

# Algorithms and Programming I

Lecture#1

Spring 2015

# CS 61002 Algorithms and Programming I

- Instructor : Maha Ali Allouzi

Office: 272 MSB

Office Hours: T TH 2:30:3:30 PM

Email: [mallouzi@kent.edu](mailto:mallouzi@kent.edu)

# The Course

- Course Goal: the course is for Student with little or no prior programming Experience. in which an introduction to the algorithms and tools used in computer ; it include programming in a high level languages ( PYTHON and C++)
- Textbook: many online books are on the class website  
[www.cs.kent.edu/~mallouzi](http://www.cs.kent.edu/~mallouzi)

# The Course

- Grading policy:

- Homework & Quizzes: 20%

- Assignment details will be given at the time assigned, and will be posted on the class website
    - Late Assignments will not be accepted .
    - There will be a Quiz at the time you turn your assignment on.

- Exam 1: Feb,5<sup>th</sup> 3:45-5:00 pm 25%
  - Exam 2: Mar5<sup>th</sup> 3:45-5:00 pm 25%
  - Final: May 6<sup>th</sup> 7:45-10:00 am 30%

# The Course

- Grading scale:

93% ≤ A ≤ 100

90% ≤ A- < 93%

87% ≤ B+ < 90

83% ≤ B < 87%

80% ≤ B- < 83%

77% ≤ C+ < 90

73% ≤ C < 77%

70% ≤ C- < 73%

67% ≤ D+ < 90

63% ≤ D < 67%

60% ≤ D- < 63%

0% ≤ f < 60%

# The Course

- Format
  - Two lectures/week
  - Programming Assignments
    - there will be around 10 assignments in the semester
  - Two tests + final exam.

# Cheating

**In short: don't do it!**

*If caught cheating, you will fail this course!!*

# Module 1: introduction to Algorithms

- Module Goal: a rigorous introduction to the design and analysis of algorithms
- Textbook: *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein
  - An excellent reference you should own



# Algorithms

## **Definition of Algorithm:**

- An algorithm is a procedure for solving a mathematical problem in a finite number of steps.
- A step by step method for solving some task.

# Example 1

How to get to School in the morning

- Different ways with same start and end

## **Algorithm 1: walking**

1. Walk out the front door and lock it.
2. Walk 3 miles.
3. Enter the department building.

# Example 2

## **Algorithm 2: Bicycle**

1. Walk out front door and lock it
2. Unlock bicycle , put on helmet.
3. Ride bicycle for 3 miles.
4. Lock up bicycle, take off helmet.
5. Enter department building.

# Example 3

## **Algorithm 3: bus**

1. Walk out front door and lock it.
2. Walk half a mile to the bus stop.
3. Ride the bus.
4. Walk to the office.
5. Enter office building.

# Example 4

## **Algorithm 4: Taxi**

1. Call taxi company.
2. Walk out front door and lock it.
3. Ride the taxi for miles.
4. Enter the department building.

# Compare Algorithms

**Walking**

**free**

**Bicycle**

**cheap**

**Bus**

**cheap**

**Taxi**

**expensive**

# Compare Algorithms...

**Walking**

**slow**

**Bicycle**

**Medium**

**Bus**

**Medium**

**Taxi**

**fast**

# Pseudocode

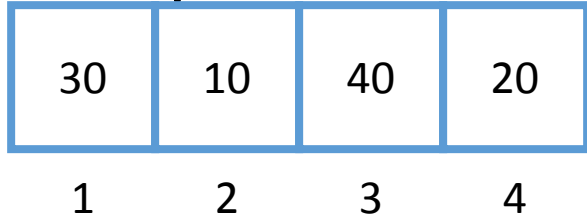
- **Pseudocode** is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a programming language, but is intended for human reading rather than machine reading.



# An Example: Insertion Sort

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

# An Example: Insertion Sort



$i = \emptyset$	$j = \emptyset$	$key = \emptyset$
$A[j] = \emptyset$	$A[j+1] = \emptyset$	




```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

# An Example: Insertion Sort

30	10	40	20
1	2	3	4

$i = 2$	$j = 1$	$key = 10$
$A[j] = 30$		$A[j+1] = 10$

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```



# An Example: Insertion Sort

30	30	40	20
1	2	3	4

$i = 2$	$j = 1$	$key = 10$
$A[j] = 30$		$A[j+1] = 30$

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```



# An Example: Insertion Sort

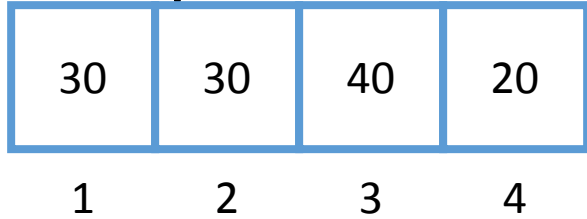
30	30	40	20
1	2	3	4

$i = 2$	$j = 1$	$key = 10$
$A[j] = 30$		$A[j+1] = 30$

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```



# An Example: Insertion Sort



$i = 2$	$j = 0$	$key = 10$
$A[j] = \emptyset$		$A[j+1] = 30$

