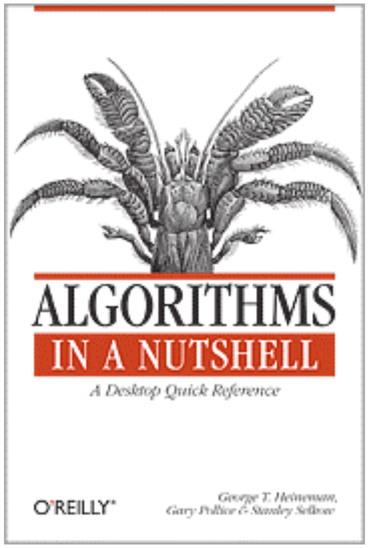
Algorithms in a Nutshell



Session 1

Introduction

9:10 – 9:40

Outline

- What is an algorithm?
- An example: INSERTION SORT
- Pattern format description
- Mathematical notations

What is an Algorithm?

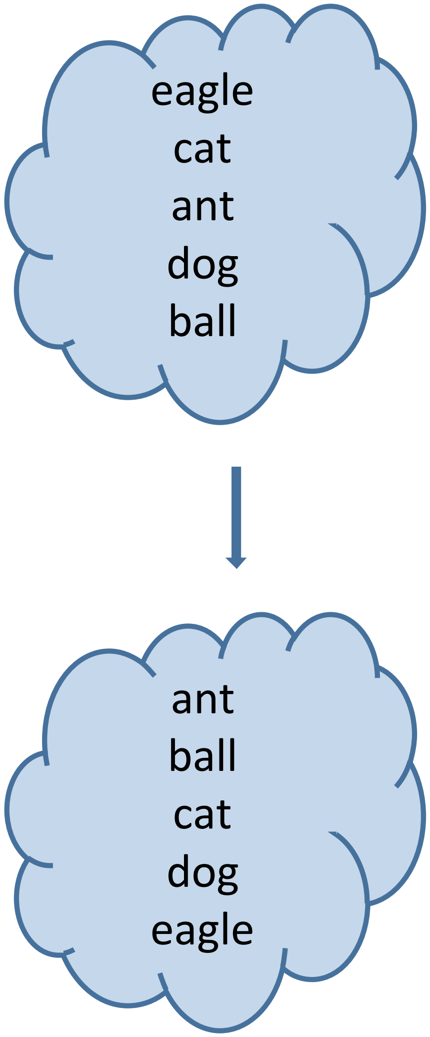
- A deterministic sequence of operations for solving a problem given a specific input set

- Deterministic – this means it always works, given the known constraints on the problem
  - Produces solution for all possible inputs
- Input Set – the (often arbitrary) way in which a problem instance is represented

Problem: Sort a collection of strings

- **Input**
  - Collection of String S
- **Output**
  - Ordered result

- **Assumptions**
  - Complete ordering between any two elements of S



eagle
cat
ant
dog
ball

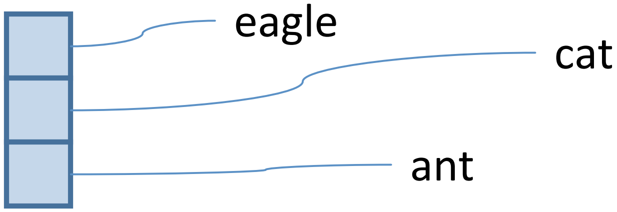
ant
ball
cat
dog
eagle

Problem Instance Representations

- Array of fixed structured content

e	a	g	l	e	¬	c	a	t	¬	¬	¬	a	n	t	¬	¬	¬

- Array of pointers to content

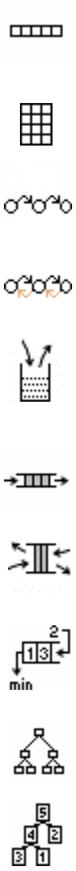


- Compact representation

e	a	g	l	e	¬	c	a	t	¬	a	n	t	¬

Common Data Structures

- Array
  - N dimensional
- Linked List
  - Doubly-linked
- Stack
- Queue
  - Double-ended queue
  - Priority queue
- Binary Tree
- Binary Heap



