Acknowledgement: The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.
Elementary Sorting Algorithms

- Sorting review
- Rules of the game
- Selection sort
- Insertion sort
- Shellsort
Elementary Sorting Algorithms

- Sorting review
- Rules of the game
- Selection sort
- Insertion sort
- Shellsort
### Ex.

**Student records in a university.**

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Key Value</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen</td>
<td>A</td>
<td>991-878-4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Rohde</td>
<td>A</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
</tr>
<tr>
<td>Gazsi</td>
<td>B</td>
<td>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Furia</td>
<td>A</td>
<td>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Kanaga</td>
<td>B</td>
<td>898-122-9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Andrews</td>
<td>A</td>
<td>664-480-0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Battle</td>
<td>C</td>
<td>874-088-1212</td>
<td>121 Whitman</td>
</tr>
</tbody>
</table>

### Sort.

**Rearrange array of** $N$ **items into ascending order.**

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Key Value</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrews</td>
<td>A</td>
<td>664-480-0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Battle</td>
<td>C</td>
<td>874-088-1212</td>
<td>121 Whitman</td>
</tr>
<tr>
<td>Chen</td>
<td>A</td>
<td>991-878-4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Furia</td>
<td>A</td>
<td>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Gazsi</td>
<td>B</td>
<td>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Kanaga</td>
<td>B</td>
<td>898-122-9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Rohde</td>
<td>A</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
</tr>
</tbody>
</table>
Goal. Sort any type of data.

Ex 1. Sort random real numbers in ascending order.

Sample sort client

```java
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] = StdRandom.uniform();
        Insertion.sort(a);
        for (int i = 0; i < N; i++)
            StdOut.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

seems artificial, but stay tuned for an application
Sample sort client

**Goal.** Sort *any* type of data.

**Ex 2.** Sort strings from file in alphabetical order.

```java
public class StringSorter {
    public static void main(String[] args) {
        String[] a = In.readStrings(args[0]);
        Insertion.sort(a);
        for (int i = 0; i < a.length; i++)
            StdOut.println(a[i]);
    }
}
```

% more words3.txt
bed bug dad yet zoo ... all bad yes

% java StringSorter words3.txt
all bad bed bug dad ... yes yet zoo
**Goal.** Sort any type of data.

**Ex 3.** Sort the files in a given directory by filename.

```java
import java.io.File;
public class FileSorter {
    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++)
            StdOut.println(files[i].getName());
    }
}
```
**Callbacks**

**Goal.** Sort any type of data.

**Q.** How can `sort()` know how to compare data of type `Double`, `String`, and `java.io.File` without any information about the type of an item's key?

**Callback = reference to executable code.**

- Client passes array of objects to `sort()` function.
- The `sort()` function calls back object's `compareTo()` method as needed.

**Implementing callbacks.**

- **Java:** interfaces.
- **C:** function pointers.
- **C++:** class-type functors.
- **C#:** delegates.
- **Python, Perl, ML, Javascript:** first-class functions.
Callbacks: roadmap

**Client**

```java
import java.io.File;
public class FileSorter
{
    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++)
            StdOut.println(files[i].getName());
    }
}
```

**Object implementation**

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

**Comparable interface (built in to Java)**

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

**Sort implementation**

```java
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]) < 0)
                exch(a, j, j-1);
            else break;
}
```

**Key point:** no dependence on `File` data type
A total order is a binary relation \( \leq \) that satisfies

- **Antisymmetry**: if \( v \leq w \) and \( w \leq v \), then \( v = w \).
- **Transitivity**: if \( v \leq w \) and \( w \leq x \), then \( v \leq x \).
- **Totality**: either \( v \leq w \) or \( w \leq v \) or both.

**Ex.**

- Standard order for natural and real numbers.
- Alphabetical order for strings.
- Chronological order for dates.
- ...

![an intransitive relation](image)
Implement `compareTo()` so that `v.compareTo(w)`

- Is a total order.
- Returns a negative integer, zero, or positive integer if `v` is less than, equal to, or greater than `w`, respectively.
- Throws an exception if incompatible types (or either is `null`).

Built-in comparable types. `Integer`, `Double`, `String`, `Date`, `File`, ...

User-defined comparable types. Implement the `Comparable` interface.
Implementing the Comparable interface

**Date data type.** Simplified version of `java.util.Date`.

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

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

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

Only compare dates to other dates.
Two useful sorting abstractions

**Helper functions.** Refer to data through compares and exchanges.

**Less.** Is item \( v \) less than \( w \) ?