```
InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
```


⇒

# An Example: Insertion Sort

30	30	40	20
1	2	3	4

$i = 2$	$j = 0$	$key = 10$
$A[j] = \emptyset$	$A[j+1] = 30$	

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```




# An Example: Insertion Sort

10	30	40	20
1	2	3	4

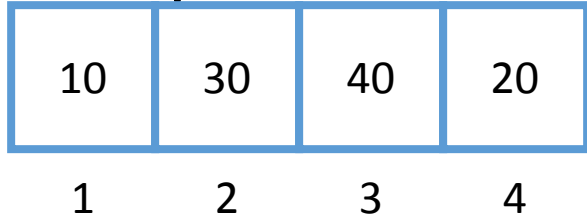
$i = 2$	$j = 0$	$key = 10$
$A[j] = \emptyset$	$A[j+1] = 10$	

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```





# An Example: Insertion Sort

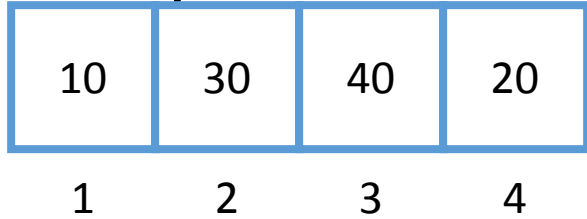


$i = 3$	$j = 0$	$key = 10$
$A[j] = \emptyset$		$A[j+1] = 10$



```
InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
```

# An Example: Insertion Sort

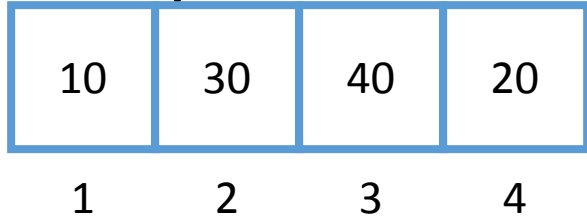


$i = 3$	$j = 0$	$key = 40$
$A[j] = \emptyset$		$A[j+1] = 10$



```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

# An Example: Insertion Sort

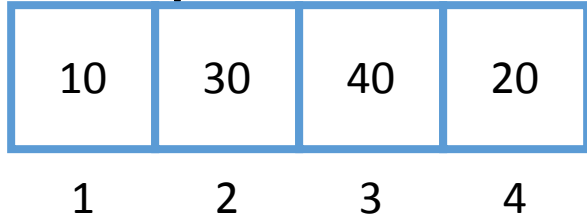


$i = 3$	$j = 0$	$key = 40$
$A[j] = \emptyset$		$A[j+1] = 10$



```
InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
```

# An Example: Insertion Sort



$i = 3$	$j = 2$	$key = 40$
$A[j] = 30$	$A[j+1] = 40$	




```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

# An Example: Insertion Sort

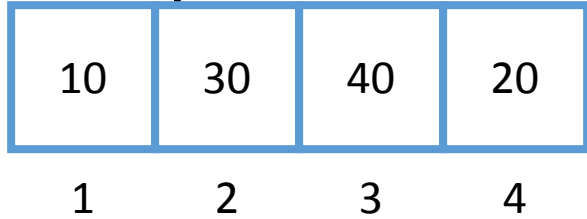
10	30	40	20
1	2	3	4

$i = 3$	$j = 2$	$key = 40$
$A[j] = 30$	$A[j+1] = 40$	

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```




# An Example: Insertion Sort



$i = 3$	$j = 2$	$key = 40$
$A[j] = 30$	$A[j+1] = 40$	

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```



# An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 4$	$j = 2$	$key = 40$
$A[j] = 30$	$A[j+1] = 40$	