Data structures provide various ways to representation information

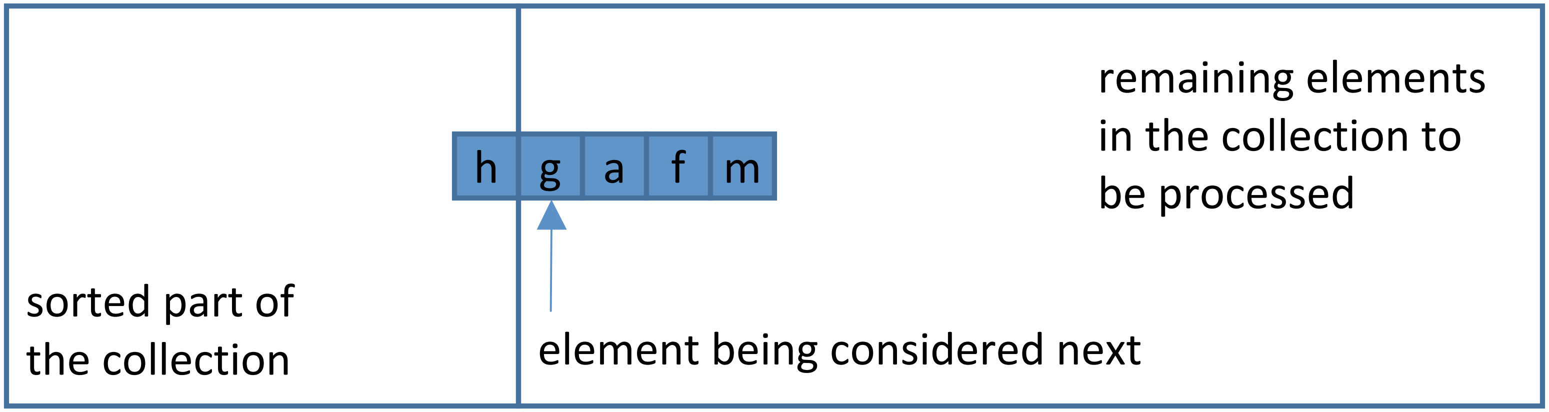
Note the glyphs used throughout the book will need to be consistent here as well