```java
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 \( j \).

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

- Sorting review
- Rules of the game
- Selection sort
- Insertion sort
- Shellsort
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[min]$. 
Selection sort

- In iteration $i$, find index $\text{min}$ of smallest remaining entry.
- Swap $a[i]$ and $a[\text{min}]$. 

Remaining entries
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. 

![Remaining entries diagram]
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$.

in final order

remaining entries
Selection sort

- In iteration \( i \), find index \( \text{min} \) of smallest remaining entry.
- Swap \( a[i] \) and \( a[\text{min}] \).
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. 

![Card deck with selection sort example](image)
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. 

![Image of playing cards](image-url)
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. 

Remaining entries and in final order
Selection sort

- In iteration $i$, find index $\text{min}$ of smallest remaining entry.
- Swap $a[i]$ and $a[\text{min}]$. 

![Diagram of selection sort with cards]
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$.
Selection sort

- In iteration $i$, find index $\text{min}$ of smallest remaining entry.
- Swap $a[i]$ and $a[\text{min}]$. 

![Diagram showing selection sort process with playing cards]
Selection sort

- In iteration $i$, find index $\text{min}$ of smallest remaining entry.
- Swap $a[i]$ and $a[\text{min}]$. 

![Diagram of selection sort with cards and indices indicating the sorting process.](image)
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. 

in final order

remaining entries
Selection sort

- In iteration \( i \), find index \( \min \) of smallest remaining entry.
- Swap \( a[i] \) and \( a[\min] \).
• In iteration $i$, find index $\min$ of smallest remaining entry.
• Swap $a[i]$ and $a[\min]$.
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. 

![Diagram showing selection sort process with playing cards](image-url)
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. 

![Image showing selection sort process with playing cards.](image-url)
Selection sort

- In iteration \( i \), find index \( \text{min} \) of smallest remaining entry.
- Swap \( a[i] \) and \( a[\text{min}] \).
In iteration $i$, find index $\min$ of smallest remaining entry.
Swap $a[i]$ and $a[\min]$.
Selection sort

- In iteration \( i \), find index \( \text{min} \) of smallest remaining entry.
- Swap \( a[i] \) and \( a[\text{min}] \).
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$.
Selection sort

- In iteration $i$, find index $\text{min}$ of smallest remaining entry.
- Swap $a[i]$ and $a[\text{min}]$.  

![Image of playing cards with selection sort process]

- in final order
- remaining entries
Selection sort

• In iteration $i$, find index $\min$ of smallest remaining entry.
• Swap $a[i]$ and $a[\min]$. 

![Card image](image_url)
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$.
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. 

![Diagram showing selection sort process with playing cards.](image-url)
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. 
In iteration $i$, find index $\text{min}$ of smallest remaining entry.
Swap $a[i]$ and $a[\text{min}]$. 

Selection sort
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. 

in final order
Selection sort

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$. sorted
Selection sort: Java implementation

```java
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: mathematical analysis**

**Proposition.** Selection sort uses \((N-1) + (N-2) + \ldots + 1 + 0 \sim N^2 / 2\) compares and \(N\) exchanges.

<table>
<thead>
<tr>
<th>i</th>
<th>min</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>A</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>A</td>
<td>E</td>
<td>R</td>
<td>T</td>
<td>O</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>T</td>
<td>O</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>R</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>O</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>X</td>
<td>S</td>
<td>O</td>
<td>P</td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>S</td>
<td>X</td>
<td>P</td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>X</td>
<td>S</td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
</tr>
</tbody>
</table>

**Trace of selection sort (array contents just after each exchange)**

Entries in black are examined to find the minimum. Entries in red are \(a[\text{min}]\). Entries in gray are in final position.

**Running time insensitive to input.** Quadratic time, even if input array is sorted. Data movement is minimal. Linear number of exchanges.
Selection sort: animations

20 random items

algorithm position

in final order
not in final order

http://www.sorting-algorithms.com/selection-sort
Selection sort: animations

20 partially-sorted items

http://www.sorting-algorithms.com/selection-sort
Elementary Sorting Algorithms

- Sorting review
- Rules of the game
- Selection sort
- Insertion sort
- Shellsort
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
Selection sort

• In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
- In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.

In ascending order

not yet seen
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
In iteration $i$, swap $a[i]$ with each larger entry to its left.
In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.

Insertion sort
In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[j]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.

![Card images]

- not yet seen
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

- In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
public class Insertion
{
    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;
    }

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

    private static void exch(Comparable[] a, int i, int j)
    { /* as before */  }
}
**Proposition.** To sort a randomly-ordered array with distinct keys, insertion sort uses $\sim \frac{1}{4} N^2$ compares and $\sim \frac{1}{4} N^2$ exchanges on average.

**Pf.** Expect each entry to move halfway back.

---

Trace of insertion sort (array contents just after each insertion)

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a[]</td>
<td>S O R T E X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>O S R T E X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>O R S T E X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>O R S T E X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>E O R S T E X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>E O R S T X E A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>A E O R S T X E A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>A E O R S T X E A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>A E O M R S T X E A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>A E L M O P R S T X E A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>A E E L M O P R S T X A E E L M O P R S T X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

entries in gray do not move

entry in red is a[j]

entries in black moved one position right for insertion
Insertion sort: animation

40 random items

http://www.sorting-algorithms.com/insertion-sort
Insertion sort: best and worst case

**Best case.** If the array is in ascending order, insertion sort makes $N - 1$ compares and 0 exchanges.

