CS 171: Introduction to Computer Science II

Algorithm Analysis

Li Xiong

Today

- Hw1 discussion
- Recap: linear search and binary search
- Algorithm Analysis
- Big-O Notation
- Loop Analysis

Hw1 Discussion

- Read the instructions carefully
- Think before you code
- Useful classes/methods
  - ArrayList
  - Random Number generation

ArrayList

- Use generics - parameterized types
  - Type parameters have to be instantiated as reference types
- Autoboxing
  - Autoboxing: Automatically casting a primitive type to a wrapper type
  - Auto-unboxing: automatically casting a wrapper type to a primitive type

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(1001);
int mynumber = numbers.remove(0);
```

ArrayList

- Useful methods
  - add(E e): Appends the specified element to the end of this list
  - size(): returns the number of elements in this list
  - remove(int index): Removes the element at the specified position in this list. Shifts any subsequent elements to the left (subtracts one from their indices)
  - get(int index): Returns the element at the specified position in this list.

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(1001);
int n = numbers.size();
int mynumber2 = numbers.get(0);
int mynumber = numbers.remove(0);
```

Random number generation

- If you want to generate random test numbers

  Math.random() method:

```
double x = Math.random();
```

  This generates a double between [0.0, 1.0].

Today

- Hw1 discussion
- Recap: linear search and binary search
- Algorithm Analysis
- Big-O Notation
- Loop Analysis

Search in an Array

- Unordered array: ~N
- Order array: ~lgN

Hunts Needle in a Haystack

HOW LONG does it take to find a needle in a haystack? Jim Moran, Washington, D. C., publicity man, recently dropped a needle into a convenient pile of hay, hopped in after it, and began an intensive search for (a) some publicity and (b) the needle. Having found the former, Moran abandoned the needle hunt.

Review question 1

- The maximum number of elements to examine to complete binary search of 30 elements is:
  - A: 1
  - B: 30
  - C: 7
  - D: 5

Review Question 2

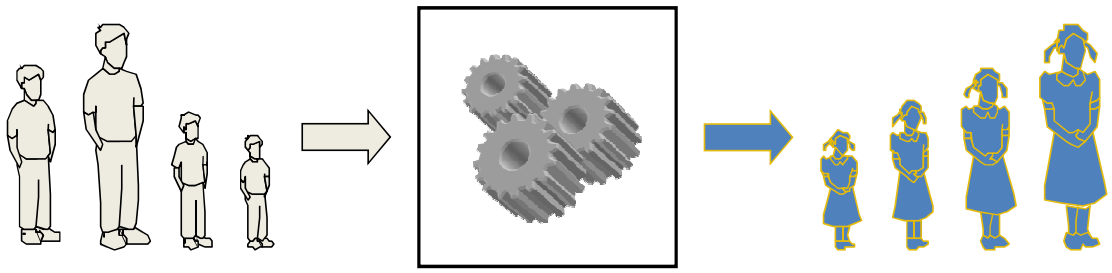
- True or false: It is generally faster to find an **existing** item in an ordered array than a **missing** one (item not there).
- Trust or false: It is generally faster to search an item in an **ordered** array than in an **unordered** array of the same size

Algorithm Analysis

- An algorithm is a method for solving a problem expressed as a sequence of steps that is suitable for execution by a computer (machine)
  - E.g. Search, insertion, deletion in an array
- We are interested in designing good algorithms
  - Linear search vs. binary search
- Good algorithms
  - Running time
  - Space usage (amount of memory required)

Running time of an algorithm

- Running time typically increases with the input size (problem size)
- Also affected by hardware and software environment
- We would like to focus on the relationship between the running time and the input size

