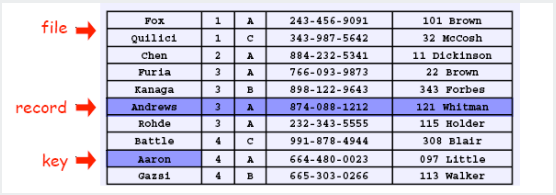
Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- sorting challenges
- shellsort

Reference: Algorithms in Java, Chapter 6

Basic terms

Ex: student record in a University.



Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Sort: rearrange sequence of objects into ascending order.

Aaron	4	A	664-480-0023	097 Little
Andrews	3	A	874-088-1212	121 Whitman
Battle	4	C	991-878-4944	308 Blair
Chen	2	A	884-232-5341	11 Dickinson
Fox	1	A	243-456-9091	101 Brown
Furia	3	A	766-093-9873	22 Brown
Gazsi	4	B	665-303-0266	113 Walker
Kanaga	3	B	898-122-9643	343 Forbes
Rohde	3	A	232-343-5555	115 Holder
Quilici	1	C	343-987-5642	32 McCosh

rules of the game
insertion sort
selection sort
sorting challenges
shellsort

Sample sort client

Goal: Sort any type of data

Example. List the files in the current directory, sorted by file name.

```
import java.io.File;
public class Files
{
   public static void main(String[] args)
   {
      File directory = new File(args[0]);
      File[] files = directory.listFiles();
      Insertion.sort(files);
      for (int i = 0; i < files.length; i++)
         System.out.println(files[i]);
   }
}
```

```
% java Files .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
ShellX.class
ShellX.java
index.html
```

Next: How does sort compare file names?

Callbacks

Goal. Write robust sorting library method that can sort any type of data using the data type's natural order.

Callbacks.
- Client passes array of objects to sorting routine.
- Sorting routine calls back object's comparison function as needed.

Implementing callbacks.
- Java: interfaces.
- C: function pointers.
- C++: functors.

Callbacks

client

```
import java.io.File;
public class SortFiles
{
   public static void main(String[] args)
   {
      File directory = new File(args[0]);
      File[] files = directory.listFiles();
      sort(files);
      for (int i = 0; i < files.length; i++)
         System.out.println(files[i]);
   }
}
```

interface

```
interface Comparable <Item>
{
   public int compareTo(Item);
}
```

object implementation

```
public class File
implements Comparable
{
   ...
   public int compareTo(File b)
   {
      ...
      return -1;
      ...
      return +1;
      ...
      return 0;
   }
}
```

sort implementation

```
public static void sort(Comparable[] a)
{
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0; j--)
         if (a[j].compareTo(a[j-1]))
            exch(a, j, j-1);
         else break;
}
```

Key point: no reference to File

Callbacks

Goal. Write robust sorting library that can sort any type of data into sorted order using the data type's natural order.

Callbacks.
- Client passes array of objects to sorting routine.
- Sorting routine calls back object's comparison function as needed.

Implementing callbacks.
- Java: interfaces.
- C: function pointers.
- C++: functors.

Plus: Code reuse for all types of data
Minus: Significant overhead in inner loop

This course:
- enables focus on algorithm implementation
- use same code for experiments, real-world data

Interface specification

Comparable interface.
Must implement method `compareTo()` so that `v.compareTo(w)` returns:
- A negative integer if `v` is less than `w`.
- A positive integer if `v` is greater than `w`.
- Zero if `v` is equal to `w`.

Consistency. Implementation must ensure a total order.
- If (a < b) and (b < c), then (a < c).
- Either (a < b) or (b < a) or (a = b).

Built-in comparable types. `String`, `Double`, `Integer`, `Date`, `File`.
User-defined comparable types. Implement the `Comparable` interface.