Key operations for each data structure to be discussed as needed

63

INSERTION SORT

- Frame the problem
  - *Insert* each new element into proper location



remaining elements in the collection to be processed

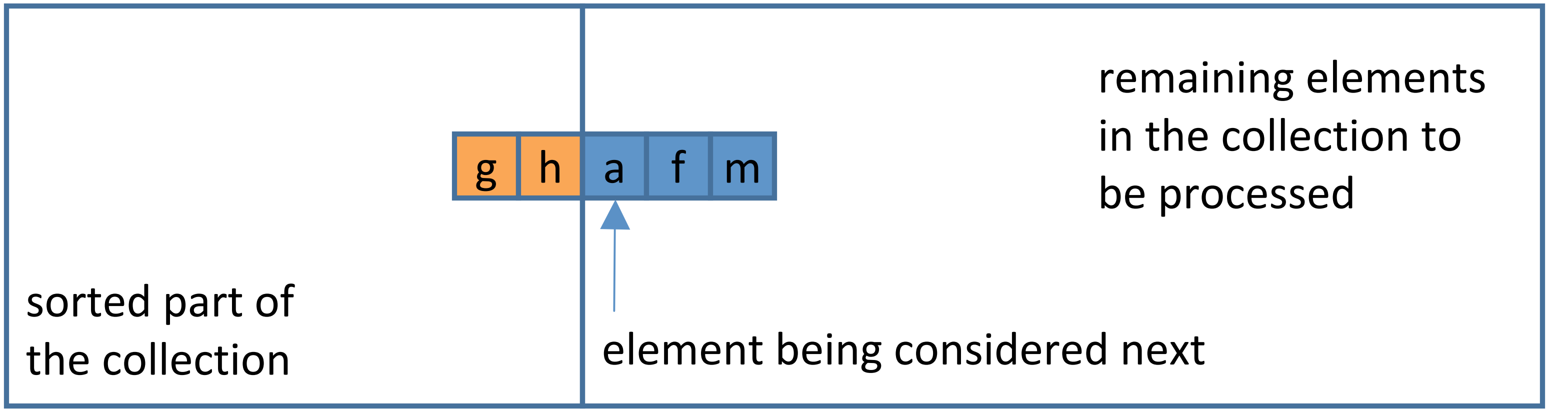
sorted part of the collection

element being considered next

Algorithm iteratively applies this key operation

INSERTION SORT

- Occasionally all elements must be moved



g	h	a	f	m

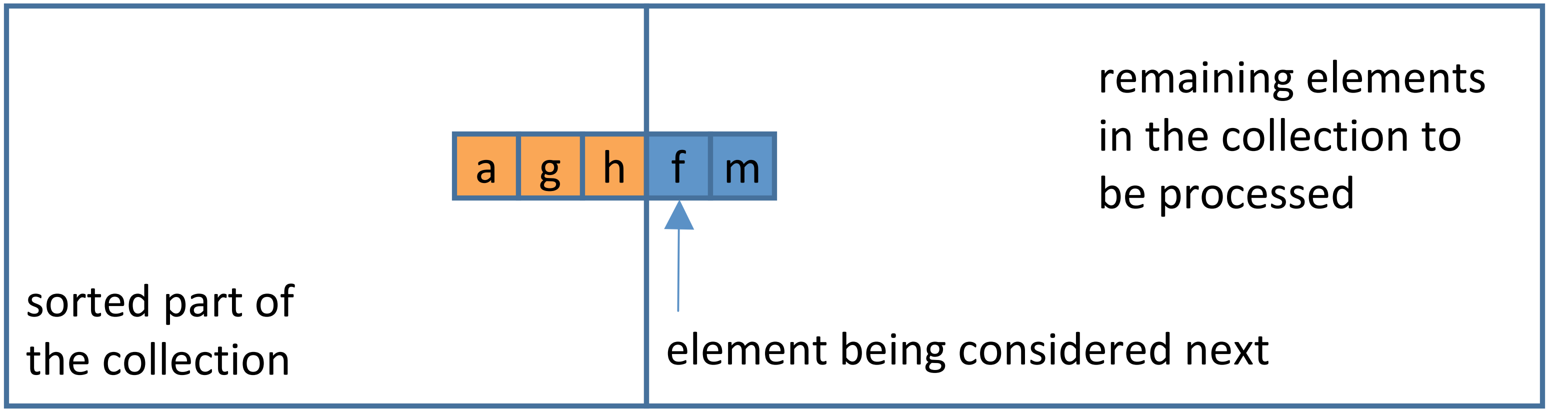
remaining elements in the collection to be processed

