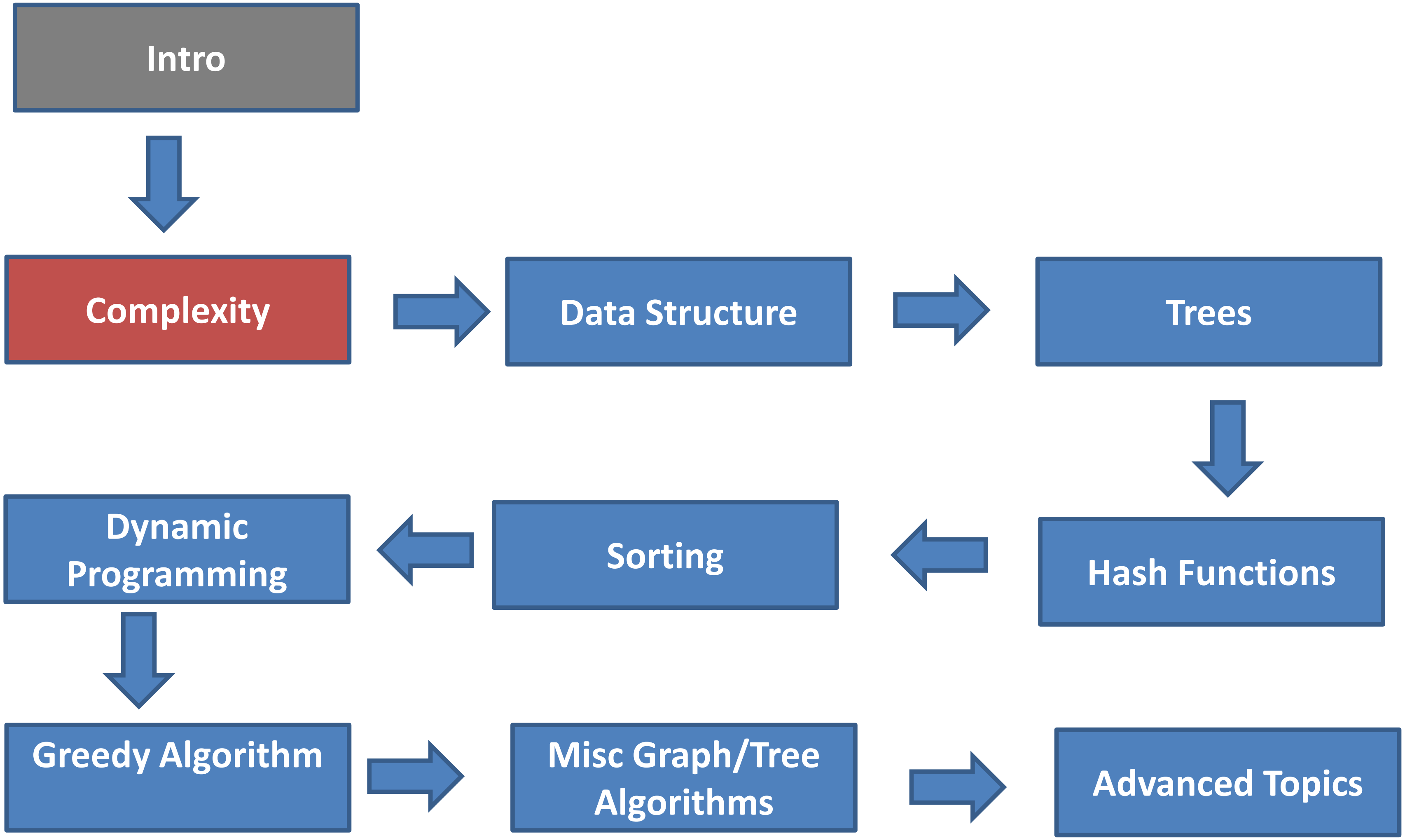
# An Introduction to Algorithms

By

**Hossein Rahmani**

h_rahmani@iust.ac.ir
http://webpages.iust.ac.ir/h_rahmani/



Intro
Complexity
Data Structure
Trees
Hash Functions
Sorting
Dynamic Programming
Greedy Algorithm
Misc Graph/Tree Algorithms
Advanced Topics

# Run-time Analysis

- Depends on
  - input size
  - input quality (partially ordered)
- Kinds of analysis
  - Worst case
  - Average case
  - Best case

# What do we mean by Analysis?

- Analysis is performed with respect to a computational model
- We will usually use a generic uniprocessor random-access machine (RAM)
  - All memory is equally expensive to access
  - No concurrent operations
  - All reasonable instructions take unit time
    - Except, of course, function calls

# Example: Searching

- Assume we have a sorted array of integers, X[1..N] and we are searching for "key"

| Code | Cost | Times |
|---|---|---|
| found = 0; | C0 | 1 |
| i  = 0; | C1 | 1 |
| while (!found && i < N)  { | C2 | 0 <= L < N |
| if (key == X[i]) | C3 | 1 <= L <= N |
| found = 1; | ? | ? |
| i++; | C4 | 1 <= L <= N |
| } | | |

T(n) = C0 + C1 + L*(C2 + C3 + C4), where 1 <= L <= N is the number of times that the loop is iterated.

