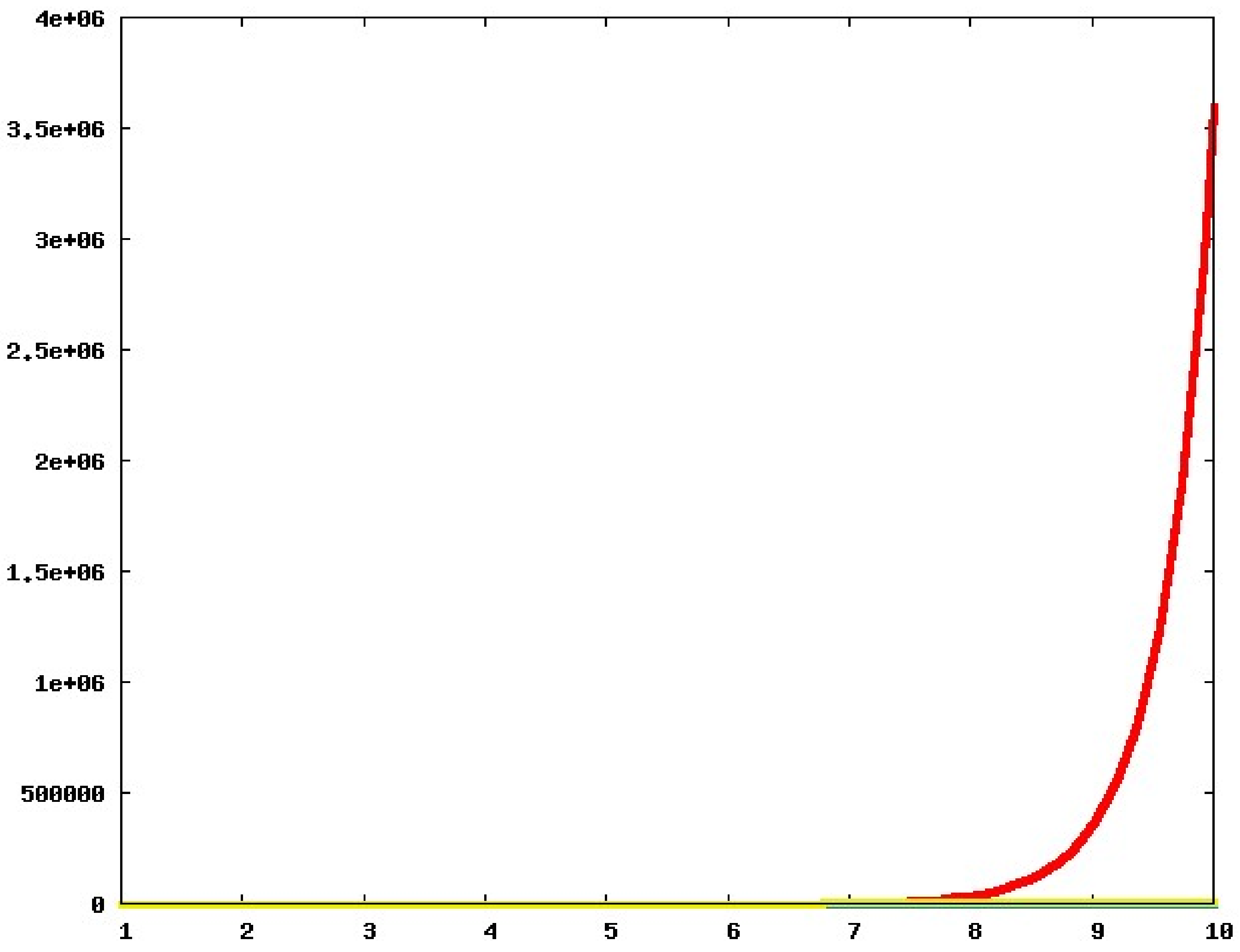
Which one is better?