```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

# An Example: Insertion Sort

10	30	40	20
1	2	3	4

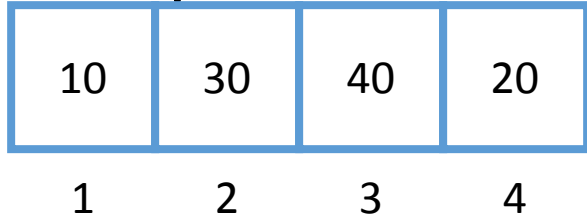
$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 40$	



```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```



# An Example: Insertion Sort

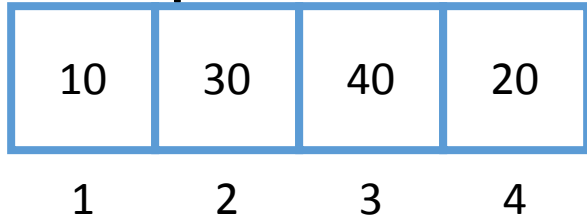


$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 40$	



```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

# An Example: Insertion Sort



$i = 4$	$j = 3$	$key = 20$
$A[j] = 40$	$A[j+1] = 20$	




```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

# An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 4$	$j = 3$	$key = 20$
$A[j] = 40$	$A[j+1] = 20$	

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```



# An Example: Insertion Sort

10	30	40	40
1	2	3	4

$i = 4$	$j = 3$	$key = 20$
$A[j] = 40$	$A[j+1] = 40$	



```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

# An Example: Insertion Sort

10	30	40	40
1	2	3	4

$i = 4$	$j = 3$	$key = 20$
$A[j] = 40$		$A[j+1] = 40$




```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

# An Example: Insertion Sort

10	30	40	40
1	2	3	4

$i = 4$	$j = 3$	$key = 20$
$A[j] = 40$		$A[j+1] = 40$

```
InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
```




# An Example: Insertion Sort

10	30	40	40
1	2	3	4

$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 40$	

```
InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
```




# An Example: Insertion Sort

10	30	40	40
1	2	3	4

$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 40$	

```
InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
```





# An Example: Insertion Sort

10	30	30	40
1	2	3	4

$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 30$	




```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

# An Example: Insertion Sort

10	30	30	40
1	2	3	4

$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 30$	

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```




# An Example: Insertion Sort

10	30	30	40
1	2	3	4

$i = 4$	$j = 1$	$key = 20$
$A[j] = 10$	$A[j+1] = 30$	

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```




# An Example: Insertion Sort

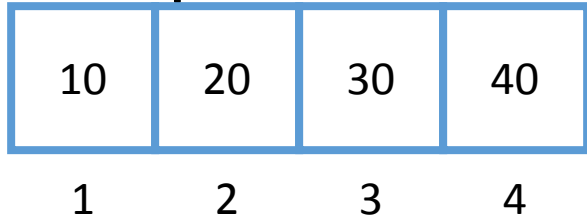
10	30	30	40
1	2	3	4

$i = 4$	$j = 1$	$key = 20$
$A[j] = 10$	$A[j+1] = 30$	

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```



# An Example: Insertion Sort



$i = 4$	$j = 1$	$key = 20$
$A[j] = 10$	$A[j+1] = 20$	

```
InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
```

⇒

# An Example: Insertion Sort

10	20	30	40
1	2	3	4

$i = 4$	$j = 1$	$key = 20$
$A[j] = 10$	$A[j+1] = 20$	

```
InsertionSort(A, n) {  
  for i = 2 to n {  
    key = A[i]  
    j = i - 1;  
    while (j > 0) and (A[j] > key) {  
      A[j+1] = A[j]  
      j = j - 1  
    }  
    A[j+1] = key  
  }  
}
```

Done!

# Analysis of Algorithms

**Analysis of Algorithms**: is the theoretical study of computer program performance and resource usage.

- Study how to make things fast.
- In programming ...What is more important than performance?
  1. Correctness
  2. Simplicity
  3. Maintainability
  4. Robustness of the software
  5. Security.
  6. Modularity .
  7. Scalability
  8. User friendliness !
- Why study algorithm and performance if it is at the bottom of the heap?

Performance

  1. measure the line between the feasible and the unfeasible if its is not fast enough its simply not functional. Or if it uses too much memory its simply not going to work
  2. Correlated with user friendliness ( waiting !!!)

# Asymptotic Performance

- In this course, we care most about *asymptotic performance*
  - How does the algorithm behave as the problem size gets very large?
    - Running time
    - Memory/storage requirements
    - Bandwidth/power requirements/logic gates/etc.



# Running Time

- Number of primitive steps that are executed
  - Except for time of executing a function call most statements roughly require the same amount of time
  - We can be more exact if need be
- Worst case vs. average case
  - ( best case is ....)