# Example: Searching

- What's the best case? Loop iterates just once =>
  - T(n) = C0 + C1 + C2 + C3 + C4
- What's the average (expected) case? Loop iterates N/2 times =>
  - T(n) = C0 + C1 + N/2 * (C2 + C3 + C4)
  - Notice that this can be written as T(n) = a + b*n where a, b are constants
- What's the worst case? Loop iterates N times =>
  - T(n) = C0 + C1 + N * (C2 + C3 + C4)
  - Notice that this can be written as T(n) = a + b*n where a, b are constants

# Worst Case Analysis

- We will only look at WORST CASE running time of an algorithm. Why?
  - Worst case is an upper bound on the running time. It gives us a guarantee that the algorithm will never take any longer
  - For some algorithms, the worst case happens fairly often. As in this search example, the searched item is typically not in the array, so the loop will iterate N times
  - The "average case" is often roughly as bad as the "worst case". In our search algorithm, both the average case and the worst case are linear functions of the input size "n"

# Quiz Time


- Target: Complexity Analysis of Insertion Sort
- Make the group of 2
- First, write what you remember from Insertion sort
  - Do not check internet or the previous slides
- Second, explore the complexity analysis
  - Find out the number of times each command runs
  - Discuss the Worst Case and Best Case
- <u>Be careful, You **only** have 15 minutes</u>
- Then, we proceed with the slides