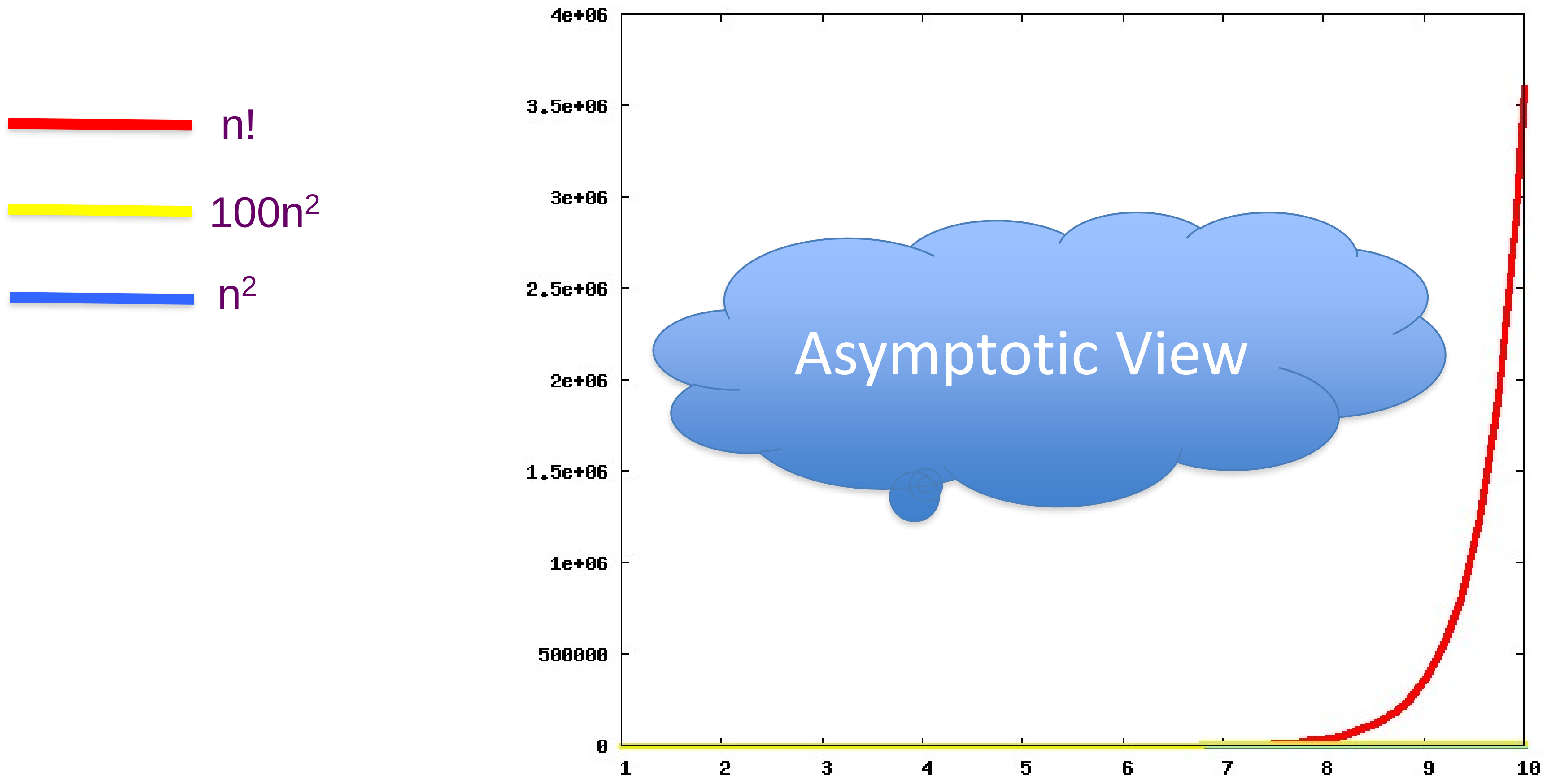
Now?