# Insertion Sort

<u>Statement</u>	<u>Effort</u>
<b>InsertionSort(A, n) {</b>	
<b>for i = 2 to n {</b>	$c_1 n$
<b>key = A[i]</b>	$c_2(n-1)$
<b>j = i - 1;</b>	$c_3(n-1)$
<b>while (j &gt; 0) and (A[j] &gt; key) {</b>	$c_4 T$
<b>A[j+1] = A[j]</b>	$c_5(T-(n-1))$
<b>j = j - 1</b>	$c_6(T-(n-1))$
<b>}</b>	0
<b>A[j+1] = key</b>	$c_7(n-1)$
<b>}</b>	0
<b>}</b>	

$T = t_2 + t_3 + \dots + t_n$  where  $t_i$  is number of while expression evaluations for the  $i^{\text{th}}$  for loop iteration

# Analysis

- Simplifications
  - Ignore actual and abstract statement costs
  - *Order of growth* is the interesting measure:
    - Highest-order term is what counts
      - Remember, we are doing asymptotic analysis
      - As the input size grows larger it is the high order term that dominates

# Upper Bound Notation

- We say InsertionSort's run time is  $O(n^2)$ 
  - Properly we should say run time is *in*  $O(n^2)$
  - Read O as "Big-O" (you'll also hear it as "order")
- In general a function
  - $f(n)$  is  $O(g(n))$  if there exist positive constants  $c$  and  $n_0$  such that  $f(n) \leq c \cdot g(n)$  for all  $n \geq n_0$
- Formally
  - $O(g(n)) = \{ f(n) : \exists \text{ positive constants } c \text{ and } n_0 \text{ such that } f(n) \leq c \cdot g(n) \forall n \geq n_0$

