2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- binary search
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- binary search
Problem. Given an array of $n$ elements, rearrange in ascending order by key.

<table>
<thead>
<tr>
<th>Last</th>
<th>First</th>
<th>House</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longbottom</td>
<td>Neville</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Weasley</td>
<td>Ron</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Abbott</td>
<td>Hannah</td>
<td>Hufflepuff</td>
<td>1998</td>
</tr>
<tr>
<td>Potter</td>
<td>Harry</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Chang</td>
<td>Cho</td>
<td>Ravenclaw</td>
<td>1997</td>
</tr>
<tr>
<td>Granger</td>
<td>Hermione</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Malfoy</td>
<td>Draco</td>
<td>Slytherin</td>
<td>1998</td>
</tr>
<tr>
<td>Diggory</td>
<td>Cedric</td>
<td>Hufflepuff</td>
<td>1996</td>
</tr>
<tr>
<td>Weasley</td>
<td>Ginny</td>
<td>Gryffindor</td>
<td>1999</td>
</tr>
<tr>
<td>Parkinson</td>
<td>Pansy</td>
<td>Slytherin</td>
<td>1998</td>
</tr>
</tbody>
</table>
**Sorting problem**

**Problem.** Given an array of $n$ elements, rearrange in ascending order by key.

<table>
<thead>
<tr>
<th>Last</th>
<th>First</th>
<th>House</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abbott</td>
<td>Hannah</td>
<td>Hufflepuff</td>
<td>1998</td>
</tr>
<tr>
<td>Chang</td>
<td>Cho</td>
<td>Ravenclaw</td>
<td>1997</td>
</tr>
<tr>
<td>Granger</td>
<td>Hermione</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Diggory</td>
<td>Cedric</td>
<td>Hufflepuff</td>
<td>1996</td>
</tr>
<tr>
<td>Longbottom</td>
<td>Neville</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Malfoy</td>
<td>Draco</td>
<td>Slytherin</td>
<td>1998</td>
</tr>
<tr>
<td>Parkinson</td>
<td>Pansy</td>
<td>Slytherin</td>
<td>1998</td>
</tr>
<tr>
<td>Potter</td>
<td>Harry</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Weasley</td>
<td>Ron</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Weasley</td>
<td>Ginny</td>
<td>Gryffindor</td>
<td>1999</td>
</tr>
</tbody>
</table>
Sorting is a well-defined problem if there is a binary relation $\leq$ that satisfies:

- **Totality:** either $v \leq w$ or $w \leq v$ or both.
- **Transitivity:** if both $v \leq w$ and $w \leq x$, then $v \leq x$.

Examples.

<table>
<thead>
<tr>
<th>No.</th>
<th>Video name</th>
<th>Views (billions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>&quot;Baby Shark Dance&quot;</td>
<td>10.15</td>
</tr>
<tr>
<td>2.</td>
<td>&quot;Despacito&quot;</td>
<td>7.73</td>
</tr>
<tr>
<td>3.</td>
<td>&quot;Johny Johny Yes Papa&quot;</td>
<td>6.15</td>
</tr>
<tr>
<td>4.</td>
<td>&quot;Shape of You&quot;</td>
<td>5.61</td>
</tr>
<tr>
<td>5.</td>
<td>&quot;See You Again&quot;</td>
<td>5.41</td>
</tr>
</tbody>
</table>
Sorting problem

Sorting is a well-defined problem if there is a binary relation $\leq$ that satisfies:

- **Totality:** either $v \leq w$ or $w \leq v$ or both.
- **Transitivity:** if both $v \leq w$ and $w \leq x$, then $v \leq x$.

Mathematically, a "weak order"

Non-examples.

- Ro–sham–bo order (violates transitivity)
- Course prerequisites (violates totality)
- The $\leq$ operator for double (irreflexive, which violates totality)
Sample sort clients

**Goal.** General-purpose sorting function.

**Ex 1.** Sort strings in **lexicographic order.** *alphabetical order, using Unicode character ordering*

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

```bash
~/cos226/sort> more words3.txt
bed bug dad yet zoo ... all bad yes

~/cos226/sort> java StringSorter < words3.txt
all bad bed bug dad ... yes yet zoo
[suppressing newlines]
```
Sample sort clients

Goal. General-purpose sorting function.

Ex 2. Sort real numbers in numerical order (ascending).

```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.uniformDouble();
        Insertion.sort(a);
        for (int i = 0; i < n; i++)
            StdOut.println(a[i]);
    }
}
```

```
~/cos226/sort> java Experiment 10
0.08614716385210452
0.09054270895414829
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```
Sample sort clients

**Goal.** General-purpose sorting function.

**Ex 3.** Sort playing cards by *suit and rank*.

```java
public class Deck {
    ...