sorted part of the collection

element being considered next

INSERTION SORT

- Only need to move elements “higher” than the one being inserted



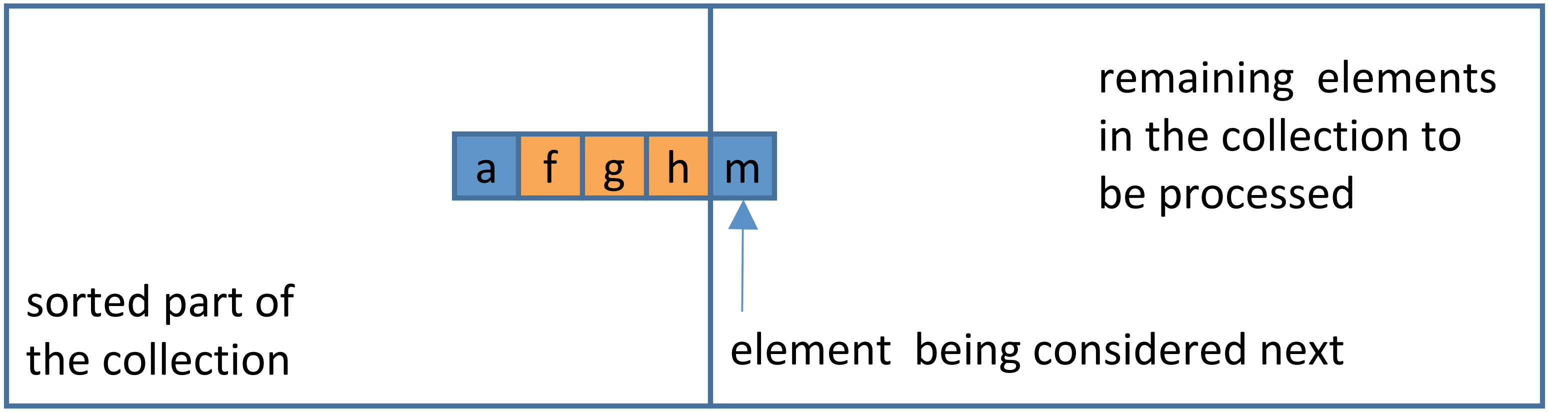
sorted part of
the collection

remaining elements
in the collection to
be processed

element being considered next

INSERTION SORT

- Sometimes the element to be inserted is greater than all existing elements – no swaps!



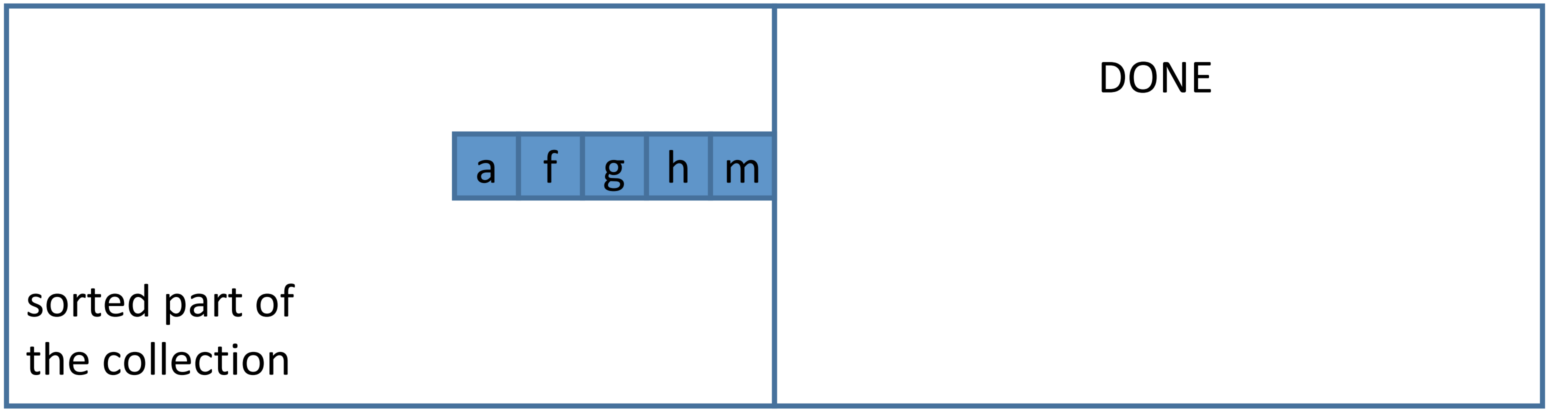
remaining elements in the collection to be processed

sorted part of the collection

element being considered next

INSERTION SORT

- Sometimes the element to be inserted is greater than all existing elements – no swaps!



64

Algorithm Fact Sheet

Name of the Algorithm

INSERTION SORT

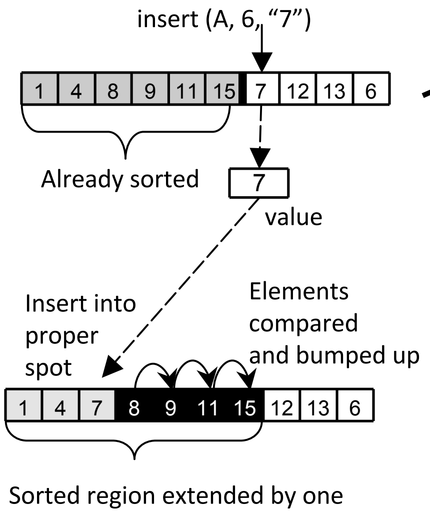
Concepts

Array

Pseudocode description

```
sort (A)
    for i=1 to n–1 do
        insert (A, i, A[i])
end

insert (A, pos, value)
    i = pos–1
    while (i ≥ 0 and A[i] > value) then
        A[i+1] = A[i]
        i = i–1
    A[i+1]=value
end
```