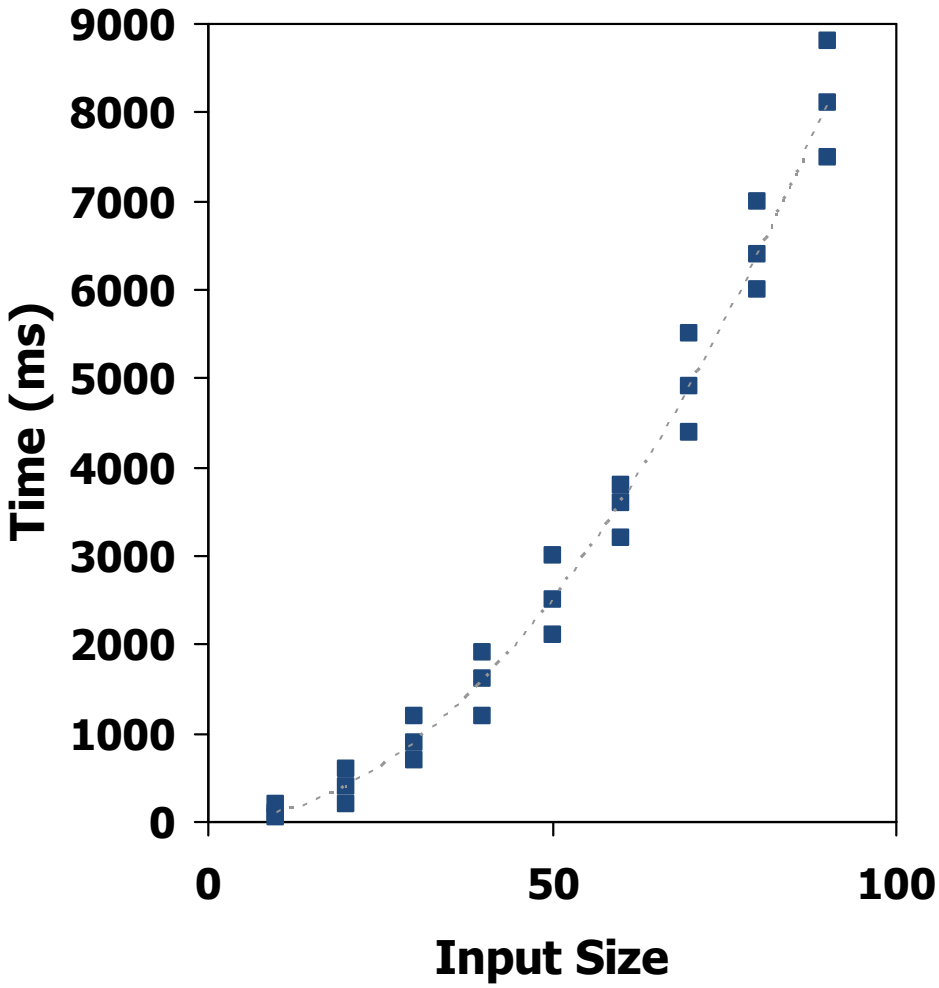


How to measure running time

- Experimental studies
- Theoretical analysis

Experimental Studies

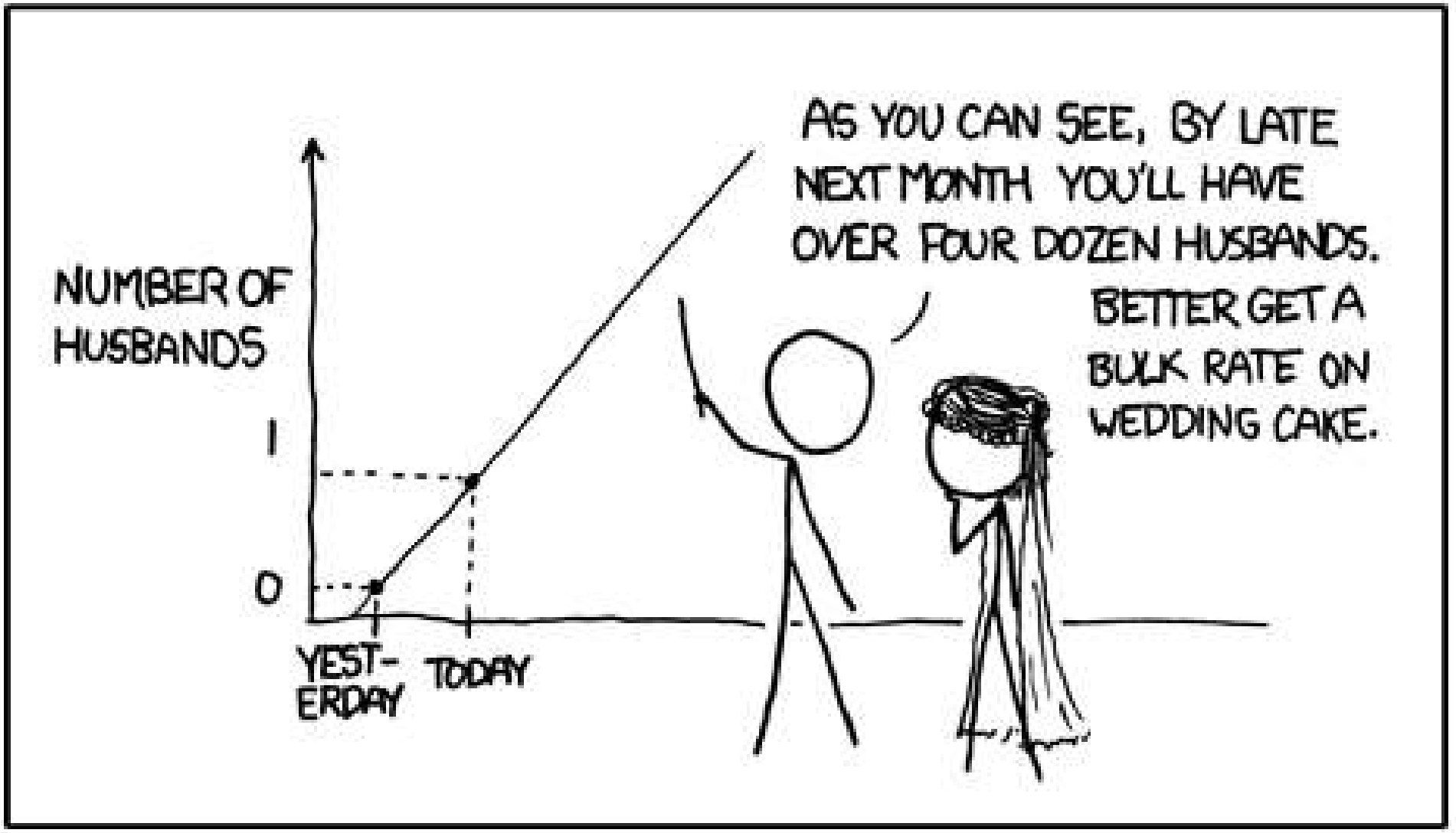
- Write a program implementing the algorithm
- Run the program with inputs of varying size and composition
- Use a method like System.currentTimeMillis() to get an accurate measure of the actual running time
- Plot the results



Limitations of Experiments

- It is necessary to implement the algorithm, which may be difficult
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used

MY HOBBY: EXTRAPOLATING



Mathematical Analysis - insight

- Total running time of a program is determined by two primary factors:
  - Cost of executing each statement (property of computer, Java compiler, OS)
  - Frequency of execution of each statement (property of program and input)

