CS 171: Introduction to Computer Science II

Algorithm Analysis + Simple Sorting

Li Xiong

Today

- Algorithm Analysis (cont)
- Simple sorting algorithms

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

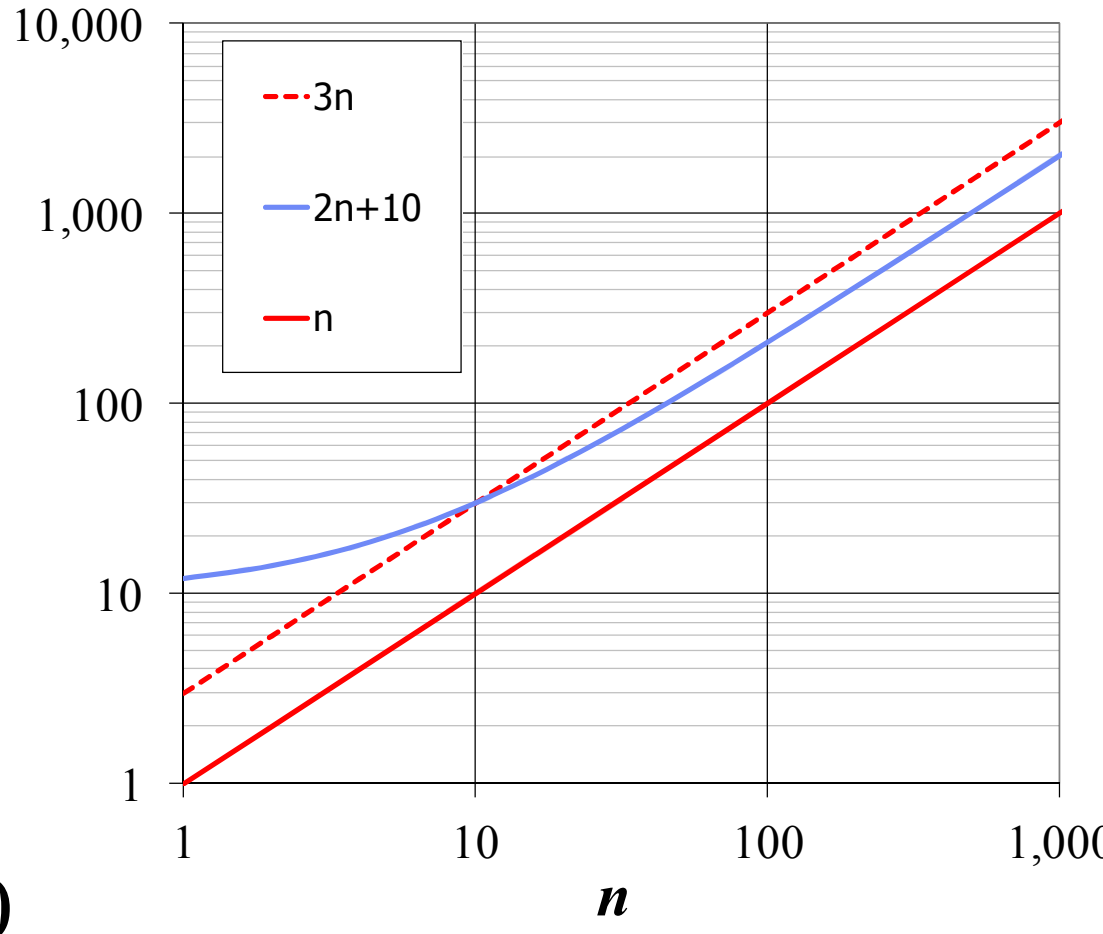
- $2\boldsymbol{n} + 10 \sim 2n$
- $3n^3 + 20n^2 + 5 \sim 3n^3$