Implementing the Comparable interface: dates

```
public class Date implements Comparable<Date>
{
   private int month, day, year;

   public Date(int m, int d, int y)
   {
      month = m;
      day   = d;
      year  = y;
   }

   public int compareTo(Date b)
   {
      Date a = this;
      if (a.year  < b.year ) return -1;
      if (a.year  > b.year ) return +1;
      if (a.month < b.month) return -1;
      if (a.month > b.month) return +1;
      if (a.day   < b.day  ) return -1;
      if (a.day   > b.day  ) return +1;
      return 0;
   }
}
```

only compare dates to other dates

Implementing the comparable interface: domain names

To study web traffic.

- Subdomain: `bolle.cs.princeton.edu`.
- Reverse subdomain: `edu.princeton.cs.bolle`.
- Sort by reverse subdomain to group by category.

```
public class Domain implements Comparable<Domain>
{
   private String[] fields;
   public Domain(String name)
   { fields = name.split("\\."); }
   public int length()
   { return fields.length; }
   public int compareTo(Domain b)
   {
      Domain a = this;
      int N = Math.min(a.length, b.length);
      for (int i = N; i > 0; i--)
      {
         int c = a.fields[i].compareTo(b.fields[i]);
         if      (c < 0) return -1;
         else if (c > 0) return +1;
      }
      return a.length - b.length;
   }
```

Details are for the bored...

unsorted

```
ee.princeton.edu
cs.princeton.edu
princeton.edu
cnn.com
google.com
apple.com
www.cs.princeton.edu
bolle.cs.princeton.edu
```

sorted

```
com.apple
com.cnn
com.google
edu.princeton
edu.princeton.cs
edu.princeton.cs.bolle
edu.princeton.cs.www
edu.princeton.ee
```


Sample sort clients

File names

```
import java.io.File;
public class Files
{
   public static void main(String[] args)
   {
      File directory = new File(args[0]);
      File[] a = directory.listFiles()
      Insertion.sort(a);
      for (int i = 0; i < a.length; i++)
         System.out.println(a[i]);
   }
}
```

```
% java Files .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
```

Random numbers

```
public class Experiment
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      Double[] a = new Double[N];
      for (int i = 0; i < N; i++)
         a[i] = Math.random();
      Selection.sort(a);
      for (int i = 0; i < N; i++)
         System.out.println(a[i]);
   }
}
```

```
% java Experiment 10
0.08614716385210452
0.09054270895414829
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

Several Java library data types implement Comparable

You can implement Comparable for your own types

Two useful abstractions

Helper functions. Refer to data only through two operations.

- less. Is `v` less than `w` ?

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

- exchange. Swap object in array at index `i` with the one at index `j`.

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

Check if sorted

Example usage. Is the input sorted?

```
public static boolean isSorted(Comparable[] a)
{
   for (int i = 1; i < a.length; i++)
      if (less(a[i], a[i-1]))
         return false;
   return true;
}
```


Why use less() and exch() ?

Easy switch to faster implementation for primitive types

```
private static boolean less(double v, double w)
{
   return v < w;
}
```

Instrument for experimentation and animation

```
private static boolean less(double v, double w)
{
   cnt++;
   return v < w;
}
```

Translate to other languages