# Insertion Sort Is $O(n^2)$

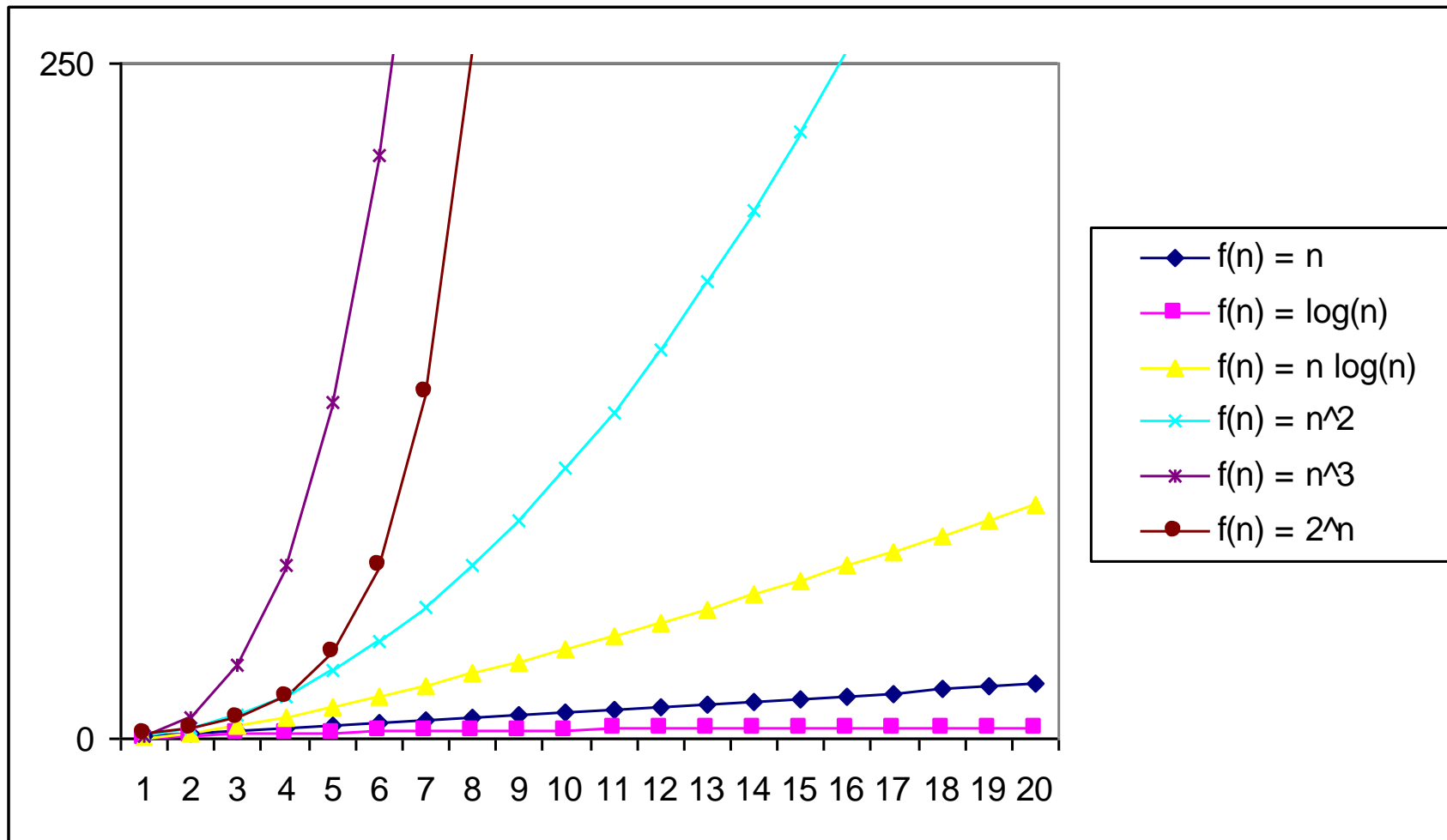
- Proof

- Suppose runtime is  $an^2 + bn + c$ 
  - If any of  $a$ ,  $b$ , and  $c$  are less than 0 replace the constant with its absolute value
- $an^2 + bn + c \leq (a + b + c)n^2 + (a + b + c)n + (a + b + c)$
- $\leq 3(a + b + c)n^2$  for  $n \geq 1$
- Let  $c' = 3(a + b + c)$  and let  $n_0 = 1$