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

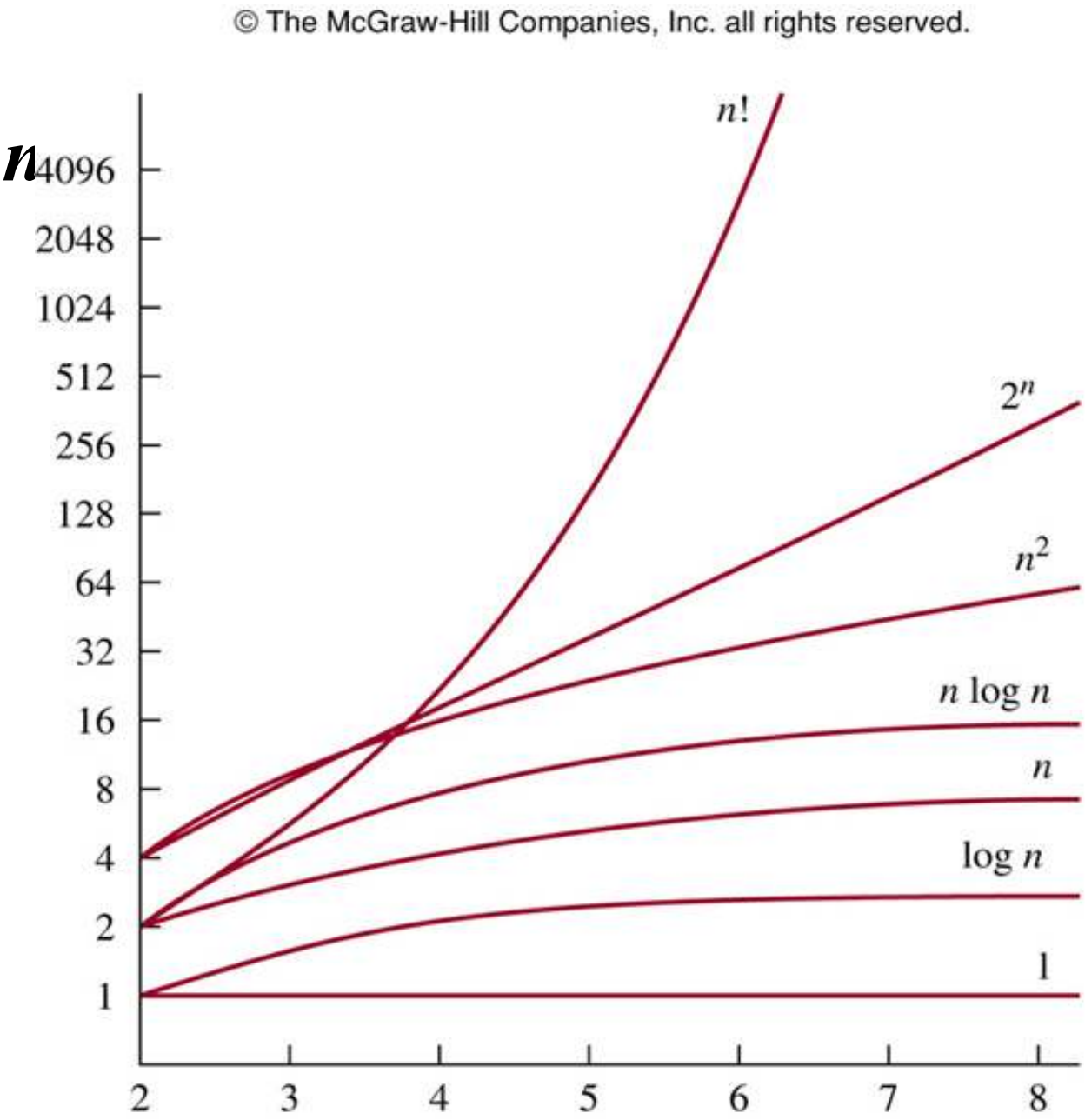
  $f(n) \leq cg(n)$ for $n \geq n_0$

