Sorting
Chapter 7
Objectives

- Understand the importance of the sort problem
- Analyze the running times of different sorting algorithms
- Choose the most efficient sorting algorithms based on the problem requirements
Sorting

- For simplicity, we will assume no repeated values
- The values have a total order
- All comparisons are done through the $<$ or $>$ operators
Insertion Sort

- For $j = 2$ to $n$
  - Keep $A[1..j]$ sorted
Insertion Sort

For $j = 1$ to $n$
    key = $A[j]$
    $i = j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = key$
Selection Sort

- For $j = 1$ to $n$
  - Find the $j$th smallest element and put it in place

```plaintext
For j = 1 to n
  min = j
  for i = j+1 to n
    if A[i] < A[min]
      min = i
  swap(A[j], A[min])
```
Selection Sort

- For $j = 1$ to $n$
  - Find the $j^{th}$ smallest element and put it in place

```
For j = 1 to n
    min = j
    for i = j+1 to n
        if A[i] < A[min]
            min = i
    swap(A[j], A[min])
```
Bubble Sort

- Whenever you find an unordered pair, reorder them

```
For j = 1 to n
    For i = 1 to n-1
        if A[i] > A[i+1]
            swap(A[i], A[i+1])
```
Bubble Sort

Whenever you find an unordered pair, reorder them

For \( j = 1 \) to \( n \)
    For \( i = 1 \) to \( n-j \)
        if \( A[i] > A[i+1] \)
            swap(\( A[i] \), \( A[i+1] \))
Bubble Sort

Whenever you find an unordered pair, reorder them

For j = 1 to n
    sorted = true
    For i = 1 to n-j
        if A[i] > A[i+1]
            swap(A[i], A[i+1])
            sorted = false
    break if sorted
Shell Sort

- Bubble sort and insertion sort make a very slow progress
- Shell sort tries to make bigger leaps
Shell Sort

For gap = n/2 downto 1; gap = gap/2
for j = gap to n
for i = 1 to n-j
  if A[i] > A[i+gap]
    swap(A[i], A[i+gap])