Small Example

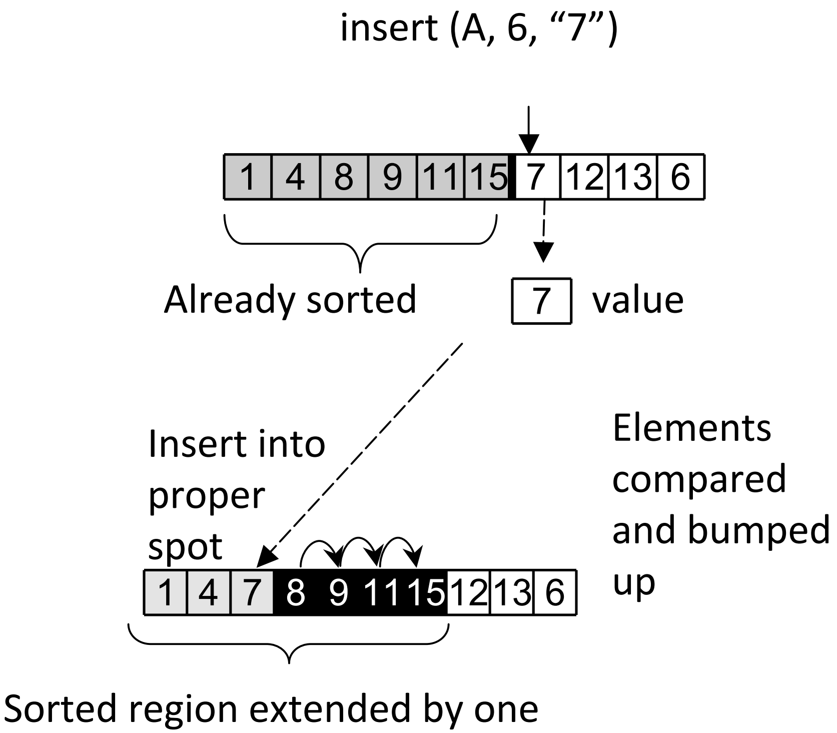


64

Array

INSERTION SORT

```
sort (A)
    for i=1 to n–1 do
        insert (A, i, A[i])
end

insert (A, pos, value)
    i = pos–1
    while (i ≥ 0 and A[i] > value) then
        A[i+1] = A[i]
        i = i–1
    A[i+1]=value
end
```



insert (A, 6, “7”)

Already sorted

value

Insert into proper spot

Elements compared and bumped up

Sorted region extended by one

Code Check

- Show actual running code
  - Handout
  - Debug example

Performance of INSERTION SORT

- As sorting algorithms go, how efficient is INSERTION SORT?
  – Is it the fastest algorithm? [NO]
  – How does it compare with other algorithms?
- Difficult to answer without a theoretic model
  – independent of programming language
  – Independent of computer hardware

Performance of Algorithms

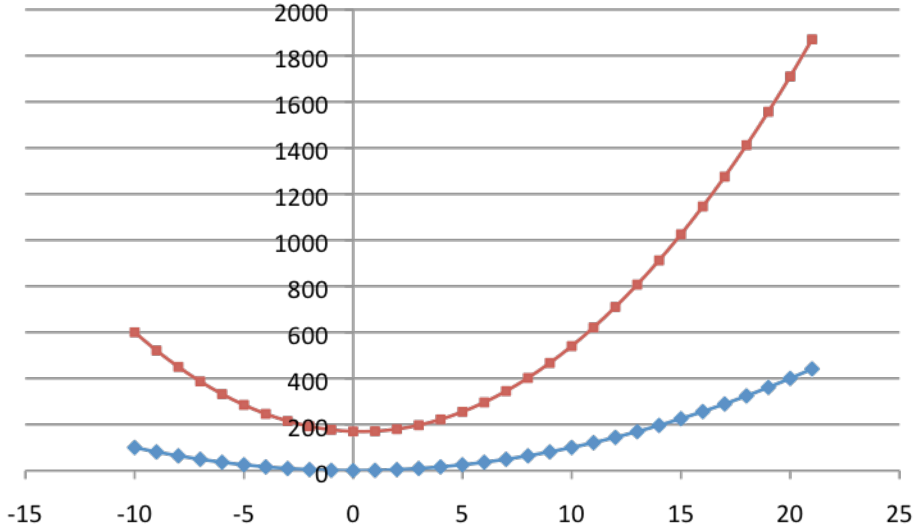
- Use standard “Big O” notation
  - Let T(*n*) be time for algorithm to perform on an average problem instance of size *n*
  - How does T(*n*) grow in proportion to increasing n?
- Example Problem
  - Given *n* integers, find the largest integer
  - Expect that T(2n) ≅ 2* T(n)
- Find performance family that most closely matches behavior of algorithm