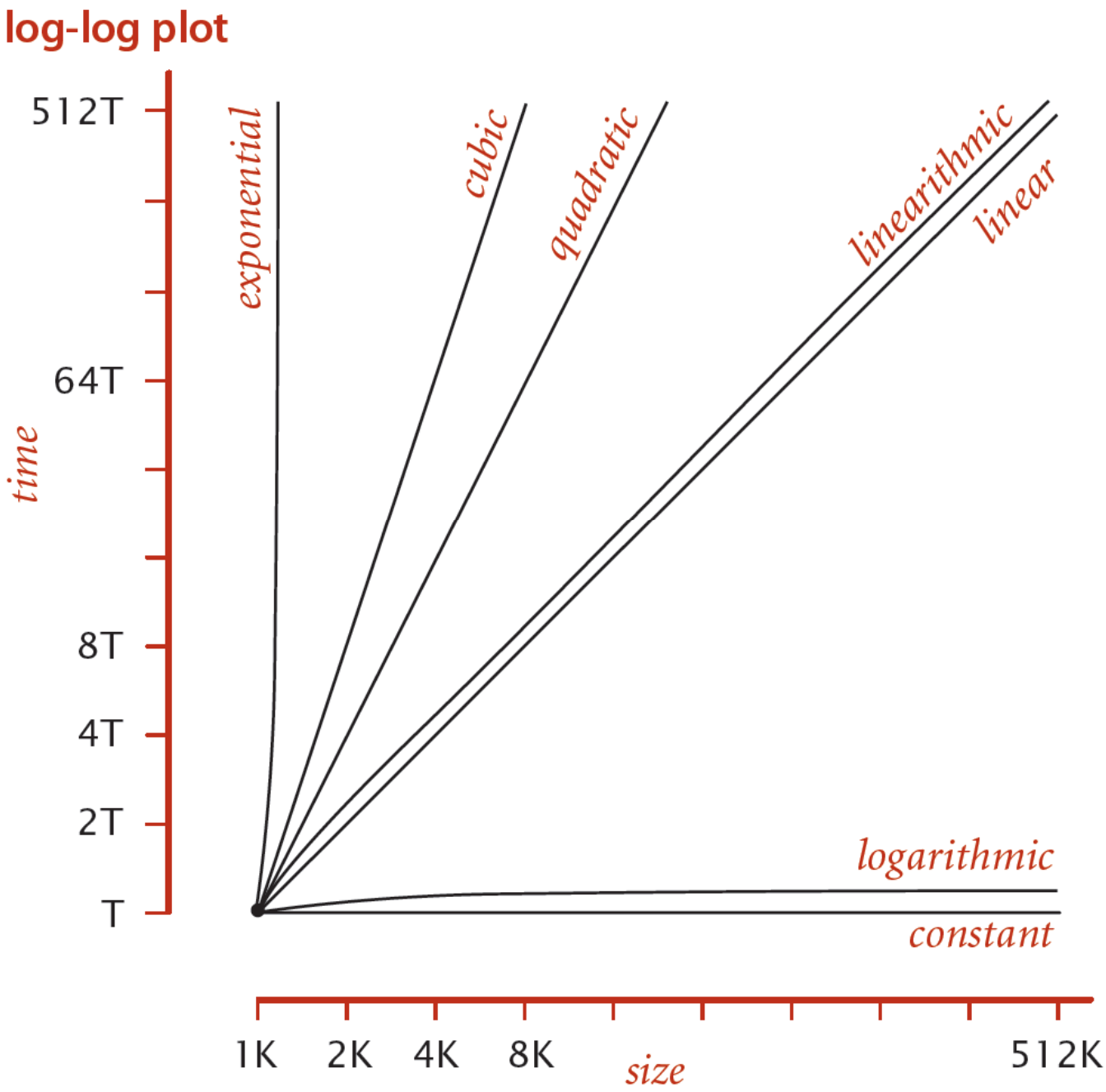
- Example: $2n + 10$ is $O(n)$
  - pick $c = 3$ and $n_0 = 10$