# 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
  }
}
```

*How many times will this loop execute?*

# Insertion Sort

| Statement | Cost | times |
|---|---|---|
| `InsertionSort(A, n) {` | | |
| `for i = 2 to n {` | $c_1$ | n |
| `key = A[i]` | $c_2$ | n-1 |
| `j = i - 1;` | $c_4$ | n-1 |
| `while (j > 0) and (A[j] > key) {` | $c_5$ | $\sum_{j=2}^{n} t_j$ |
| `A[j+1] = A[j]` | $c_6$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| `j = j - 1` | $c_7$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| `}` | 0 | |
| `A[j+1] = key` | $c_8$ | n-1 |
| `}` | 0 | |
| `}` | | |

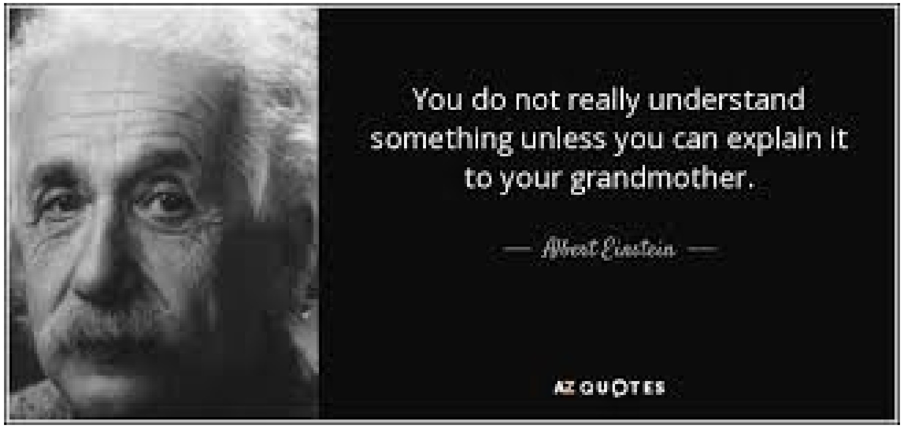
# Analyzing Insertion Sort

$$\boxed{T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5\sum_{j=2}^{n}t_j+c_6\sum_{j=2}^{n}(t_j-1)+c_7\sum_{j=2}^{n}(t_j-1)+c_8(n-1)}$$

- What can T(n) be?
  - <u>Best</u> case -- inner loop body never executed
    - $t_i = 1$ ➔ T(n) is a linear function
  - <u>Worst</u> case -- inner loop body executed for all previous elements
    - $t_i = i$ ➔ T(n) is a quadratic function
  - <u>Average</u> case
    - ???

# So, Is Insertion Sort Good?

- Criteria for selecting algorithms