Performance Analysis

- Linear or O(n)
  - As problem size is multiplied by 2, the time to complete the problem “should be” multiplied by 2
  - Holds true for any constant multiplicative factor
- How to capture this concept?
  - Once n is “sufficiently large”, there is some constant *c* such that $t(n) \leq c*n$
  - Not an estimate but a firm upper bound
- In practice, “c” is never computed, though its existence is guaranteed
  - Depends upon hardware platform, language, etc…

Functional Families

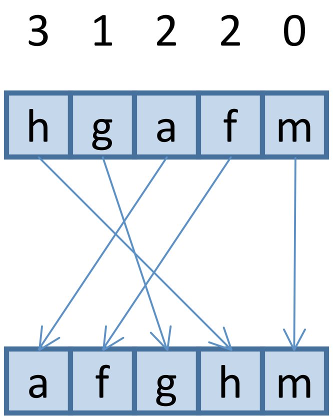
- You already know this concept from algebra
  - x^2 and $4x^2–3*x+170$ are both “quadratic” formulae
  - Distinctive shape
  - Highest exponent is most important
  - In “long run” they behave similarly



66

Performance of INSERTION SORT

- Average Case
  - Given random permutation of *n* elements, each element is (on average) *n*/3 positions from proper location



Average = 8/5 =1.6 ≅ 5/3

Performance of INSERTION SORT

- insert(A, pos, value) is invoked *n–1* times
  - On average, **while** loop invoked $n/3$ times
- Quick estimate = $(n-1)*(n/3) = \frac{1}{3}n^2 - n/3$
  - The critical factor is the highest exponent
  - $\frac{1}{3}n^2 - n/3$ is $O(n^2)$
- INSERTION SORT is not linear
  - It is Quadratic
  - As problem size is multiplied by $k=2$, the time to complete the problem is multiplied by $k^2=4$

```
sort (A)
  for i=1 to n–1 do
    insert (A, i, A[i])
end

insert (A, pos, value)
  i = pos–1
  while (i ≥ 0 and A[i] > value) then
    A[i+1] = A[i]
    i = i–1
  A[i+1]=value
end
```