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: $n!$





Typical orders of growth

Practical method for Big-Oh Analysis

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

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

Common notations for algorithm analysis

notation	provides	example	shorthand for	used to
Tilde	leading term	$\sim 10\ N^2$	$10\ N^2$ $10\ N^2 + 22\ N \log N$ $10\ N^2 + 2\ N + 37$	provide approximate model
Big Theta	asymptotic growth rate	$\Theta(N^2)$	$\frac{1}{2}\ N^2$ $10\ N^2$ $5\ N^2 + 22\ N \log N + 3N$	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	$10\ N^2$ $100\ N$ $22\ N \log N + 3\ N$	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2}\ N^2$ N^5 $N^3 + 22\ N \log N + 3\ N$	develop lower bounds

Useful Approximations

- Harmonic sum

  $1 + 1/2 + 1/3 + \ldots + 1/N \sim \ln N$

- Triangular sum

  $1 + 2 + 3 + \ldots + N = N(N+1)/2 \sim N^2/2$

- Geometric sum

  $1 + 2 + 4 + \ldots + N = 2N - 1 \sim 2N$ when $N = 2^n$

- Stirling's approximation

  $\lg N! = \lg 1 + \lg 2 + \lg 3 + \ldots + \lg N \sim N \lg N$

Common order-of-growth classifications

growth rate	name	typical code framework	description	example	T(2N) / T(N)
1	constant	`a = b + c;`	statement	add two numbers	1
log N	logarithmic	`while (N > 1) { N = N / 2; ... }`	divide in half	binary search	~ 1
N	linear	`for (int i = 0; i < N; i++) { ... }`	loop	find the maximum	2
N log N	linearithmic	[see mergesort lecture]	divide and conquer	mergesort	~ 2
N^2	quadratic	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }`	double loop	check all pairs	4
N^3	cubic	`for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }`	triple loop	check all triples	8
2^N	exponential	[see combinatorial search lecture]	exhaustive search	check all subsets	T(N)

Practical implications of Order-or-growth

growth rate	problem size solvable in minutes			
	1970s	1980s	1990s	2000s
1	any	any	any	any
log N	any	any	any	any
N	millions	tens of millions	hundreds of millions	billions
N log N	hundreds of thousands	millions	millions	hundreds of millions
N^2	hundreds	thousand	thousands	tens of thousands
N^3	hundred	hundreds	thousand	thousands
2^N	20	20s	20s	30

Example 4

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

Example 4

```
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.

Example 6

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

Example 6

- **What about this:**

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

$1+2+4+8+...+ \text{n}$ is $O(n)$

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++;
    }
}
```

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

Today

- Algorithm Analysis (cont)
- Simple sorting algorithms

Sorting problem

Ex. Student records in a university.