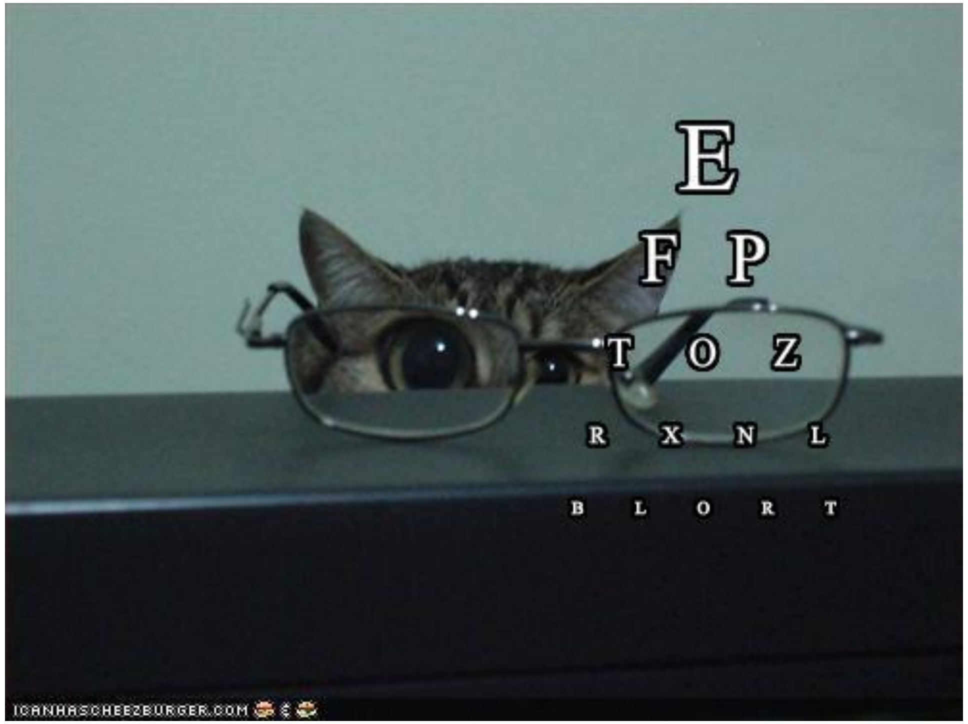
And now?