  - Correctness
  - Amount of work done
  - Amount of space used
  - Simplicity, clarity, maintainability
  - Optimality



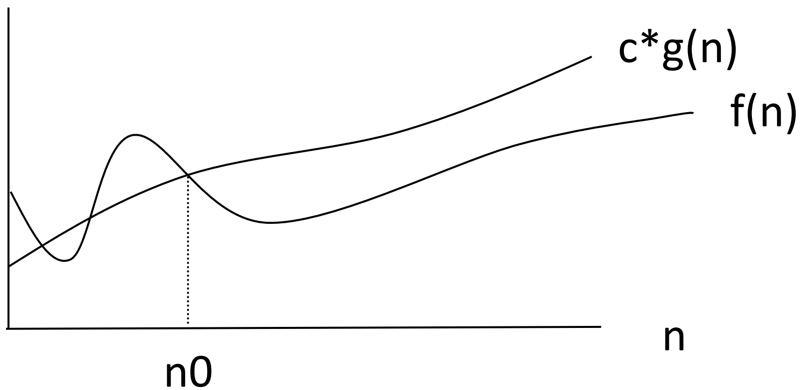
# Asymptotic Notation

- We will study the asymptotic efficiency of algorithms
  - To do so, we look at input sizes large enough to make only the order of growth of the running time relevant
  - That is, we are concerned with how the running time of an algorithm increases with the size of the input in the limit as the size of the input increases without bound.
  - Usually an algorithm that is asymptotically more efficient will be the best choice for all but very small inputs.
    - Real-time systems, games, interactive applications need to limit the input size to sustain their performance.
- 3 asymptotic notations
  - Big O, $\Theta$, $\Omega$ Notations

# Big-Oh Notation: Asymptotic Upper Bound

Want g(n) to be simple.

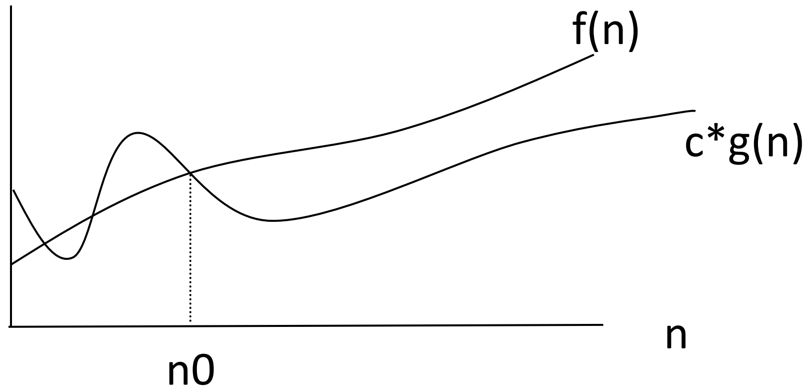
- **T(n) = f(n) = O(g(n))**
  - if f(n) <= c*g(n) for all n > n0, where c & n0 are constants > 0




- Example: T(n) = 2n + 5 is O(n). Why?
  - 2n+5 <= 3n, for all n >= 5
- T(n) = $5*n^2 + 3*n + 15$ is $O(n^2)$. Why?