```
...
for (int i = 1; i < a.length; i++)
      if (less(a[i], a[i-1]))
         return false;
   return true;}
...
```

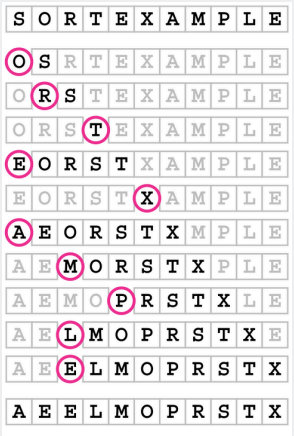
Good code in C, C++, Python, JavaScript, Ruby....

Insertion sort

Insertion sort.

- Scans from left to right.
- Element to right of ↑ are not touched.
- Invariant: elements to the left of ↑ are in ascending order.
- Inner loop: repeatedly swap element ↑ with element to its left.

Insertion sort example



rules of the game
insertion sort
selection sort
sorting challenges
shellsort

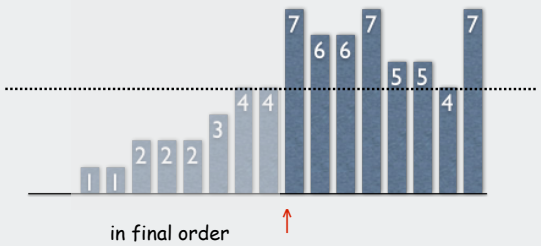
Insertion sort: Java implementation

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


Selection sort

Selection sort.

- ↑ scans from left to right.
- Elements to the left of ↑ are fixed and in ascending order.
- No element to left of ↑ is larger than any element to its right.



Selection sort: example

Selection sort: Java implementation

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


Maintaining an invariant

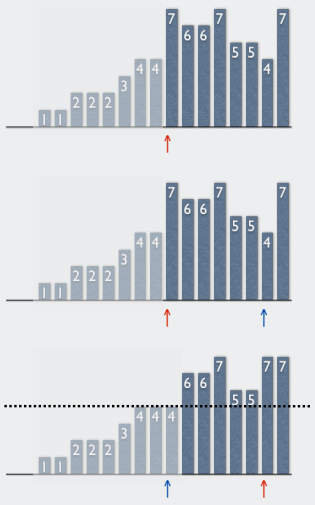
Selection sort inner loop.

- Identify index of minimum item.