The actual run times



Asymptotic Notation

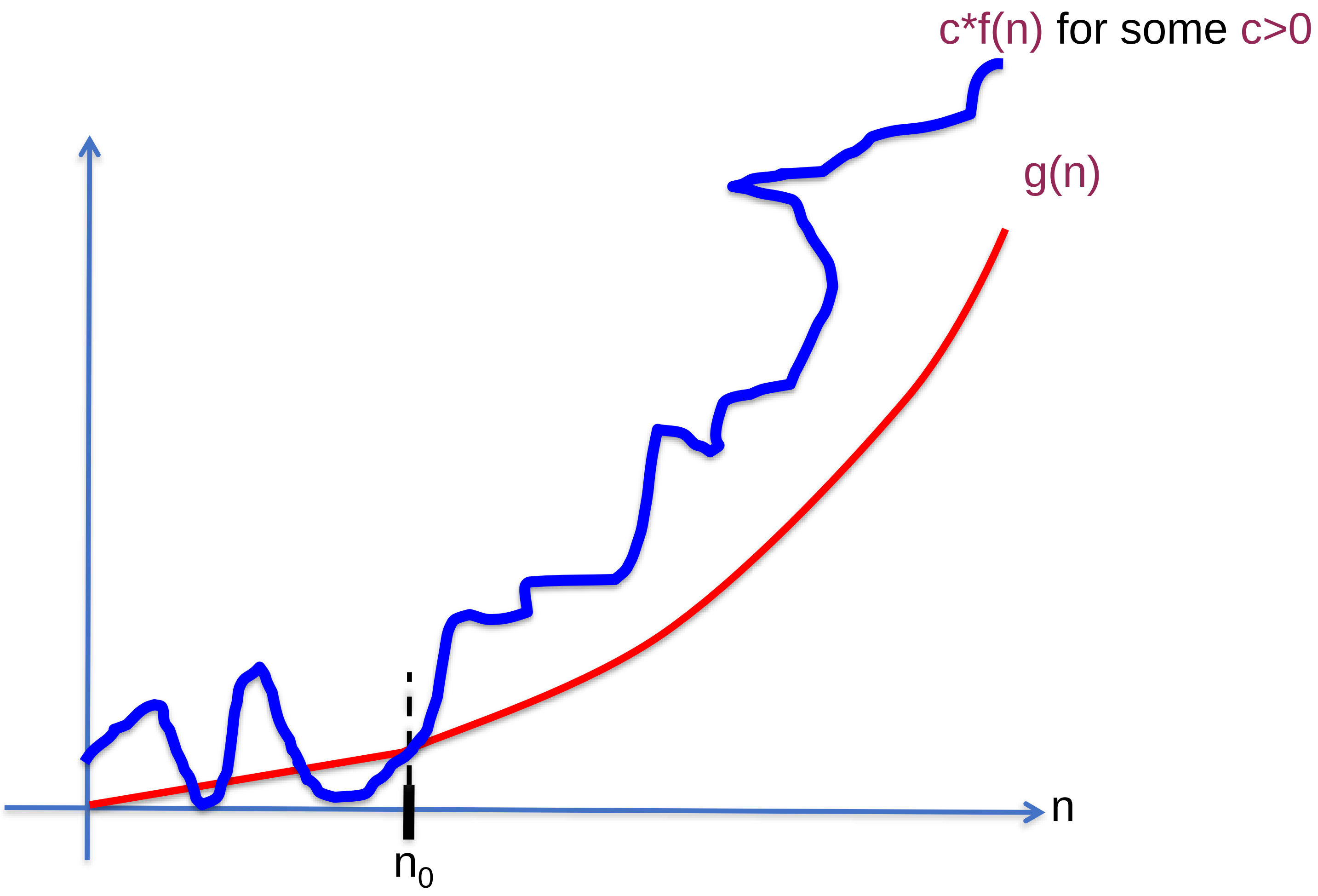


$\leq$ is O with glasses

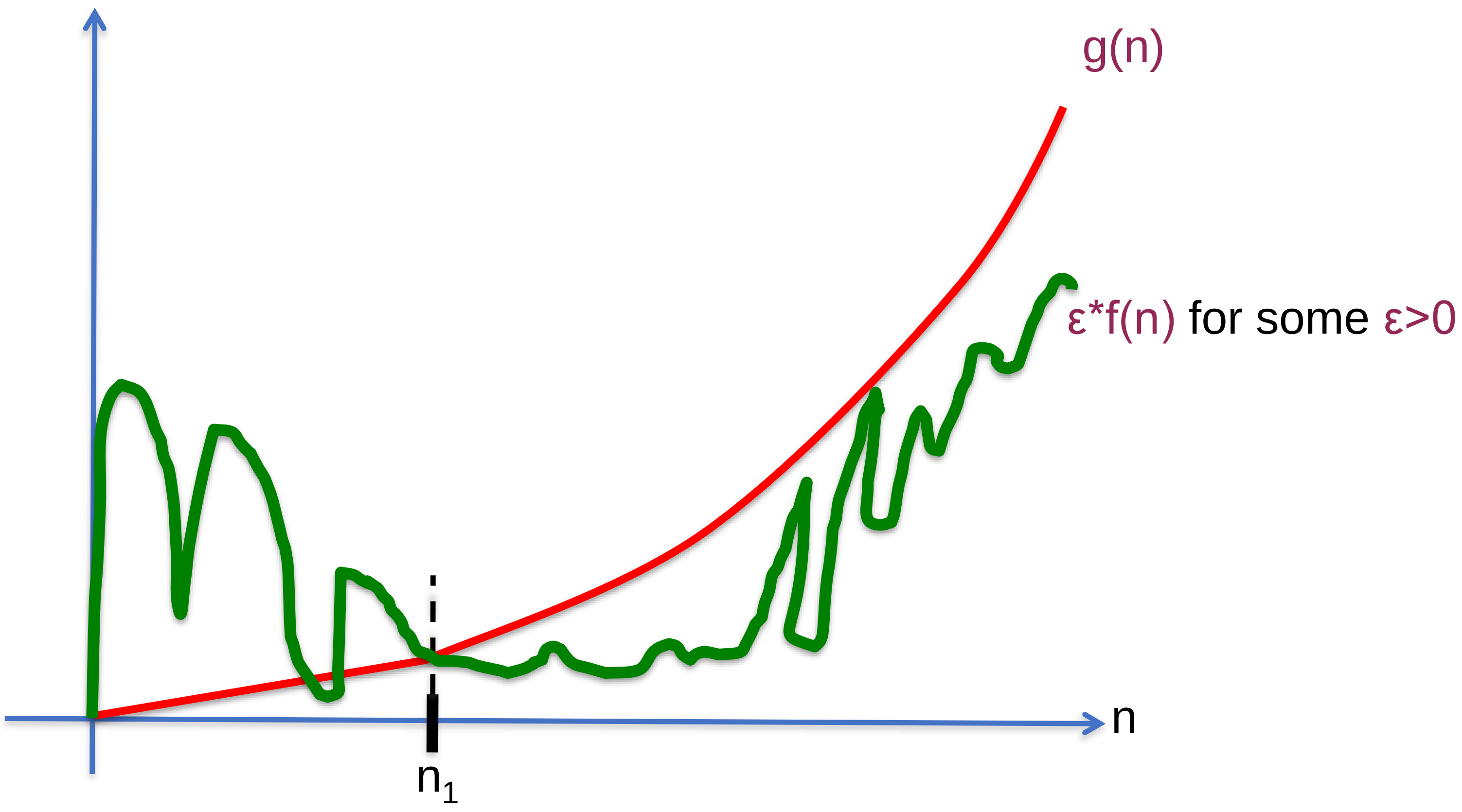
$\geq$ is Ω with glasses

= is Θ with glasses

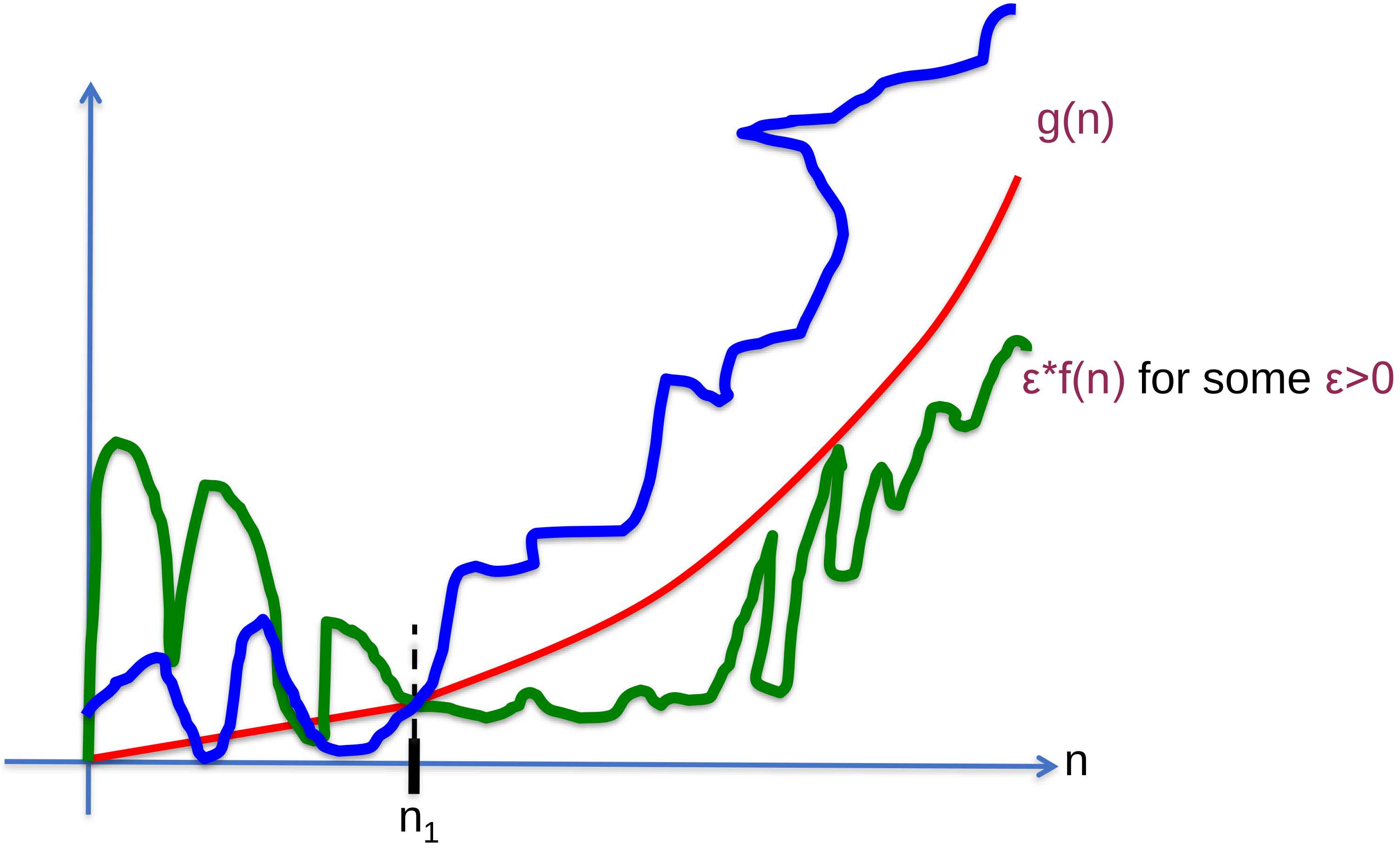
g(n) is O(f(n))



g(n) is Ω(f(n))



g(n) is Θ(f(n))



Properties of O (and Ω)

Transitive

g is O(f) and f is O(h) then
g is O(h)

Additive

g is O(h) and f is O(h) then
g+f is O(h)

Step 1 // O(n) time
Step 2 // O(n) time

Overall:
O(n) time

Multiplicative

g is $O(h_1)$ and f is $O(h_2)$ then
g*f is $O(h_1 * h_2)$

While (loop condition) // $O(n^2)$ iterations
Stuff happens // O(1) time

Overall:
$O(n^2)$ time

Another Reading Assignment

CSE 331 Support Pages

Analyzing the worst-case runtime of an algorithm

Some notes on strategies to prove Big-Oh and Big-Omega bounds on runtime of an algorithm.

The setup

Let $\mathcal{A}$ be the algorithm we are trying to analyze. Then we will define $T(N)$ to be the worst-case run-time of $\mathcal{A}$ over all inputs of size N. Slightly more formally, let $t_{\mathcal{A}}(\mathbf{x})$ be the number of steps taken by the algorithm $\mathcal{A}$ on input $\mathbf{x}$. Then

$$T(N) = \max_{\mathbf{x}:\mathbf{x} \text{ is of size } N} t_{\mathcal{A}}(\mathbf{x}).$$

In this note, we present two useful strategies to prove statements like $T(N)$ is $O(g(N))$ or $T(N)$ is $\Omega(h(N))$. Then we will analyze the run time of a very simple algorithm.