Algorithm Analysis

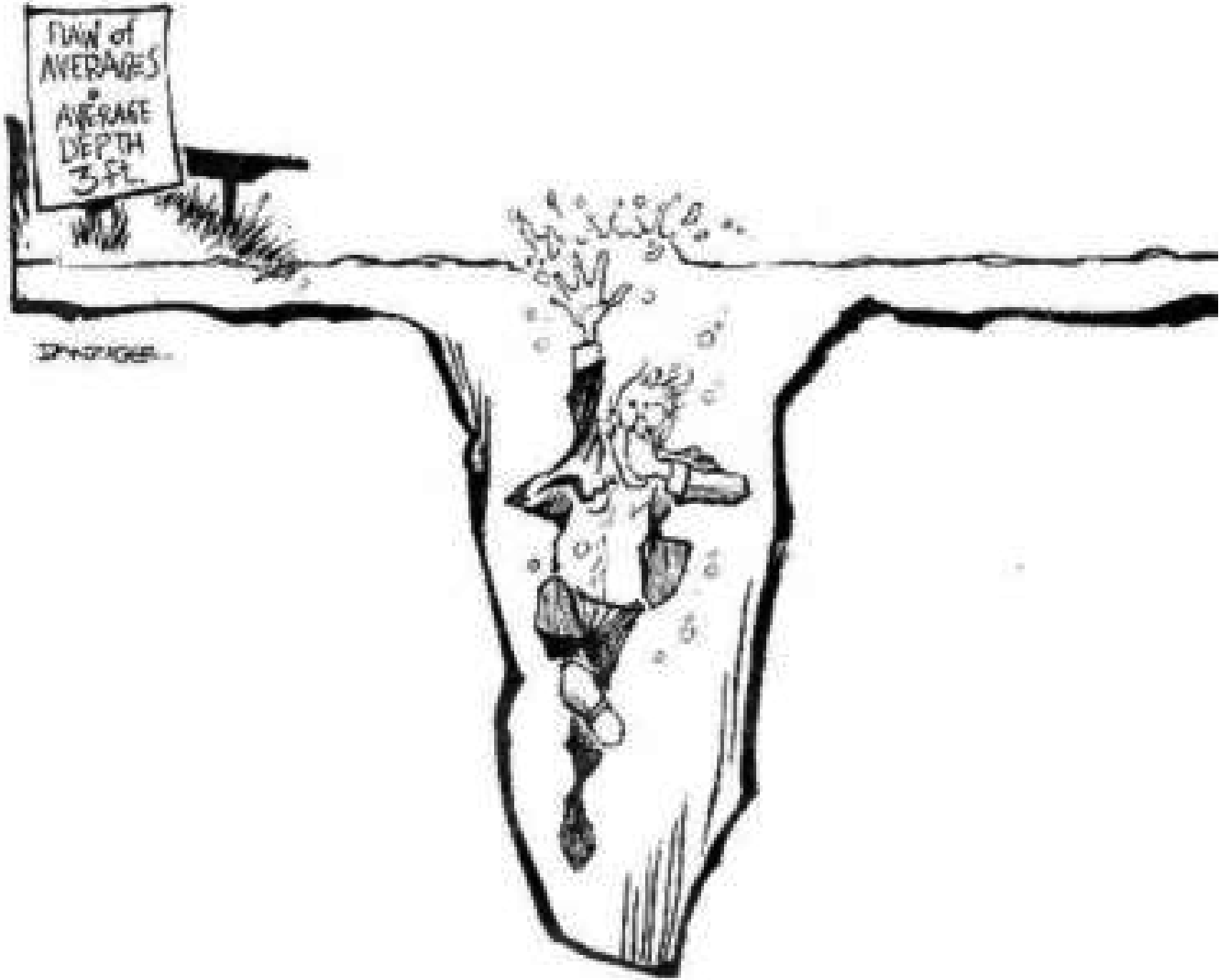
- Algorithm analysis:
  - Determine frequency of execution of statements
  - Characterizes running time as a function of the input size
- Benefit:
  - Takes into account all possible inputs
  - Allows us to evaluate the speed of an algorithm independent of the hardware/software environment

Analysis Method

- Count the number of primitive operations executed as a function of input size
- A primitive operation corresponds to a low-level (basic) computation with a constant execution time
  - Evaluating an expression
  - Assigning a value to a variable
  - Indexing into an array
- The number of primitive operations is a good estimate that is proportional to the running time of an algorithm

Average-case vs. worst-case

- An algorithm may run faster on some inputs than it does on others (with the same input size)
- Average case: taking the average over all possible inputs of the same size
  - Depends on input distribution
- Best case
- Worst case
  - Easier analysis
  - Typically leads to better algorithms


TOWN of
AVERAGES
AVERAGE
DEPTH
3 FT.

Loop Analysis

- Programs typically use loops to enumerate through input data items
- Count number of operations or steps in loops
- Each statement within the loop is counted as a step

Example 1

```
double sum = 0.0;
for (int i = 0; i < n; i ++) {
    sum += array[i];
}
```

How many steps?

Only count the loop statements (update to the loop variable i is ignored).

Example 1: Solution

```
double sum = 0.0;
for (int i = 0; i < n; i ++) {
    sum += array[i];
}
```

How many steps?

Loop will be executed n times; and there is 1 loop statement. So overall:

n

Example 2

```
double sum = 0.0;
for (int i = 0; i < n; i += 2) {
    sum += array[i];
}
```

How many steps?

Example 2: Solution

```
double sum = 0.0;
for (int i = 0; i < n; i += 2) {
    sum += array[i];
}
```

How many steps?

Loop will be executed n/2 times. So overall:

$n/2$

Example 3 – Multiple Loops

```
for (int i = 0; i < n; i ++) {
    for (int j = 0; j < n; j ++) {
        int x = i*j;
        sum += x;
    }
}
```

How many steps?

Example 3 – Solution

```
for (int i = 0; i < n; i ++) {
    for (int j = 0; j < n; j ++) {
        int x = i*j;
        sum += x;
    }
}
```

How many steps?

2 loops, each loop *n* times, so overall:

$$2\,n^2$$

Increase of Cost w.r.t. n

- Example 1 takes twice as many steps (n) as Example 2 (n/2), but both of them are **linear** to the input size n
  - If n is 3 times larger, both costs are 3 times larger
- Example 3 $(2n^2)$is different:
  - If n is 3 times larger, it becomes 9 times more expensive.
  - Therefore the cost is **quadratic** w.r.t. to problem size.