```
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);
```



Performance for randomly-ordered files

Selection.

- Always search through right part.
- (1 + 2 + ... + N) ≈ N^2 / 2 compares.
  ≈ N exchanges.

Insertion.

- Each element moves halfway back.
- (1 + 2 + ... + N) / 2 ≈ N^2 / 4 compares.
  ≈ N^2 / 4 exchanges.

Bottom line: performance difference are in details of implementation.

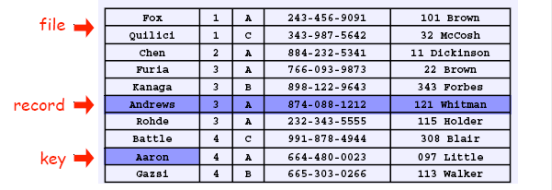
rules of the game
selection sort
insertion sort
sorting challenges
shellsort

Sorting Challenge 1

Problem: sort a file of huge records with tiny keys.
Ex: reorganizing your MP3 files.

Which sorting method to use?

1. system sort
2. insertion sort
3. selection sort



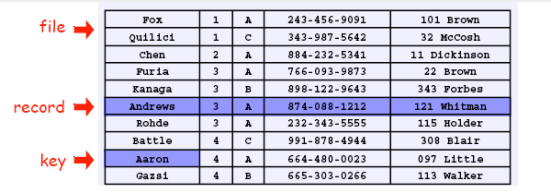
Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Sorting Challenge 2

Problem: sort a huge randomly-ordered file of small records.
Ex: process transaction records for a phone company.

Which sorting method to use?

1. system sort
2. insertion sort
3. selection sort



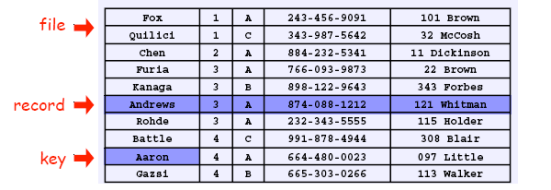
Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Sorting Challenge 3

Problem: sort a huge number of tiny files (each file is independent)
Ex: daily customer transaction records.

Which sorting method to use?

1. system sort
2. insertion sort
3. selection sort



Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Sorting Challenge 4

Problem: sort a huge file that is already almost in order.
Ex: re-sort a huge database after a few changes.

Which sorting method to use?

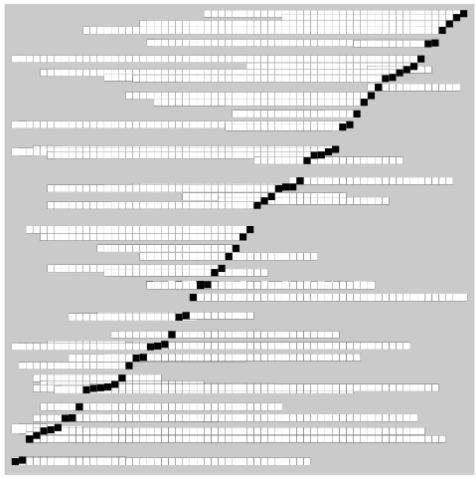
1. system sort
2. insertion sort
3. selection sort

file ➡
record ➡
key ➡

Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

rules of the game
selection sort
insertion sort
sorting challenges
shellsort

Visual representation of insertion sort



Bottom line: slow data movement

Shellsort

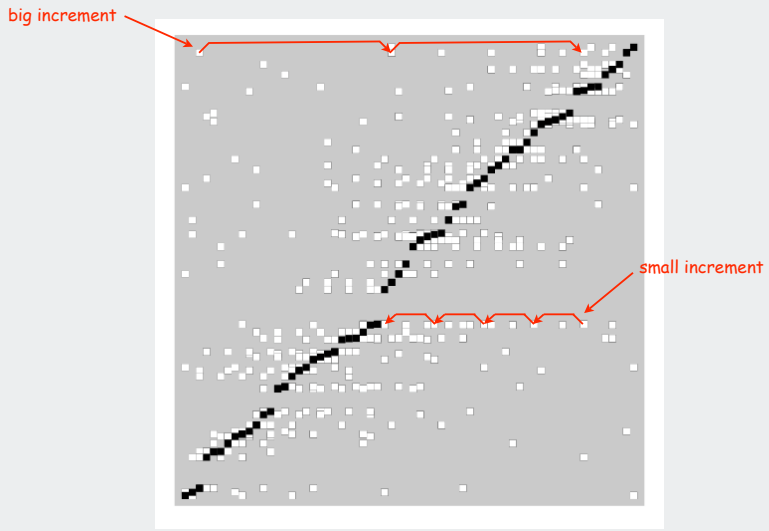
Idea: move elements more than one position at a time
For a decreasing sequence of increments

- think of file as h independent files (elements spaced h apart)
- run insertion sort on each

```
public static void sort(double[] a)
{
    int N = a.length;
    int[] incs = { 1391376, 463792, 198768, 86961,
                   33936, 13776, 4592, 1968, 861, 336,
                   112, 48, 21, 7, 3, 1 };
    for (int k = 0; k < incs.length; k++)
    {
        int h = incs[k];
        for (int i = h; i < N; i++)
            for (int j = i; j >= h; j-= h)
                if (less(a[j], a[j-h]))
                    exch(a, j, j-h);
                else break;
            }
    }
}
```

← insertion sort

Visual representation of shellsort



Bottom line: substantially faster!

Analysis of shellsort

No good model has yet been discovered (!)

N	Comparisons	$N^{1.289}$	2.5 N lg N
5,000	93	58	106
10,000	209	143	230
20,000	467	349	495
40,000	1022	855	1059
80,000	2266	2089	2257

measured in thousands

Why are we interested in shellsort?

Example of simple idea leading to substantial performance gains

Useful in practice
- fast unless file size is huge
- tiny, fixed footprint for code
- hardware sort prototype

Simple algorithm, nontrivial performance
- asymptotic growth rate?
- best sequence of increments?
- average case performance?

Your first open problem in algorithmics (see Section 6.8):

Find a better increment sequence

`mail rs@cs.princeton.edu`

Lesson: some good algorithms are still waiting discovery