  - $5*n^2 + 3*n + 15 <= 6*n^2$, for all n >= 6

# $\Omega$ Notation: Asymptotic Lower Bound

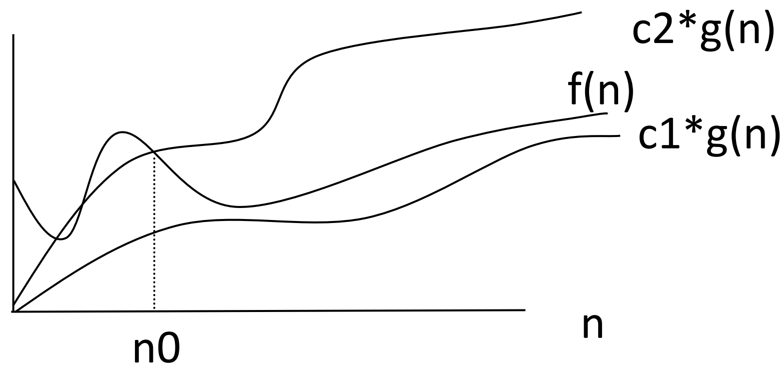
- **T(n) = f(n) = $\Omega$(g(n))**
  - if f(n) >= c*g(n) for all n > n0, where c and n0 are constants > 0




- Example: T(n) = 2n + 5 is $\Omega$(n). Why?
  - 2n+5 >= 2n, for all n > 0
- T(n) = $5*n^2 - 3*n$ is $\Omega(n^2)$. Why?
  - $5*n^2 - 3*n >= 4*n^2$, for all n >= 4

# $\Theta$ Notation: Asymptotic Tight Bound

- $T(n) = f(n) = \Theta(g(n))$
  - if c1*g(n) <= f(n) <= c2*g(n) for all n > n0, where c1, c2 and n0 are constants > 0




- Example: $T(n) = 2n + 5$ is $\Theta(n)$. Why?

  2n <= 2n+5 <= 3n, for all n >= 5

- $T(n) = 5*n^2 - 3*n$ is $\Theta(n^2)$. Why?
  - $4*n^2 <= 5*n^2 - 3*n <= 5*n^2$, for all n >= 4

# Big-Oh, Theta, Omega

Tips to guide your intuition:

- Think of $O(g(N))$ as "greater than or equal to" $f(N)$
  - Upper bound
- Think of $\Omega(g(N))$ as "less than or equal to" $f(N)$
  - Lower bound
- Think of $\Theta(g(N))$ as "equal to" $f(N)$
  - "Tight" bound

(*True for large N and ignoring constant factors*)

$$f(n) = O(g(n)) \Rightarrow f \preceq g$$

$$f(n) = \Omega(g(n)) \Rightarrow f \succeq g$$