item →

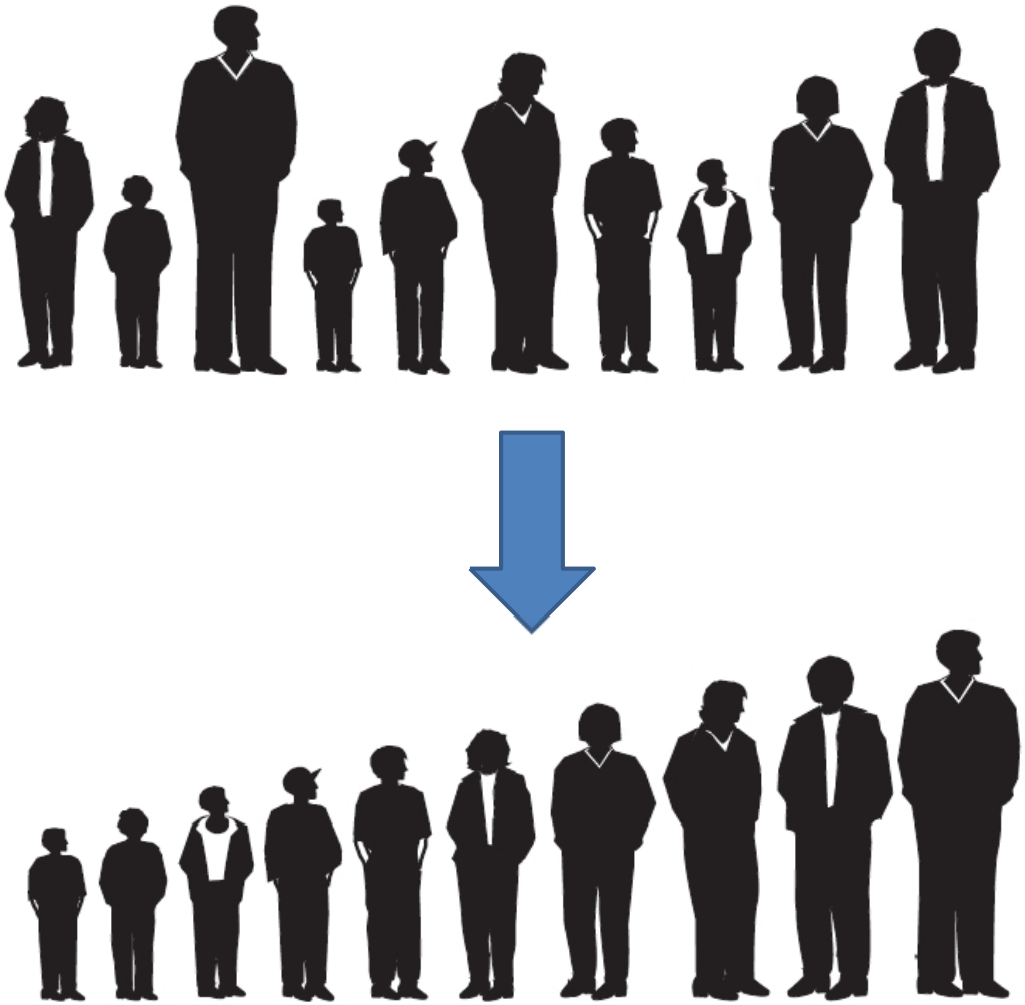
key →

Chen	3	A	991-878-4944	308 Blair
Rohde	2	A	232-343-5555	343 Forbes
Gazsi	4	B	766-093-9873	101 Brown
Furia	**1**	**A**	**766-093-9873**	**101 Brown**
Kanaga	3	B	898-122-9643	22 Brown
Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman

Sort. Rearrange array of N items into ascending order.

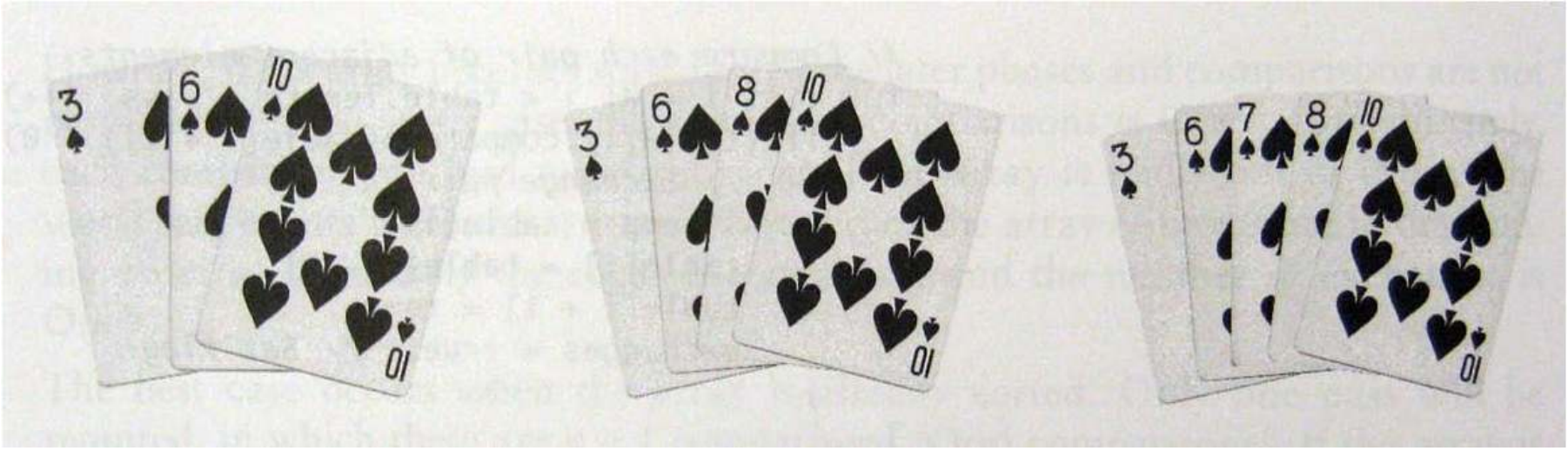
Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