```
A E E L M O P R S T X
```

**Worst case.** If the array is in descending order (and no duplicates), insertion sort makes $\sim \frac{1}{2} N^2$ compares and $\sim \frac{1}{2} N^2$ exchanges.

```
X T S R P O M L E E A
```
Insertion sort: animation

40 reverse-sorted items

http://www.sorting-algorithms.com/insertion-sort
Def. An **inversion** is a pair of keys that are out of order.

\[
\text{AEELMOTRXPS}
\]

- T-R T-P T-S R-P X-P X-S

(6 inversions)

Def. An array is **partially sorted** if the number of inversions is \( \leq cN \).
- Ex 1. A subarray of size 10 appended to a sorted subarray of size \( N \).
- Ex 2. An array of size \( N \) with only 10 entries out of place.

**Proposition.** For partially-sorted arrays, insertion sort runs in linear time.

**Pf.** Number of exchanges equals the number of inversions.

\[
\text{number of compares} = \text{exchanges} + (N - 1)
\]
Insertion sort: animation

40 partially-sorted items

http://www.sorting-algorithms.com/insertion-sort
Elementary Sorting Algorithms

- Sorting review
- Rules of the game
- Selection sort
- Insertion sort
- Shellsort
**Shellsort overview**

**Idea.** Move entries more than one position at a time by \( h \)-**sorting** the array.

An \( h \)-sorted array is \( h \) interleaved sorted subsequences

\[ h = 4 \]

\[
\begin{array}{ccccccccccccccc}
L & E & E & A & M & H & L & E & P & S & O & L & T & S & X & R \\
L & M & P & T \\
E & H & S & S \\
E & L & O & X \\
A & E & L & R \\
\end{array}
\]

**Shellsort. [Shell 1959]** \( h \)-sort the array for decreasing seq. of values of \( h \).

**Input**

<table>
<thead>
<tr>
<th>S</th>
<th>H</th>
<th>E</th>
<th>L</th>
<th>L</th>
<th>S</th>
<th>O</th>
<th>R</th>
<th>T</th>
<th>E</th>
<th>X</th>
<th>A</th>
<th>M</th>
<th>P</th>
<th>L</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>H</td>
<td>E</td>
<td>L</td>
<td>L</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>S</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>L</td>
<td>E</td>
<td>E</td>
<td>A</td>
<td>M</td>
<td>H</td>
<td>L</td>
<td>E</td>
<td>P</td>
<td>S</td>
<td>O</td>
<td>L</td>
<td>T</td>
<td>S</td>
<td>X</td>
<td>R</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>S</td>
<td>T</td>
<td>X</td>
</tr>
</tbody>
</table>
How to \( h \)-sort an array? Insertion sort, with stride length \( h \).

### 3-sorting an array

<table>
<thead>
<tr>
<th>M</th>
<th>O</th>
<th>L</th>
<th>E</th>
<th>E</th>
<th>X</th>
<th>A</th>
<th>S</th>
<th>P</th>
<th>R</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>O</td>
<td>L</td>
<td>M</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>S</td>
<td>P</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>X</td>
<td>A</td>
<td>S</td>
<td>P</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>X</td>
<td>A</td>
<td>S</td>
<td>P</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>X</td>
<td>M</td>
<td>S</td>
<td>P</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>X</td>
<td>M</td>
<td>S</td>
<td>P</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>P</td>
<td>M</td>
<td>S</td>
<td>X</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>P</td>
<td>M</td>
<td>S</td>
<td>X</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>P</td>
<td>M</td>
<td>S</td>
<td>X</td>
<td>R</td>
<td>T</td>
</tr>
</tbody>
</table>

### Why insertion sort?

- Big increments \( \Rightarrow \) small subarray.
- Small increments \( \Rightarrow \) nearly in order. [stay tuned]
Shellsort example: increments 7, 3, 1

**input**