Increase of Cost with Growth of n

- In practice we care a lot about how the cost **increases** w.r.t. the problem size, rather than the **absolute** cost.
- Therefore we can ignore the constant scale factor in the cost function, and concentrate on the part relevant to n
- We need formal mathematical definitions and tools for comparing the cost

Tilde Notation

- Tilde notation: ignore insignificant terms
- Definition: we write $f(n) \sim g(n)$ if $f(n)/g(n)$ approaches 1 as n grows

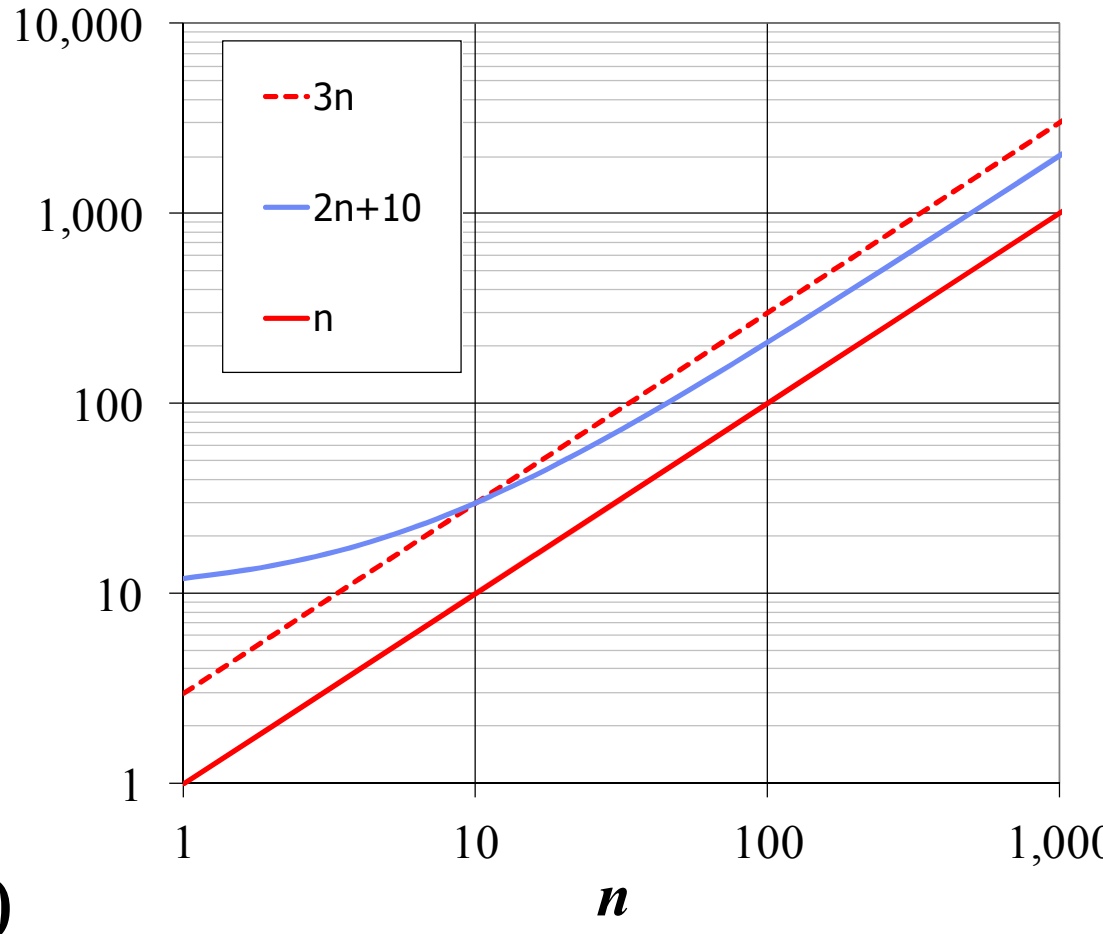
- $2\boldsymbol{n} + 10 \sim 2n$
- $3n^3 + 20n^2 + 5 \sim 3n^3$