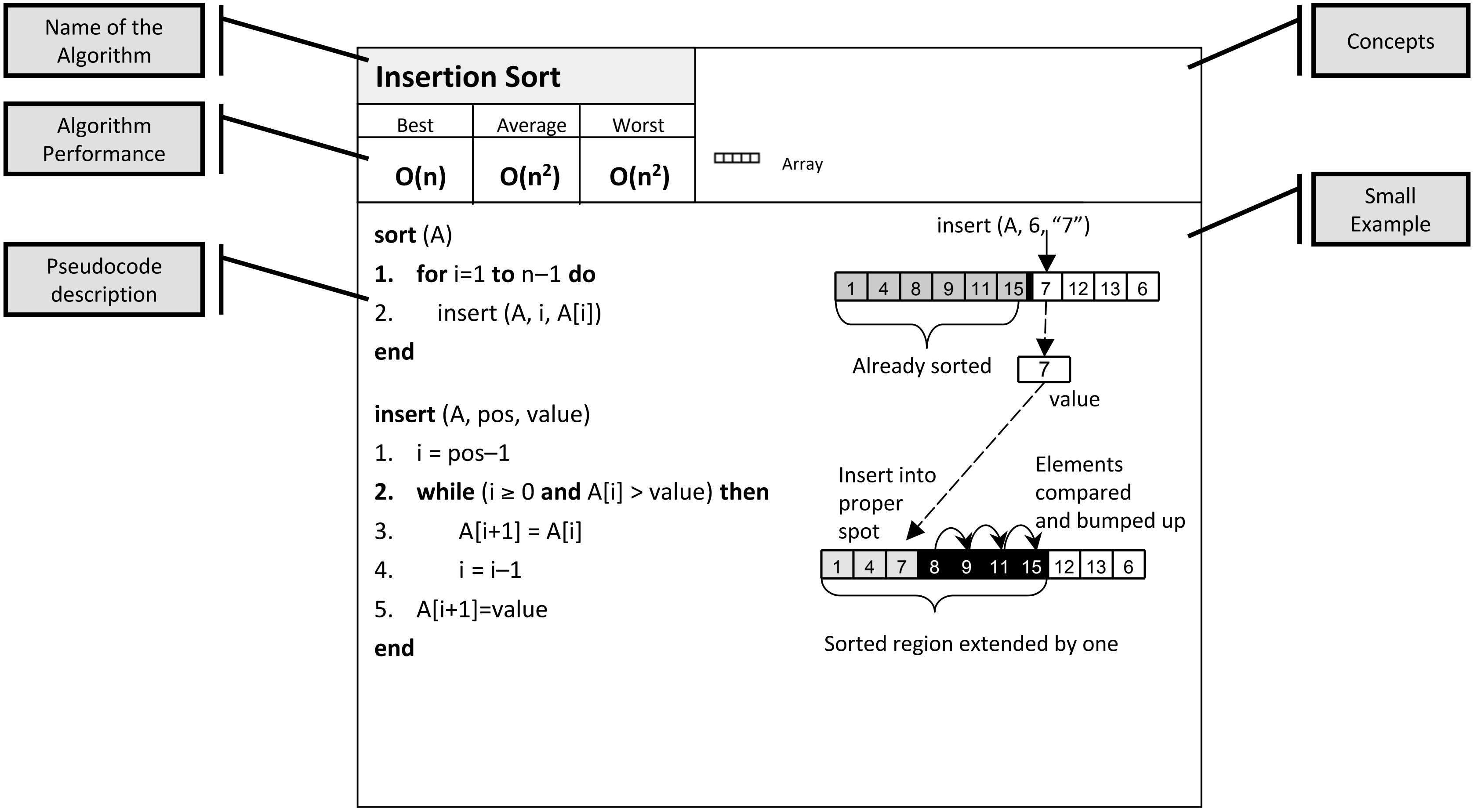
18

Rating Performance of Algorithm

- Best Case
  - Problem instances for which algorithm computes answer most efficiently
- Average Case
  - Typical random problem instance. Identifies the expected performance of the algorithm
- Worst Case
  - Unusual problem instances that force algorithm to work harder and be less efficient

64

INSERTION SORT Fact Sheet



Name of the Algorithm

Algorithm Performance

Pseudocode description

Concepts

Small Example

Insertion Sort

Best	Average	Worst
O(n)	**$O(n^2)$**	**$O(n^2)$**

Array

```
sort (A)
for i=1 to n–1 do
    insert (A, i, A[i])
end

insert (A, pos, value)
i = pos–1
while (i ≥ 0 and A[i] > value) then
    A[i+1] = A[i]
    i = i–1
A[i+1]=value
end
```

41

Algorithm Pattern Format

• Name	descriptive name
• Synopsis	what it is designed to do
• Context	“sweet spot” for algorithm
• Forces	implementation issues
• Solution	actual code for algorithm
• Consequences	advantages/disadvantages
• Analysis	show performance

```
sort (A)
  for i=1 to n–1 do
    insert (A, i, A[i])
end

insert (A, pos, value)
  i = pos–1
  while (i ≥ 0 and A[i] > value) then
    A[i+1] = A[i]
    i = i–1
  A[i+1]=value
end
```