```
SORTEXAMPLE
```

**7-sort**

```
SORTEXAMPLE
MOREXAMPLE
MOREXAMPLE
MOREXAMPLE
MOREXAMPLE
MOREXAMPLE
```

**3-sort**

```
MOREXAMPLE
ELMОСXAPRT
ELMОСXAPRT
ELMОСXAPRT
```

**1-sort**

```
AELEOPMSXT
AELEOPMSXT
AELEOPMSXT
AELEOPMSXT
```

**result**

```
AEELMOPRSTX
```
Shellsort: intuition

Proposition. A $g$-sorted array remains $g$-sorted after $h$-sorting it.

7-sort

M O R T E X A S P L E
M O R T E X A S P L E
M O L T E X A S P R E
M O L E E X A S P R T
M O L E E X A S P R T

3-sort

M O L E E X A S P R T
E O L M E X A S P R T
E E L M O X A S P R T
E E L M O X A S P R T
A E L E O X M S P R T
A E L E O X M S P R T
A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T

still 7-sorted
Shellsort: which increment sequence to use?

Powers of two. 1, 2, 4, 8, 16, 32, ...
No.

Powers of two minus one. 1, 3, 7, 15, 31, 63, ...
Maybe.

3x + 1. 1, 4, 13, 40, 121, 364, ...
OK. Easy to compute.

Sedgewick. 1, 5, 19, 41, 109, 209, 505, 929, 2161, 3905, ...
Good. Tough to beat in empirical studies.

Interested in learning more?

• See Section 6.8 of Algs, 3rd edition or Volume 3 of Knuth for details.
• Do a JP on the topic.
public class Shell
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;

        int h = 1;
        while (h < N/3) h = 3*h + 1; // 1, 4, 13, 40, 121, 364, 1093, ...

        while (h >= 1)
        {
            // h-sort the array.
            for (int i = h; i < N; i++)
            {
                for (int j = i; j >= h && less(a[j], a[j-h]); j -= h)
                exch(a, j, j-h);
            }
            h = h/3;
        }
    }

    private static boolean less(Comparable v, Comparable w)
    { /* as before */ }
    private static boolean void(Comparable[] a, int i, int j)
    { /* as before */ }
}
Shellsort: visual trace

input

40-sorted

13-sorted

4-sorted

result
Shellsort: animation

50 random items

http://www.sorting-algorithms.com/shell-sort
Shellsort: animation

50 partially-sorted items

http://www.sorting-algorithms.com/shell-sort
Proposition. The worst-case number of compares used by shellsort with the $3x+1$ increments is $O(N^{3/2})$.

Property. The number of compares used by shellsort with the $3x+1$ increments is at most by a small multiple of $N$ times the # of increments used.

<table>
<thead>
<tr>
<th>$N$</th>
<th>compares</th>
<th>$N^{1.289}$</th>
<th>$2.5 \times N \lg N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.000</td>
<td>93</td>
<td>58</td>
<td>106</td>
</tr>
<tr>
<td>10.000</td>
<td>209</td>
<td>143</td>
<td>230</td>
</tr>
<tr>
<td>20.000</td>
<td>467</td>
<td>349</td>
<td>495</td>
</tr>
<tr>
<td>40.000</td>
<td>1022</td>
<td>855</td>
<td>1059</td>
</tr>
<tr>
<td>80.000</td>
<td>2266</td>
<td>2089</td>
<td>2257</td>
</tr>
</tbody>
</table>

measured in thousands

Remark. Accurate model has not yet been discovered (!)
Why are we interested in shellsort?

Example of simple idea leading to substantial performance gains.

Useful in practice.
- Fast unless array size is huge.
- Tiny, fixed footprint for code (used in embedded systems).
- Hardware sort prototype.

Simple algorithm, nontrivial performance, interesting questions.
- Asymptotic growth rate?
- Best sequence of increments?
- Average-case performance?

Lesson. Some good algorithms are still waiting discovery.