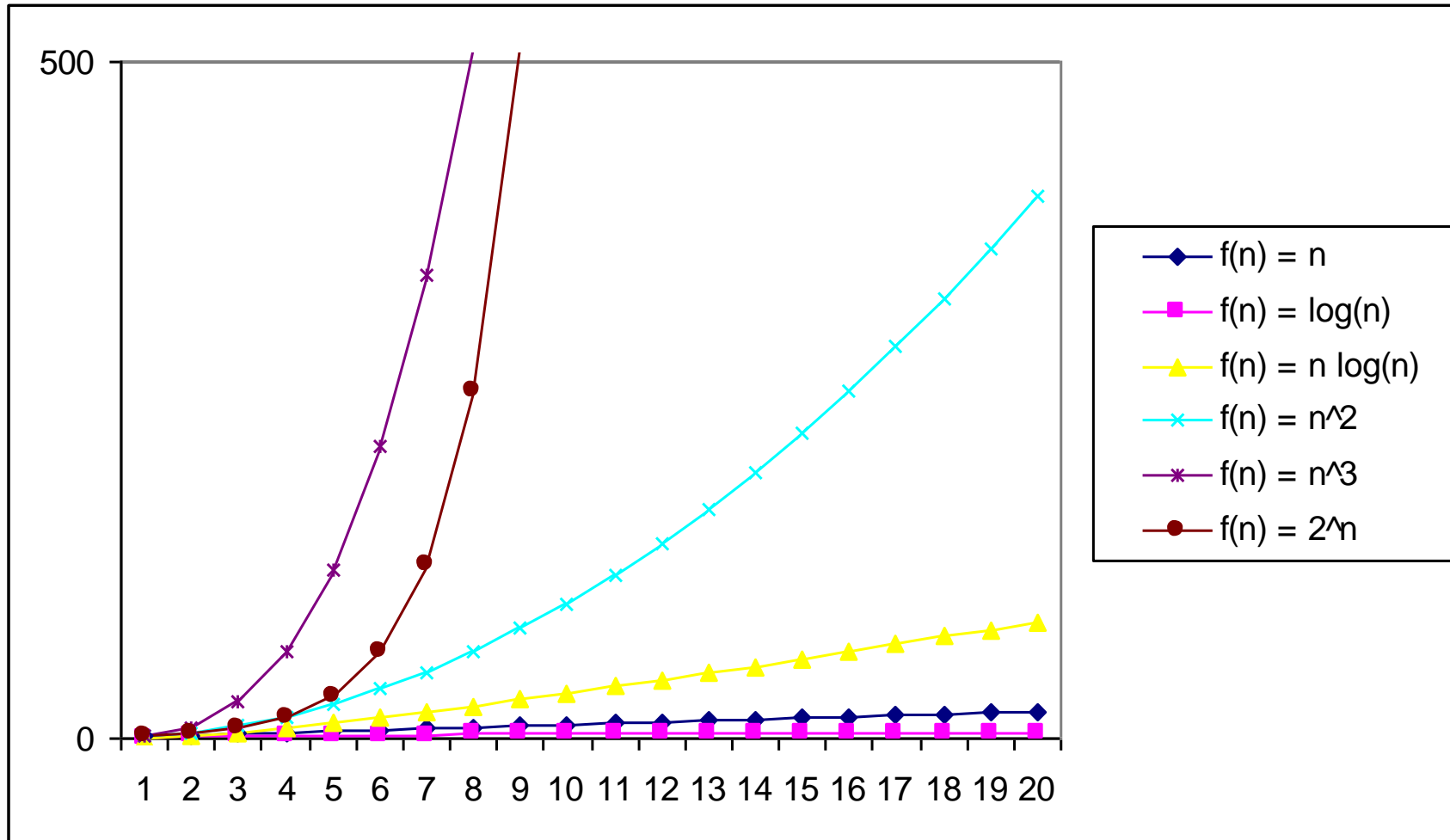
- Question

- Is InsertionSort  $O(n^3)$ ?
- Is InsertionSort  $O(n)$ ?