Sorting Problem



Sorting Problem

- How do you sort a hand of poker cards?



Simple sort

- Bubble sort
- Selection sort
- Insertion sort

Two useful sorting abstractions

Helper functions. Refer to data through compares and exchanges.

Less. Is item `v` less than `w` ?

```
private static boolean less(Comparable v, Comparable w)
{  return v.compareTo(w) < 0;  }
```

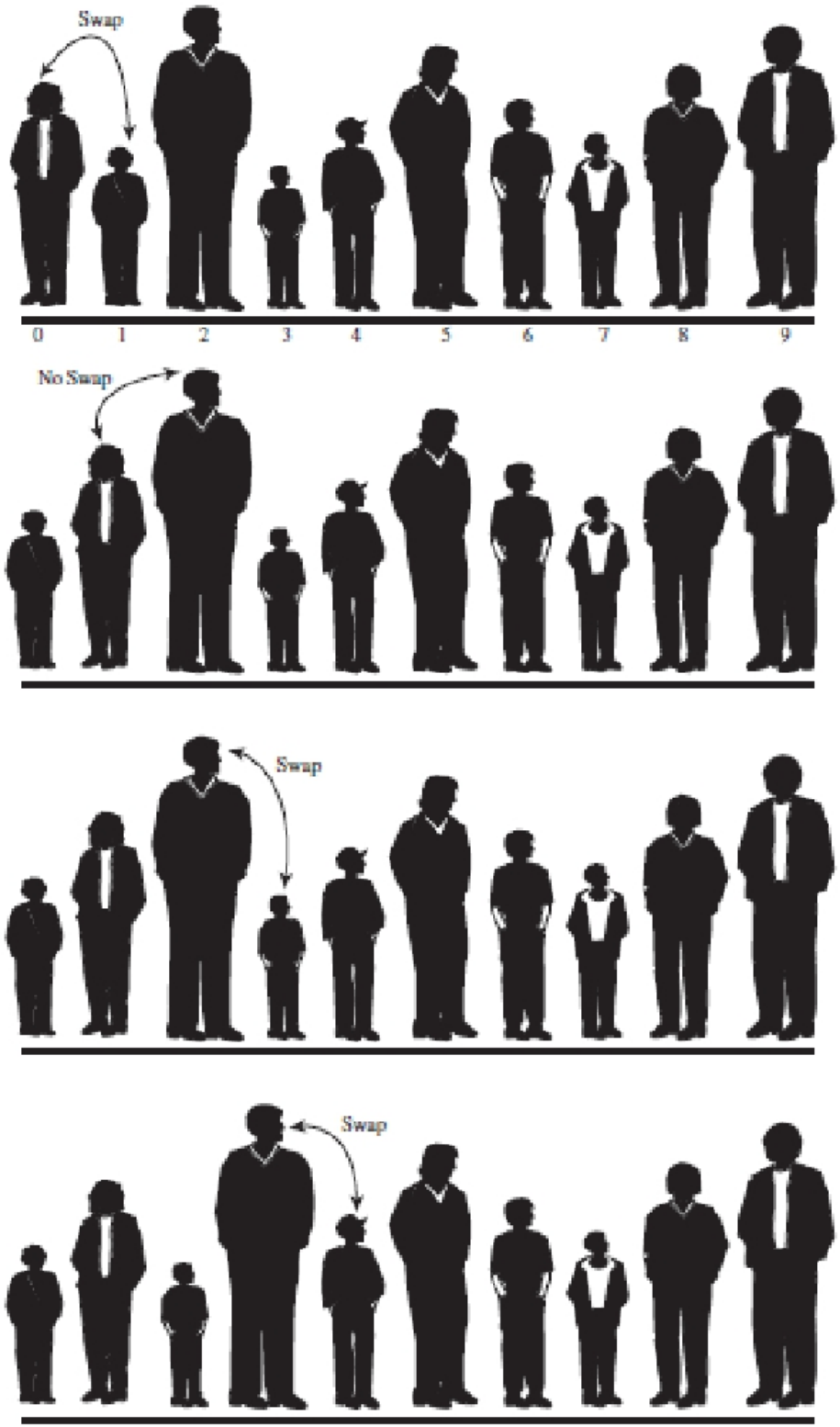
Exchange. Swap item in array `a[]` at index `i` with the one at index `j`.

```
private static void exch(Comparable[] a, int i, int j)
{
   Comparable swap = a[i];
   a[i] = a[j];
   a[j] = swap;
}
```