Big-Oh Notation

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

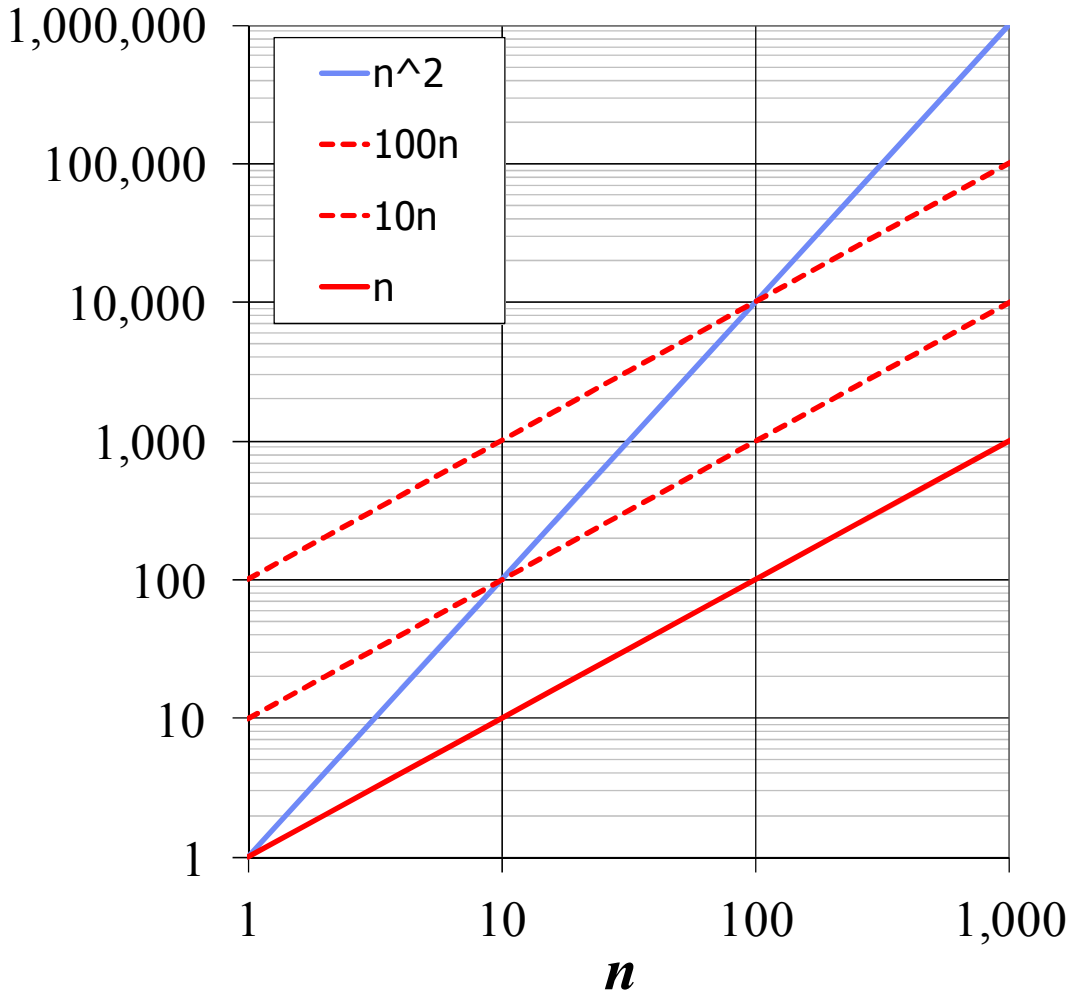
  $f(n) \leq cg(n)$ for $n \geq n_0$

- Example: $2n + 10$ is $O(n)$
  - pick $c = 3$ and $n_0 = 10$



Big-Oh Example

- Example: the function n^2 is not $O(n)$
  - $n^2 \leq cn$
  - $n \leq c$
  - The above inequality cannot be satisfied since c must be a constant