Preliminaries

We now collect two properties of asymptotic notation that we will need in this note (we saw these in class today).

Sections 1.1, 1.2, 2.1, 2.2 and 2.4 in [KT]

Gale-Shapley Algorithm

Intially all men and women are free

While there exists a free woman who can propose

Let w be such a woman and m be the best man she has not proposed to

  w proposes to m

  If m is free

    (m,w) get engaged

  Else (m,w') are engaged

    If m prefers w' to w

      w remains free

    Else

      (m,w) get engaged and w' is free

Output the engaged pairs as the final output

Implementation Steps

How do we represent the input?

How do we find a free woman w?

How would w pick her best unproposed man m?

How do we know who m is engaged to?

How do we decide if m prefers w’ to w?

Gale-Shapley Algorithm

Intially all men and women are free

While there exists a free woman who can propose

At most n^2 iterations

Let w be such a woman and m be the best man she has not proposed to

w proposes to m

If m is free

(m,w) get engaged

Else (m,w’) are engaged

If m prefers w’ to w

w remains free

Else

(m,w) get engaged and w’ is free

O(1) time implementation

Output the engaged pairs as the final output

Implementation Steps

How do we represent the input?

How do we find a free woman w?

How would w pick her best unproposed man m?

How do we know who m is engaged to?

How do we decide if m prefers w' to w?

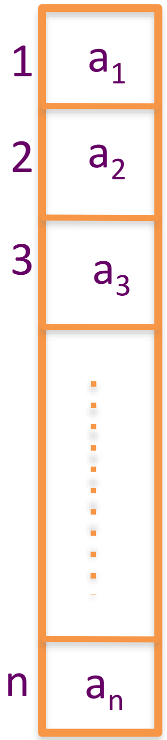
Overall running time

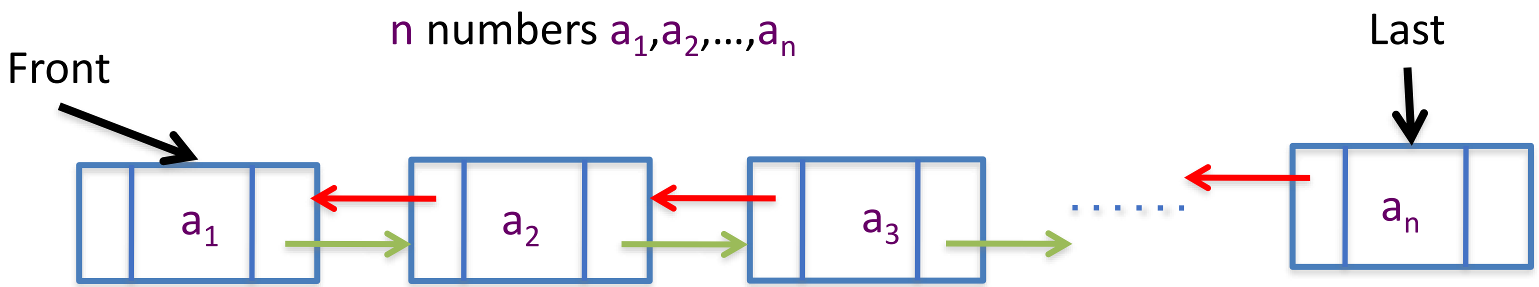
Init(1-4)

+

n^2 X (Query/Update(1-4))

Arrays and Linked Lists





	Array	Linked List
Access ith element	O(1)	O(i)
Is e present?	O(n) (O(log n) if sorted)	O(n)
Insert an element	O(n)	O(1) given pointer
Delete an element	O(n)	O(1) given pointer
Static vs Dynamic	Static	Dynamic

Rest on the board…