Bubble Sort

- Intuition:
  - Find the biggest number.
  - Find the second biggest number.
  - Find the third biggest number.
  - …
- This gives you an ordering of the numbers.

- Bubble sort achieves this by repeatedly swapping two adjacent numbers.


Swap
0 1 2 3 4 5 6 7 8 9
No Swap
Swap
Swap

Bubble Sort

- After one pass, we find the biggest number.

- It’s like the biggest ‘bubble’ floats to the top of the surface, hence the name ‘bubble sort’.

Bubble Sort

- In the second pass, we repeat the same process, but now we only have N-1 numbers to work on.
- The third pass is the same, with only N-2 numbers.
- …
- Repeat until all players are in order.

Analysis of Bubble Sort

- Number of comparisons?

- Number of swaps?

Analysis of Bubble Sort

- Number of comparisons?

$$\frac{N(N-1)}{2} = O(N^2)$$

- Number of swaps?

  best case: $O(1)$

  worst cast: $\frac{N(N-1)}{2} = O(N^2)$

  average: $\frac{N(N-1)}{4} = O(N^2)$

Selection Sort

1. Keep track of the **index** of the smallest number in each round.
2. Swap the smallest number towards the beginning of the array.
3. Repeat the above two steps.

Selection Sort

Algorithm. ↑ scans from left to right.

Invariants.

- Entries the left of ↑ (including ↑) fixed and in ascending order.
- No entry to right of ↑ is smaller than any entry to the left of ↑.

Selection Sort

- Move the pointer to the right.

```
i++;
```

- Identify index of minimum entry on right.

```
int min = i;
for (int j = i+1; j < N; j++)
   if (less(a[j], a[min]))
      min = j;
```

- Exchange into position.

```
exch(a, i, min);
```

Selection Sort Implementation

```
public class Selection
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
      {
         int min = i;
         for (int j = i+1; j < N; j++)
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min);
      }
   }

   private static boolean less(Comparable v, Comparable w)
   {  /* as before */  }

   private static void exch(Comparable[] a, int i, int j)
   {  /* as before */  }
}
```

Selection Sort

- Online demo
  - http://www.sorting-algorithms.com/selection-sort
- Gypsy dance demo
  - http://www.youtube.com/watch?v=Ns4TPTC8whw

Selection Sort

- Number of comparisons?

- Number of swaps?

Selection Sort

- Number of comparisons?

$$O(N^2)$$

- Number of swaps?

$$O(N)$$