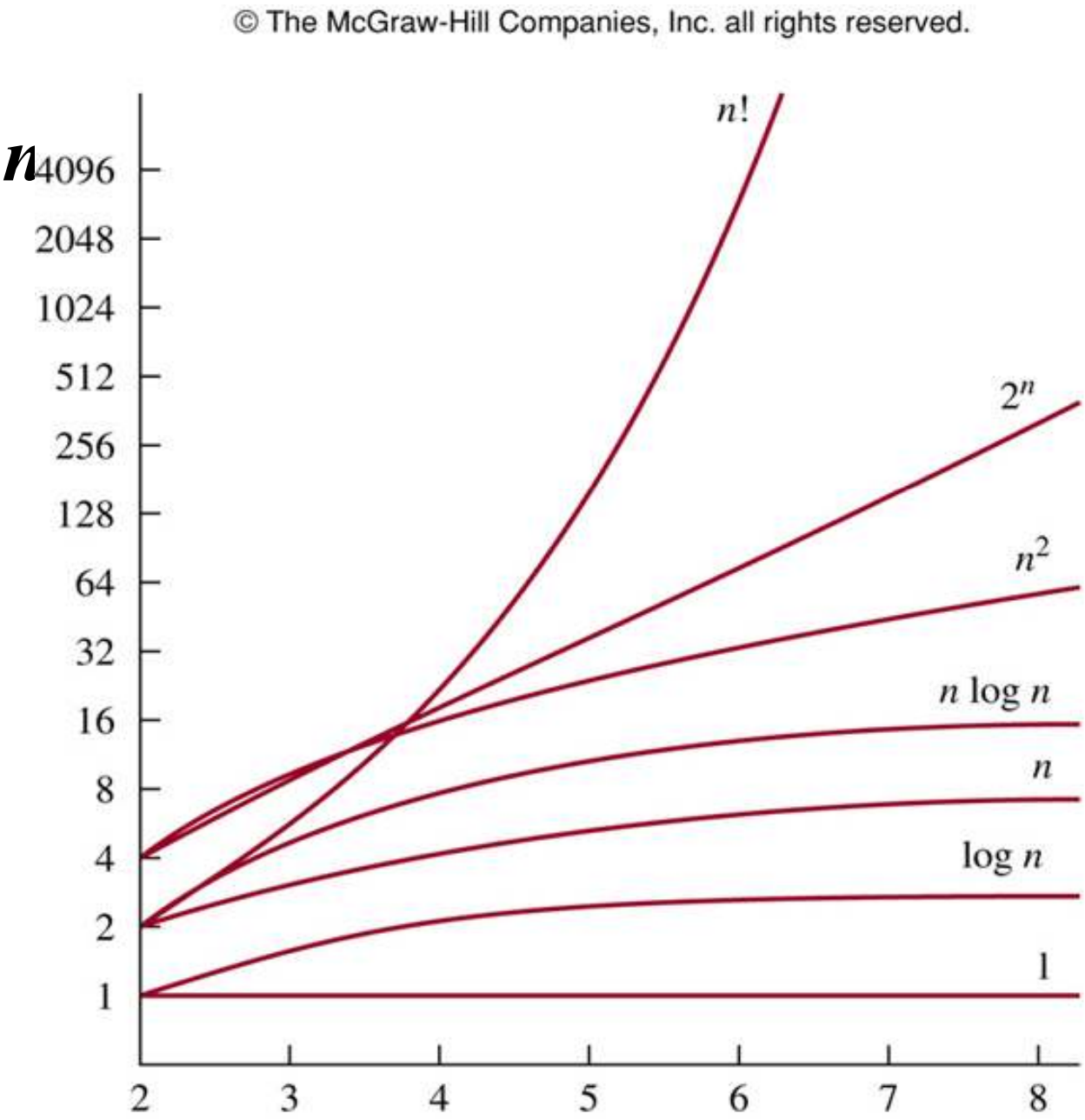


Big-Oh and Growth Rate

- The big-Oh notation gives an **upper bound** on the growth rate of a function
- The statement "$f(n)$ is $O(g(n))$" means that the growth rate of $f(n)$ is no more than the growth rate of $g(n)$
- We can use the big-Oh notation to rank functions according to their growth rate