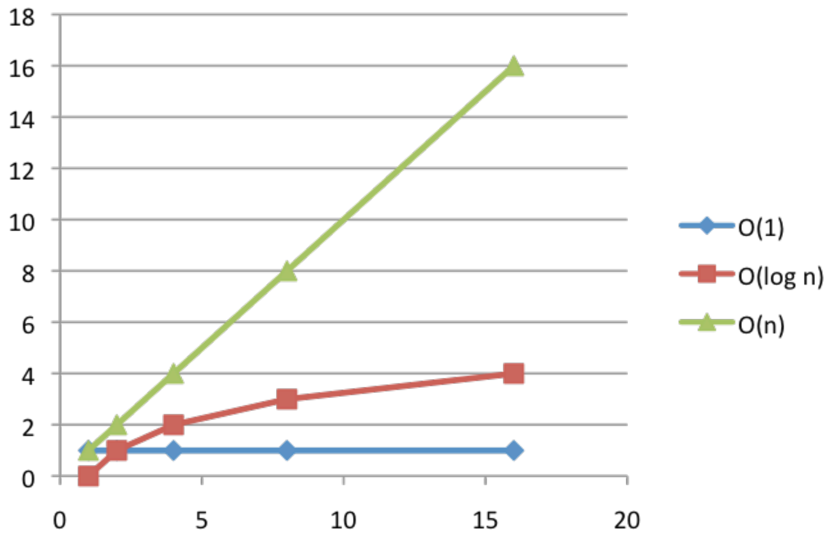
```
void sortPointers (char **ar, int n) {
 int j;
 for (j = 1; j < n; j++) {
  int i = j-1;
  char *value = ar[j];
  while (i >= 0 && strcmp(ar[i], value)> 0) {
   ar[i+1] = ar[i];
   i--;
  }

  ar[i+1] = value;
 }
}
```

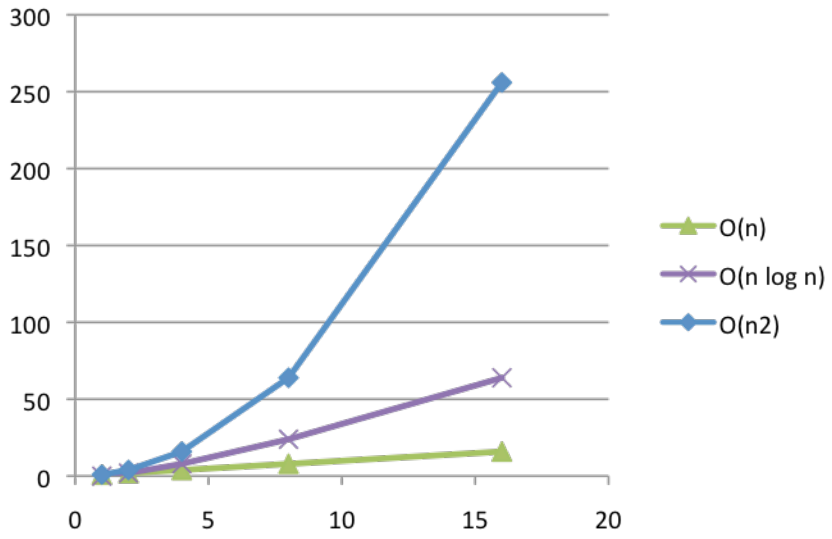
- Key Ideas
  - While locating proper location, move up those in the way
  - Simple pointer swap

- Implementation
  - No need for separate method calls
  - Assume pairwise swap is efficient
  - Can work with generic **compare** method

Rating Performance of Algorithm

- Analyze the standard “Big O” notation
  - Consider how well algorithm performs on problem instances of size *n*
- Key families include:





Simplified “Big O” notation

- We use O(…) notation a bit informally
  - For proper academic use, we should use Θ(…)
  - In academic use
    - O(…) means upper bound
    - Ω(…) means lower bound
    - Θ(…) declares both a tight upper and lower bound
- Reason?
  - Simpler presentation
  - Common ‘shorthand’ usage in industry

Summary

- Algorithm introductions
  - INSERTION SORT
- Pattern Format
- Analysis tools
  - “Big O” notation