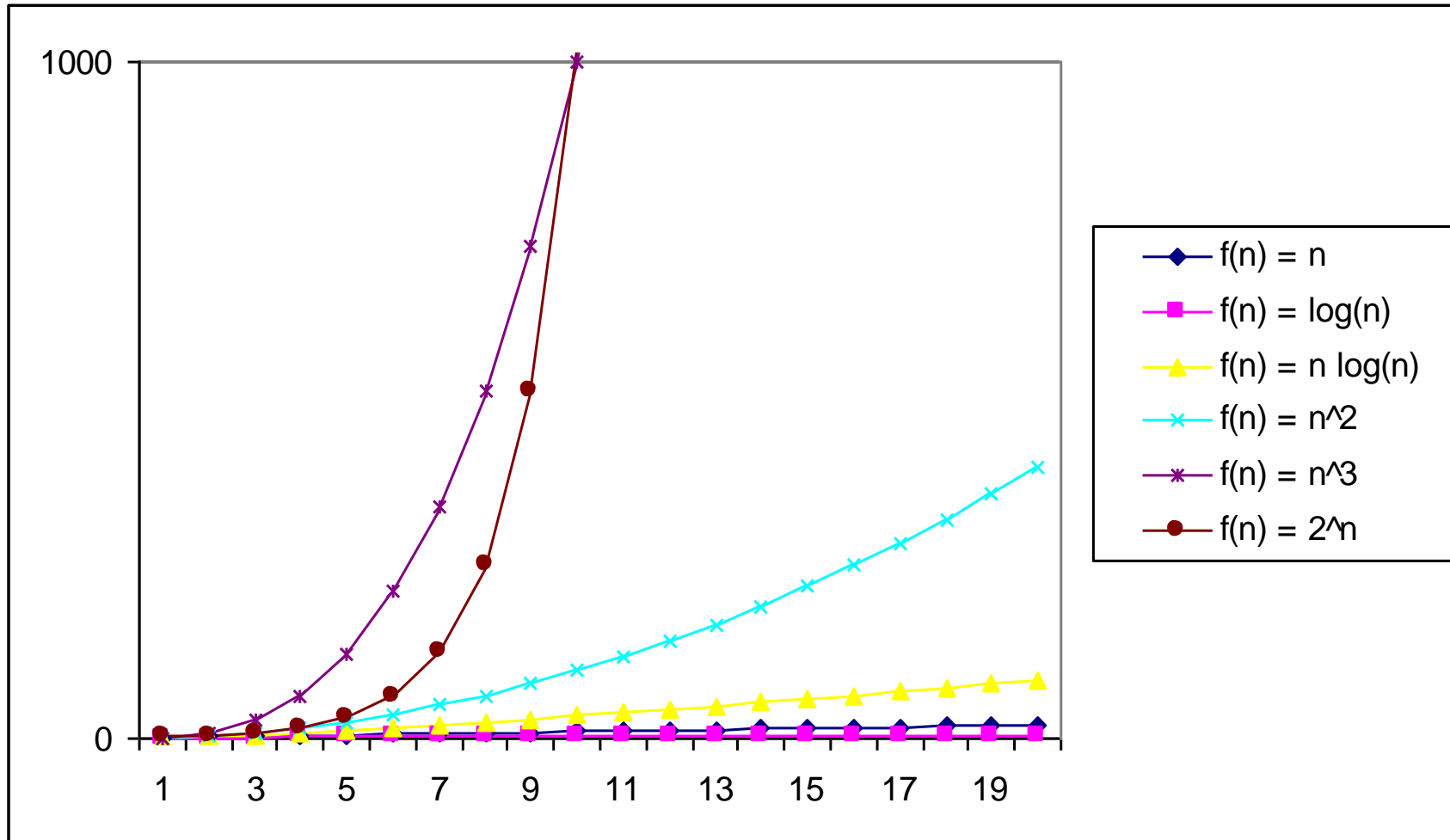
# Practical Complexity



# Practical Complexity

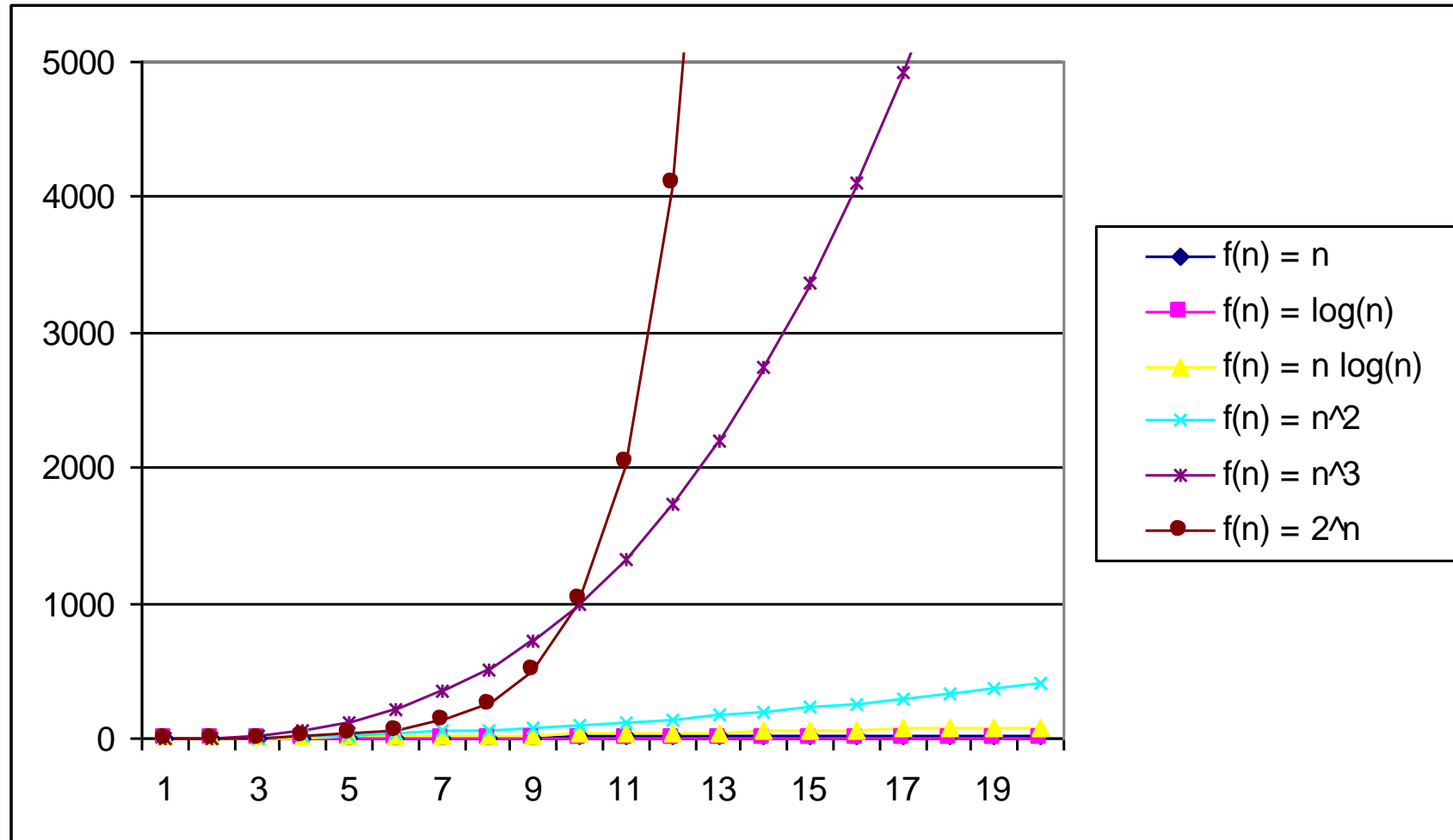


# Practical Complexity

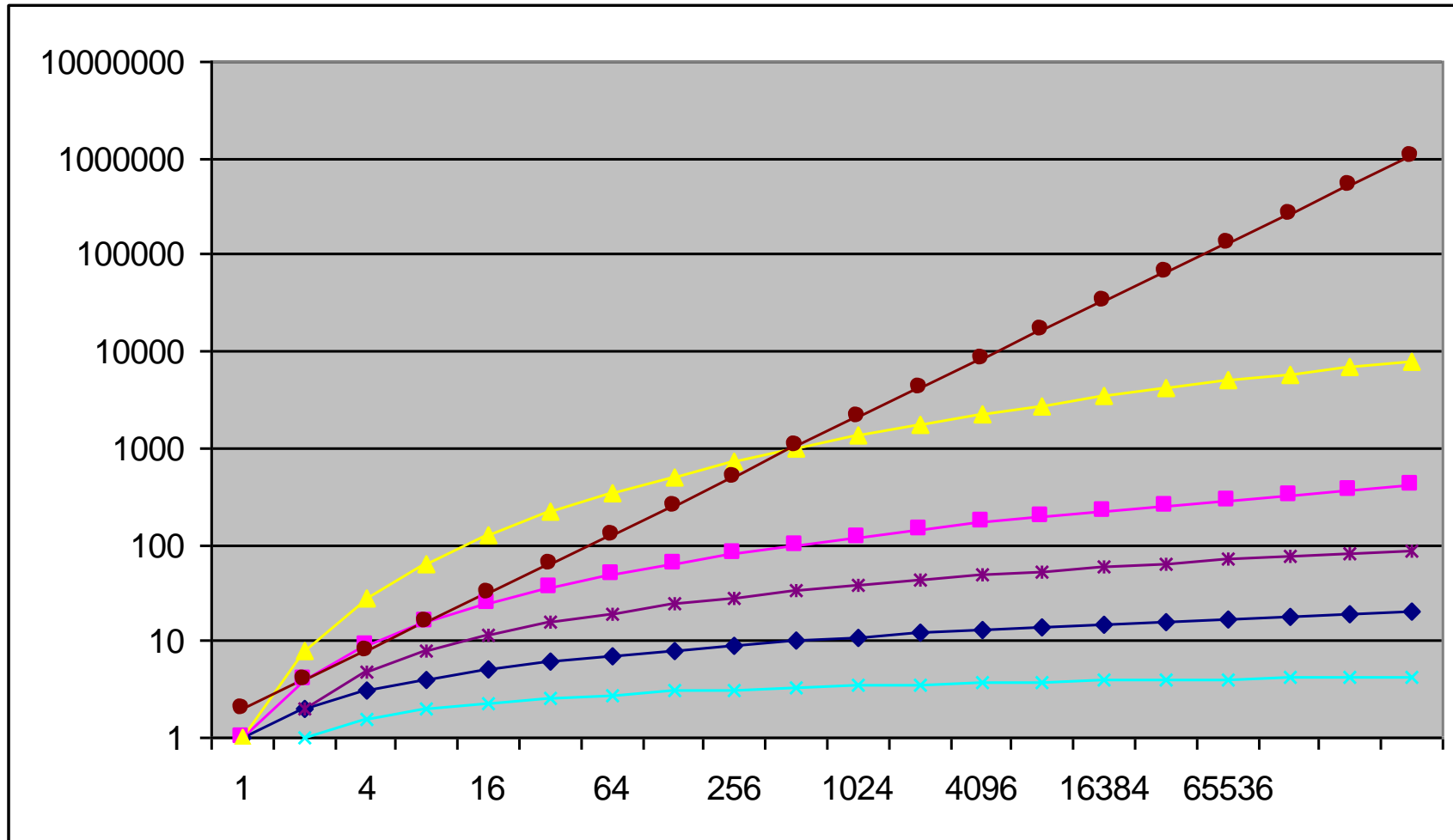




# Practical Complexity



# Practical Complexity



# Other Asymptotic Notations

- A function  $f(n)$  is  $o(g(n))$  if  $\exists$  positive constants  $c$  and  $n_0$  such that
$$f(n) < c g(n) \quad \forall n \geq n_0$$
- A function  $f(n)$  is  $\omega(g(n))$  if  $\exists$  positive constants  $c$  and  $n_0$  such that
$$c g(n) < f(n) \quad \forall n \geq n_0$$
- Intuitively,

▪  $o()$  is like  $<$

▪  $\omega()$  is like  $>$

▪  $\Theta()$  is like  $=$

▪  $O()$  is like  $\leq$

▪  $\Omega()$  is like  $\geq$