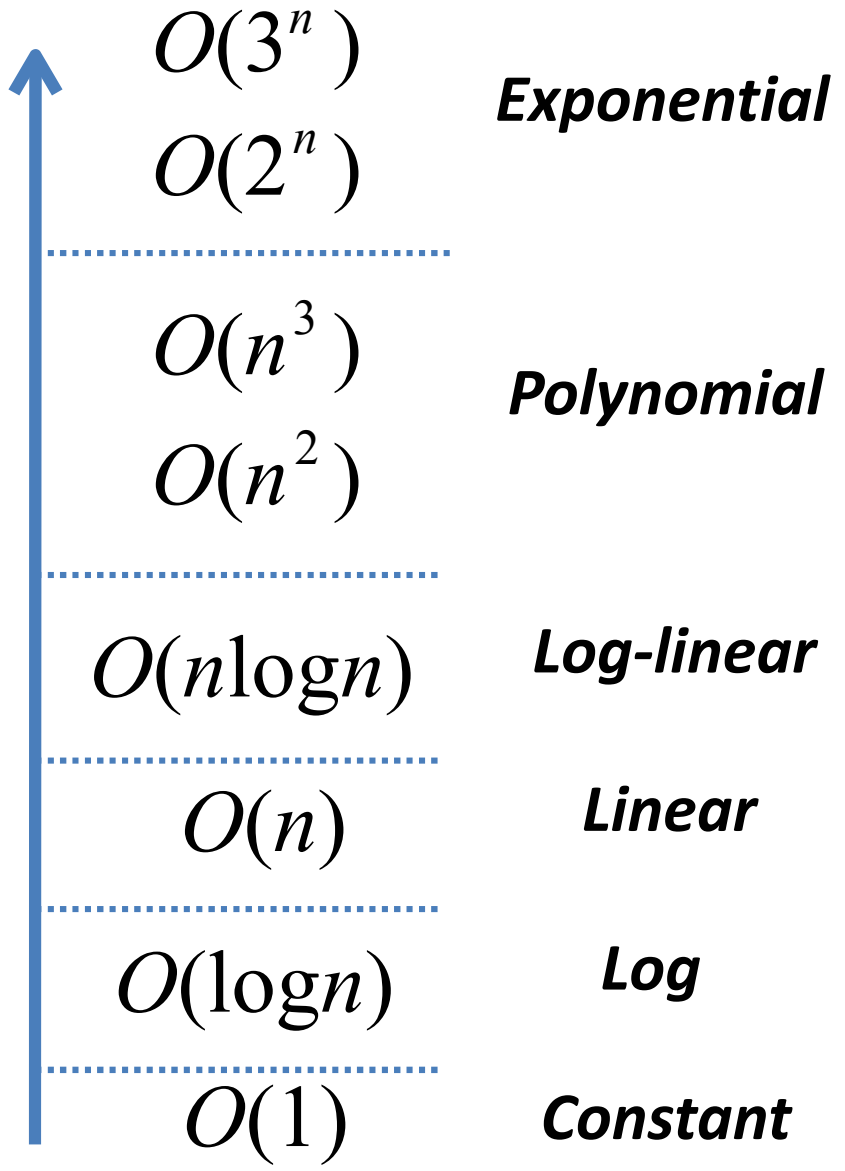
Important Functions in Big-Oh Analysis

- Constant: 1
- Logarithmic: log n
- Linear: n
- N-Log-N: n log n
- Quadratic: n^2
- Cubic: n^3
- Polynomial: n^d
- Exponential: 2^n
- Factorial: $n!$



Growth Rate

- From calculus we know that: in terms of the **order**: *exponentials > polynomials > logarithms > constant.*



Big-Oh Analysis

- Write down cost function $f(n)$
  1. Look for highest-order term
  2. Drop constant factors

- Examples
  - $3n^3 + 20n^2 + 5$
  - $n \log n + 10$

Example 4

```
for (int i = 0; i < n; i ++) {
    for (int j = i; j < n; j ++) {
        sum += i*j;
    }
}
```

Example 4: Solution

```
for (int i = 0; i < n; i ++) {
    for (int j = i; j < n; j ++) {
        sum += i*j;
    }
}
```

$$n+(n-1)+(n-2)+\ldots+1+0 = \frac{n(n+1)}{2}$$ is

0.5 (n^2 + n) → O(n^2)

Example 5

```
double product = 1.0;
for (int i = 1; i <= n; i *= 2) {
   product *= i;
}
```

Example 5: Solution

```
double product = 1.0;
for (int i = 1; i <= n; i *= 2) {
    product *= i;
}
```

- This has a logarithmic cost:

  $O(\log_2 n)$

  or $O(\log n)$ as the change of base is merely a matter of a constant factor.

Search in Ordered vs. Unordered Array

- What's the big O function for linear search?
- Binary search?

Search in Ordered vs. Unordered Array

- What's the big O function for linear search?

O(N)

- Binary search? O(lgN)
- Binary search has much better running time, particularly for large-scale problems

Review Question

- What is the Order of growth (big-oh) of the following code?

```
for (int i=1; i<=N; ++i){
    for (int j=1; j<N; j*=2){
        count++;
    }
}
```

Summary

- Today
  - –Algorithm Analysis
  - –Loop Analysis

- Next lecture
  - –Simple sorting algorithms and analysis