$$f(n) = \Theta(g(n)) \Rightarrow f \approx g$$

# Common Functions

| Name | Big-Oh | Comment |
|---|---|---|
| Constant | $O(1)$ | Can't beat it! |
| Log log | $O(\log\log N)$ | Extrapolation search |
| Logarithmic | $O(\log N)$ | Typical time for good searching algorithms |
| Linear | $O(N)$ | This is about the fastest that an algorithm can run given that we need $O(n)$ just to read the input |
| N logN | $O(N\log N)$ | Most sorting algorithms |
| Quadratic | $O(N^2)$ | Acceptable when the data size is small ($N<10000$) |
| Cubic | $O(N^3)$ | Acceptable when the data size is small ($N<1000$) |
| Exponential | $O(2^N)$ | Only good for really small input sizes ($n<=20$) |

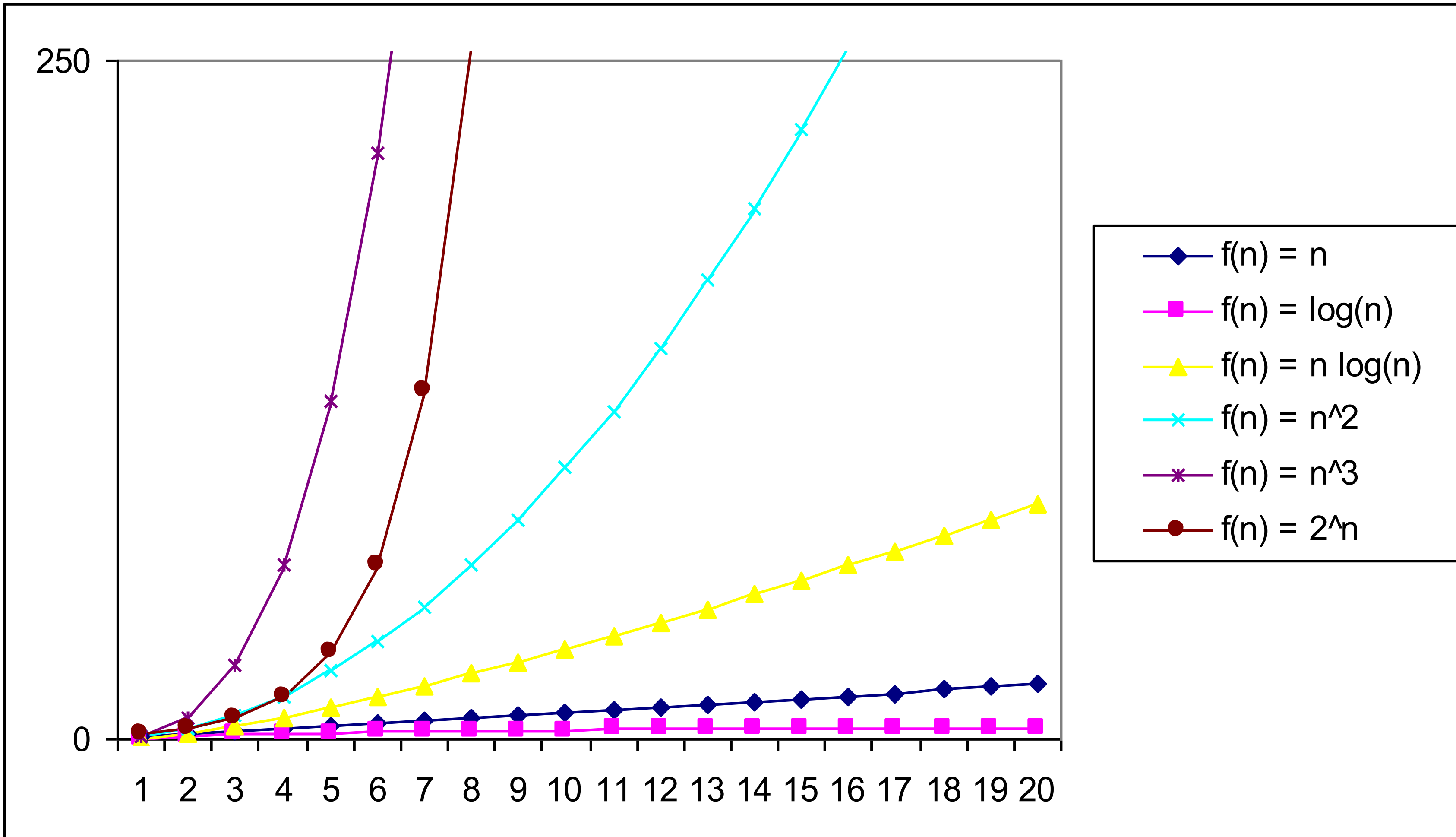
Increasing cost

Polynomial time

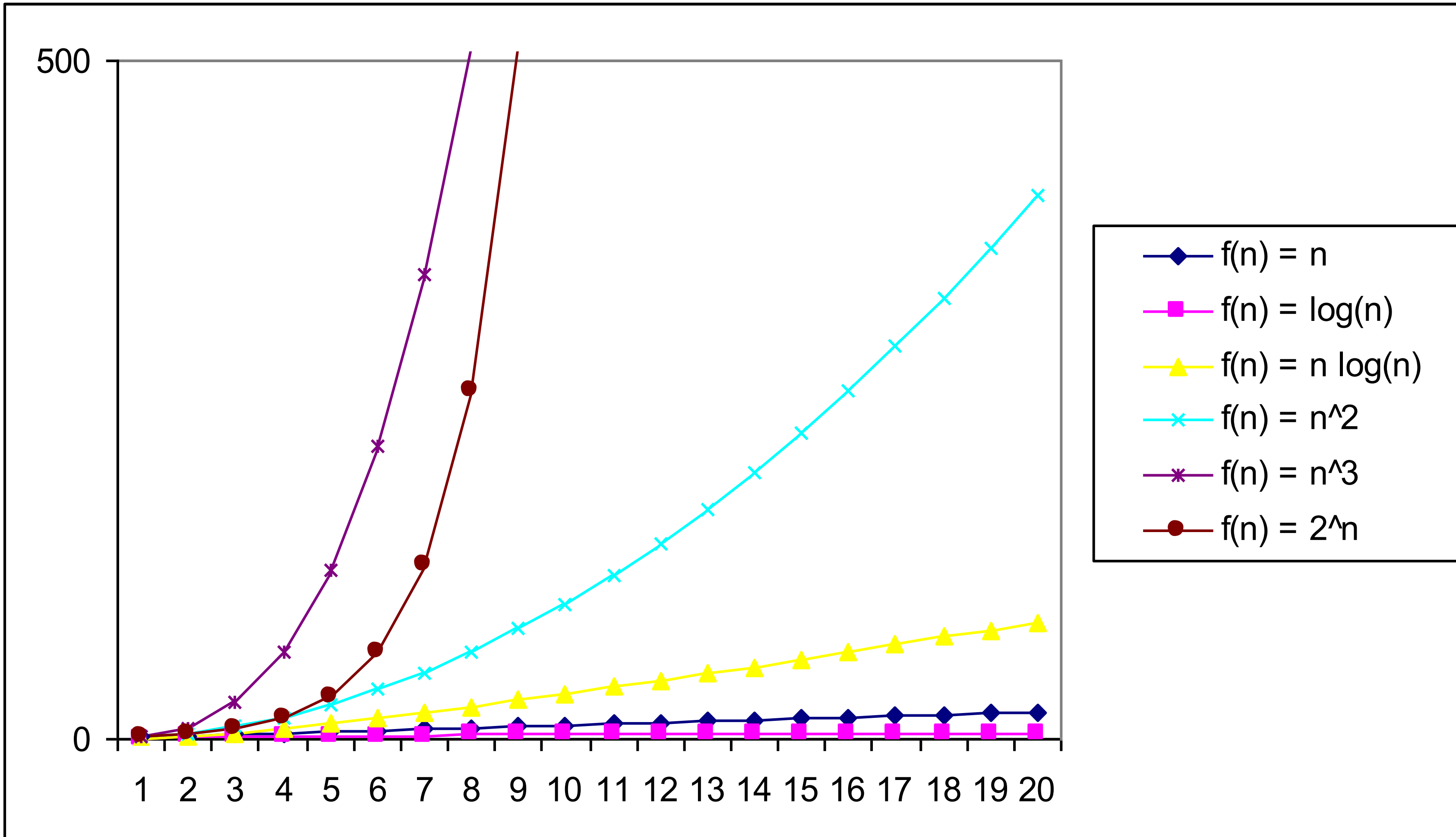
# Comparison of growth rates

| $\log n$ | $n$ | $n \log n$ | $n^2$ | $n^3$ | $2^n$ |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 2 |
| 0.6931 | 2 | 1.39 | 4 | 8 | 4 |
| 1.099 | 3 | 3.30 | 9 | 27 | 8 |
| 1.386 | 4 | 5.55 | 16 | 64 | 16 |
| 1.609 | 5 | 8.05 | 25 | 125 | 32 |
| 1.792 | 6 | 10.75 | 36 | 216 | 64 |
| 1.946 | 7 | 13.62 | 49 | 343 | 128 |
| 2.079 | 8 | 16.64 | 64 | 512 | 256 |
| 2.197 | 9 | 19.78 | 81 | 729 | 512 |
| 2.303 | 10 | 23.03 | 100 | 1000 | 1024 |
| 2.398 | 11 | 26.38 | 121 | 1331 | 2048 |
| 2.485 | 12 | 29.82 | 144 | 1728 | 4096 |
| 2.565 | 13 | 33.34 | 169 | 2197 | 8192 |
| 2.639 | 14 | 36.95 | 196 | 2744 | 16384 |
| 2.708 | 15 | 40.62 | 225 | 3375 | 32768 |
| 2.773 | 16 | 44.36 | 256 | 4096 | 65536 |
| 2.833 | 17 | 48.16 | 289 | 4913 | 131072 |
| 2.890 | 18 | 52.03 | 324 | 5832 | 262144 |

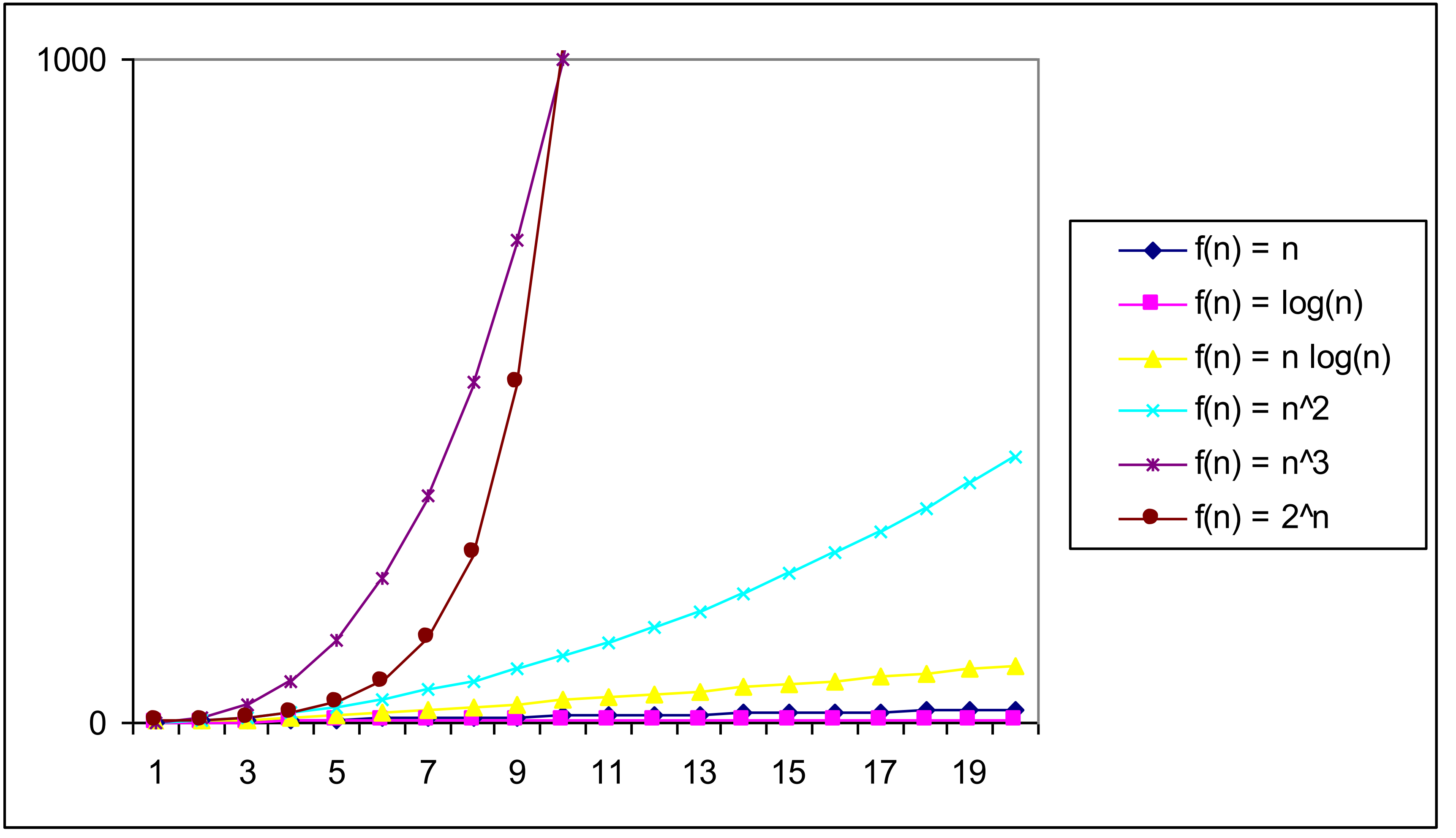
# Asymptotic Complexity



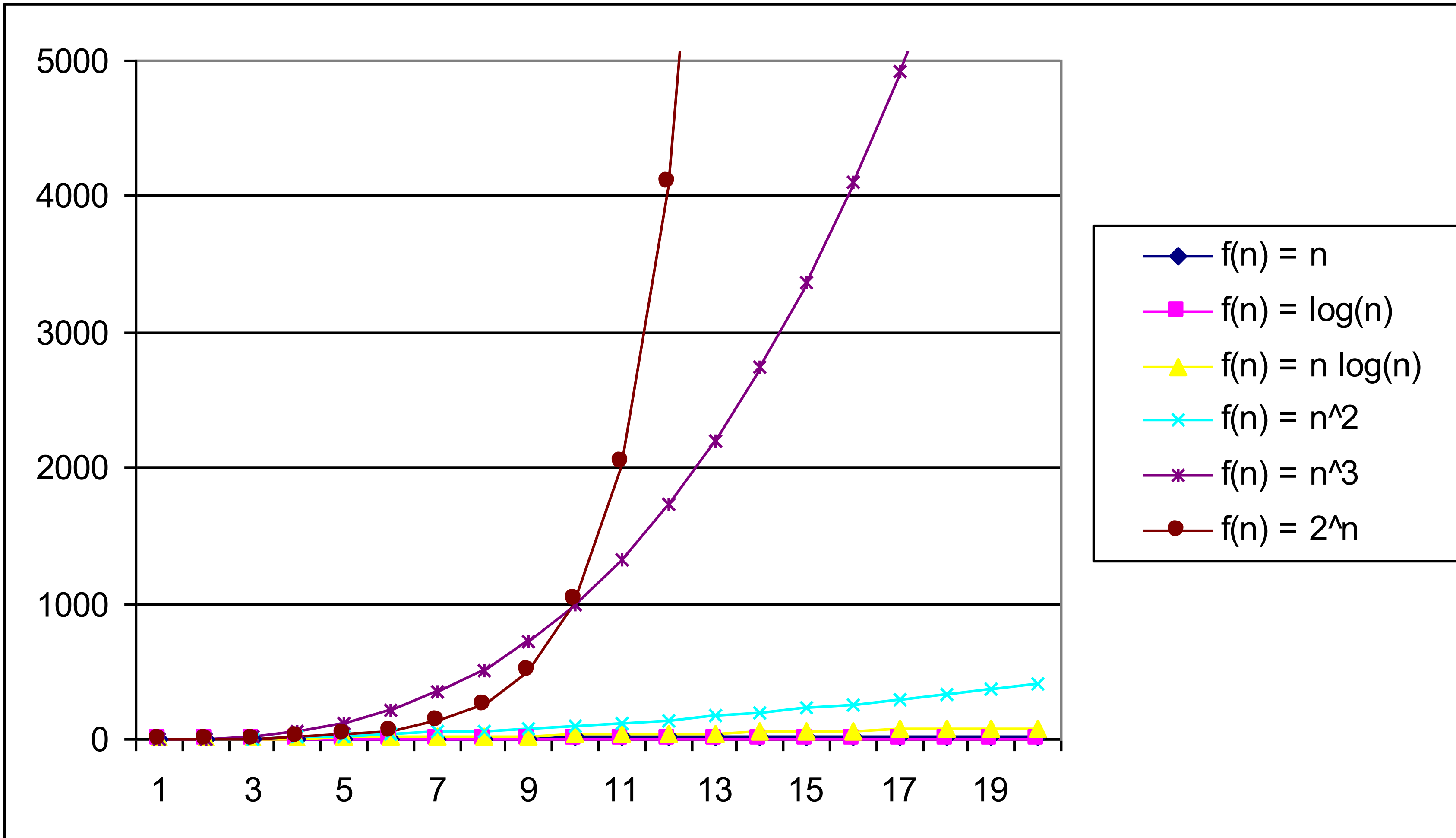
# Asymptotic Complexity



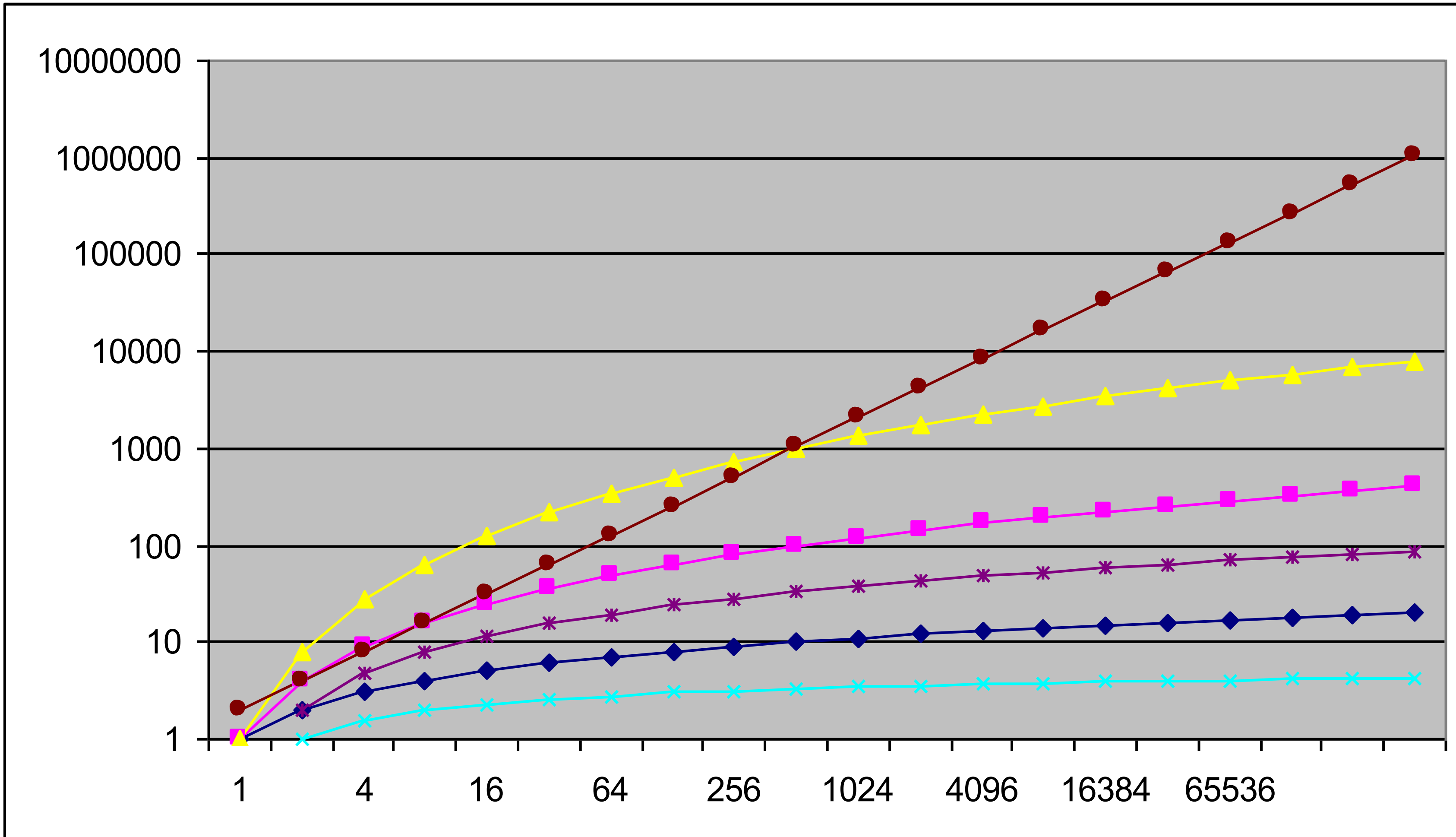
# Asymptotic Complexity



# Asymptotic Complexity



# Asymptotic Complexity



# Quiz Time: Deliver Today

- Make the group of 2
- Deliver today, before 3pm
- No need to knock the door, Just under the door

- If this is too hard for you??, we could postpone the deadline until the Monday session!!!

# Quiz 2

**Show that** $\frac{1}{2}n^2 + 3n = \Theta(n^2)$

**Show that** $(n \log n - 2\,n + 13) = \Omega(n \log n)$

# Quiz 3

- Discuss the complexity of the following algorithm:

```
void sum_first_n(int n) {
    int i,sum=0;
    for (i=1;i<=n;i++)
        sum = sum + i;
  }
```

# Quiz 4

- Discuss the complexity of the following algorithm:

```
int binarysearch(int a[], int n, int val)
{
int l=1, r=n, m;
   while (r>=1) {
     m = (l+r)/2;
     if   (a[m]==val) return m;
     if   (a[m]>val) r=m-1;
     else l=m+1; }
return -1;
}
```

# Quiz 5

- Discuss the complexity of the following algorithm:

```
int *compute_sums(int A[], int n) {
    int M[n][n];
    int i,j;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            M[i][j]=A[i]+A[j];
    return M;
    }
```

# Math Review

- S(N) = 1 + 2 + 3 + 4 + … N =

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

- Sum of Squares:

$$\sum_{i=1}^{N} i^2 = \frac{N*(N+1)*(2n+1)}{6} \approx \frac{N^3}{3}$$

- Geometric Series:

$$\sum_{i=0}^{N} A^i = \frac{1-A^{N+1}}{1-A} = \Theta(1)$$

A < 1

$$\sum_{i=0}^{N} A^i = \frac{A^{N+1}-1}{A-1}$$

A > 1

# Math Review

- Linear Geometric Series:

$$\sum_{i=0}^{n} ix^{i} = x + 2x^{2} + 3x^{3} + \ldots + nx^{n} = \frac{(n-1)x^{(n+1)} - nx^{n} + x}{(x-1)^{2}}$$

# Math Review

- Logarithms:

$$\log A^B = B * \log A$$

$$\log(A * B) = \log A + \log B$$

$$\log(\frac{A}{B}) = \log A - \log B$$

# Math Review

- Summations with general bounds:

$$\sum_{i=a}^{b} f(i) = \sum_{i=0}^{b} f(i) - \sum_{i=0}^{a-1} f(i)$$

- Linearity of Summations:

$$\sum_{i=1}^{n} (4i^2 - 6i) = 4\sum_{i=1}^{n} i^2 - 6\sum_{i=1}^{n} i$$

# PROOF TECHNIQUES

- Proof by induction

- Proof by contradiction

- Proof by construction

# Induction

We have statements $P_1, P_2, P_3, \ldots$

If we know

- for some k that $P_1, P_2, \ldots, P_k$ are true
- for any $n >= k$ that

  $P_1, P_2, \ldots, P_n$ imply $P_{n+1}$

Then

Every $P_i$ is true

# Proof by Induction

- Inductive basis

  Find $P_1, P_2, \ldots, P_k$ which are true

- Inductive hypothesis

  Let's assume $P_1, P_2, \ldots, P_n$ are true,

  for any $n >= k$

- Inductive step

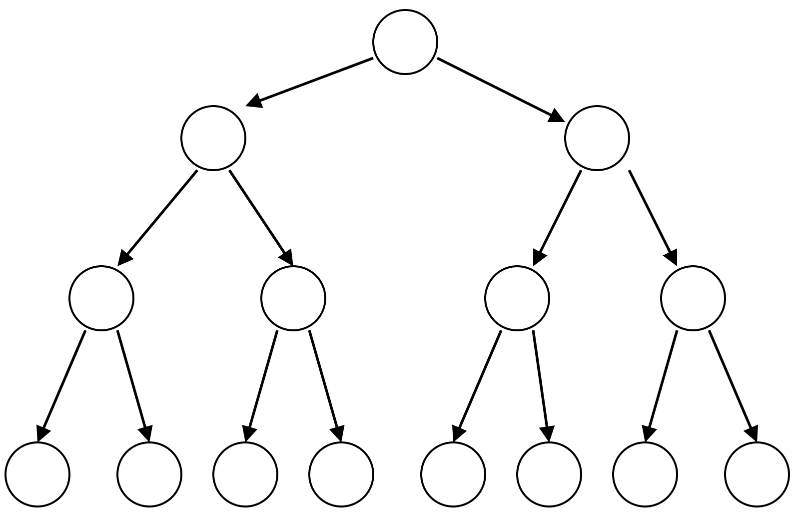
  Show that $P_{n+1}$ is true

# Example

**Theorem:** A binary tree of height $n$ has at most $2^n$ leaves.

**Proof:**

let $l(i)$ be the number of leaves at level $i$



$l(0) = 1$

$l(3) = 8$

We want to show: $l(i) <= 2^i$

- Inductive basis

  $l(0) = 1$ (the root node)

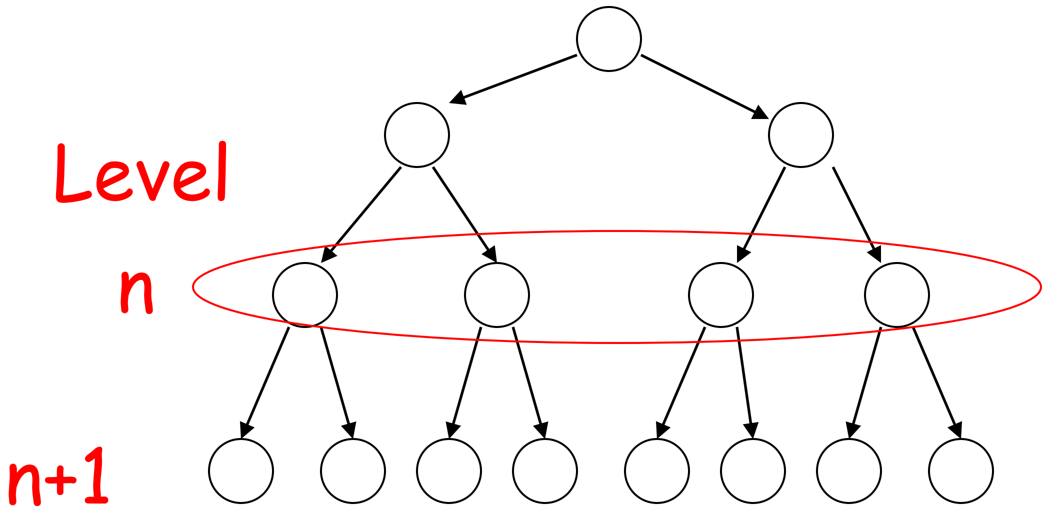
- Inductive hypothesis

  Let's assume $l(i) <= 2^i$ for all $i = 0, 1, \ldots, n$

- Induction step

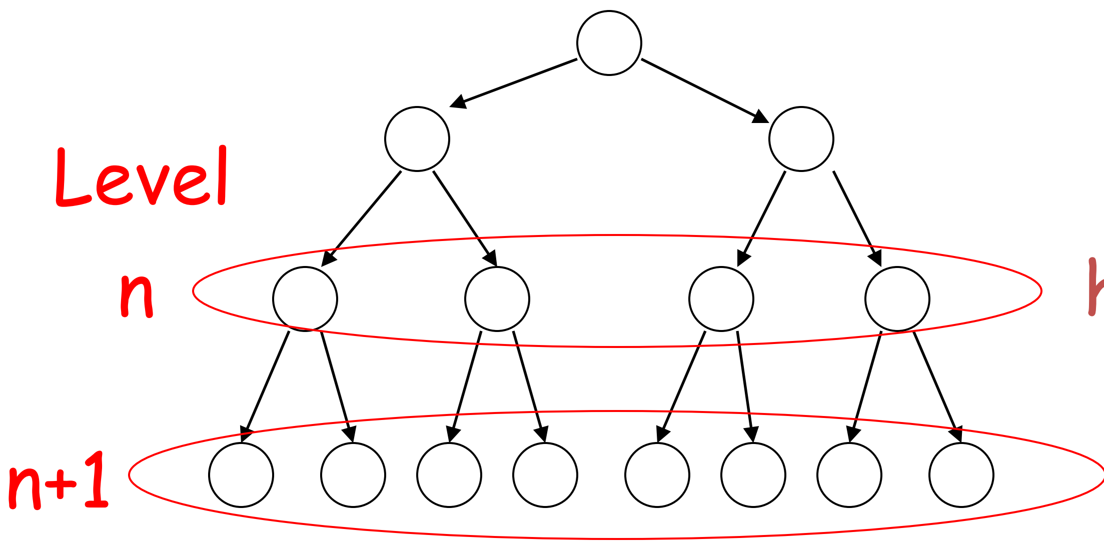
  we need to show that $l(n + 1) <= 2^{n+1}$

# Induction Step




hypothesis: $l(n) <= 2^n$

# Induction Step




hypothesis: $l(n) <= 2^n$

$$l(n+1) <= 2 * l(n) <= 2 * 2^n = 2^{n+1}$$

# Remark

Recursion is another thing

Example of recursive function:

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

$$f(0) = 1, \quad f(1) = 1$$

# Proof by Contradiction

We want to prove that a statement P is true

- we assume that P is false
- then we arrive at an incorrect conclusion
- therefore, statement P must be true

# Example

Theorem: $\sqrt{2}$ is not rational

Proof:

Assume by contradiction that it is rational

$\sqrt{2}$ = n/m

n and m have no common factors

We will show that this is impossible

$\sqrt{2} = n/m \Rightarrow 2m^2 = n^2$

Therefore, $n^2$ is even $\Rightarrow$ n is even

$n = 2k$

$2m^2 = 4k^2 \Rightarrow m^2 = 2k^2 \Rightarrow$ m is even

$m = 2p$

Thus, m and n have common factor 2

**Contradiction!**

# Proof by Construction

Many theorems state that a particular type of object exists. One way to prove such a theorem is by demonstrating how to construct the object.

This technique is a ***proof by construction***.

# Example

For each even number $n$ greater than 2, there exists a 3-regular graph with $n$ nodes.

**PROOF** Let $n$ be an even number greater than 2. Construct graph $G = (V, E)$ with $n$ nodes as follows. The set of nodes of $G$ is $V = \{0, 1, \ldots, n-1\}$, and the set of edges of $G$ is the set

$$\begin{aligned} E = \{\, \{i,\ i+1\} \mid \text{ for } 0 \leq i \leq n-2\} \cup \{\, \{n-1,\ 0\} \,\} \\ \cup \,\{\, \{i,\ i+n/2\} \mid \text{ for } 0 \leq i \leq n/2-1\}. \end{aligned}$$

Picture the nodes of this graph written consecutively around the circumference of a circle. In that case, the edges described in the top line of $E$ go between adjacent pairs around the circle. The edges described in the bottom line of $E$ go between nodes on opposite sides of the circle. This mental picture clearly shows that every node in $G$ has degree 3.

# Quiz

Let $T(1), T(2), T(3), \ldots, T(n)$ be a sequence of numbers such that for all $n \geq 2$ then

$$T(n) = 2T(n-1) + 2^{n+1}$$

If $T(1) = 4$ then prove that for all $n \geq 1$,

$$T(n) = n2^{n+1}$$