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        PlayingCard[] cards = deal(n);
        Insertion.sort(cards);
        draw(cards);
    }
}
```

![Playing cards sorted and drawn](image)
How can a single function sort any type of data?

**Goal.** General-purpose sorting function.

*Please sort these Japanese names for me:*
あゆみ, アユミ, Ayumi, 歩美, ....

*But I don’t speak Japanese and I don’t know how words are ordered.*

*No problem. Whenever you need to compare two words, give me a call back.*

*オーケー. Just make sure to use a weak order.*
Callbacks

Goal. General-purpose sorting function.

Solution. **Callback** = reference to executable code passed to other code and later executed.

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

Implementing callbacks.

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

*In effect, client passes `compareTo()` method to `sort()` function; the callback occurs when `sort()` invokes `compareTo()`*
**Review: Java interfaces**

**Interface.** A set of related methods that define some behavior (partial API) for a class.

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

**Class that implements interface.** Must implement all interface methods.

```java
public class String implements Comparable<String> {
    ...
    public int compareTo(String that) {
        ...
    }
}
```
Callbacks in Java: roadmap

**client** (StringSorter.java)

```java
public class StringSorter {
    public static void main(String[] args) {
        String[] a = StdIn.readAllStrings();
        Insertion.sort(a);
        ...
    }
}
```

**sort implementation** (Insertion.java)

```java
public class Insertion {
    public static void sort(Comparable[] a) {
        ...
        if (a[i].compareTo(a[j]) < 0)
            ...
    }
}
```

*key point: sorting code does not depend upon type of data to be sorted*

**interface** (Comparable.java)

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

**data type implementation** (String.java)

```java
public class String implements Comparable<String> {
    ...
    public int compareTo(String that) {
        ...
    }
}
```

*String[] is a subtype of Comparable[]*
Suppose that the Java architects left out `implements Comparable<String>` in the class declaration for `String`. What would be the effect?

A. Compile-time error in `String.java`.
B. Compile-time error in `StringSorter.java`.
C. Compile-time error in `Insertion.java`.
D. Run-time exception in `Insertion.java`.
Comparable API

Implement compareTo() so that v.compareTo(w)

- Returns a negative integer if \( v \) is less than \( w \).
- Returns a positive integer if \( v \) is greater than \( w \).
- Returns zero if \( v \) is equal to \( w \).
- Throws an exception if incompatible types (or either is \texttt{null})

API requirement:  
the binary relation  
v.compareTo(w) \leq 0  
is a weak order

Built-in comparable types.  Integer, Double, String, java.util.Date, ...

User-defined comparable types.  Implement 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;
    }
}
```

Implementing the Comparable interface can compare Date objects only to other Date objects.
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- binary search
**Algorithm.** For each index $i$ from 0 to $n - 1$:

- Find index $\text{min}$ of smallest remaining element.
- Swap elements at indices $i$ and $\text{min}$.  

**Initial array**
Selection sort: visualization

Visualization. Sort vertical bars by length.

https://www.toptal.com/developers/sorting-algorithms/selection-sort
Selection sort invariants

Algorithm. For each index $i$ from 0 to $n - 1$:
- Find index $\text{min}$ of smallest remaining element.
- Swap elements at indices $i$ and $\text{min}$.

Invariants.

before iteration $i$

\[
\begin{array}{cccc}
\text{sorted} & i & \text{min} \\
\text{smallest } i \text{ elements} & \ldots & \\
\end{array}
\]

after iteration $i$

\[
\begin{array}{cccc}
\text{sorted} & \text{smallest } i + 1 \text{ elements} & \ldots & \\
\end{array}
\]
Two useful sorting primitives (and a cost model)

**Helper functions.** Refer to data only through *compares* and *exchanges*. e.g., no calls to equals()

use as our cost model for sorting

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

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

polymorphic method call

use interface type as argument
⇒ method works for all subtypes

**Exchange.** Swap array entries \( a[i] \) and \( a[j] \).

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

Java arrays are "covariant"
(e.g., String[] is a subtype of Object[])
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) {
        /* see previous slide */
    }

    private static void exch(Object[] a, int i, int j) {
        /* see previous slide */
    }
}

https://algs4.cs.princeton.edu/21elementary/Selection.java.html
How many compares to selection sort an array of $n$ distinct items in reverse order?

A. $\sim n$

B. $\sim \frac{1}{4} n^2$

C. $\sim \frac{1}{2} n^2$

D. $\sim n^2$
**Selection sort: mathematical analysis**

**Proposition.** Selection sort makes \((n - 1) + (n - 2) + \ldots + 1 + 0 \approx \frac{1}{2} n^2\) compares and \(n\) exchanges to sort any array of \(n\) items.

**Running time insensitive to input.** \(\Theta(n^2)\) compares.  

**Data movement is minimal.** \(\Theta(n)\) exchanges.  

**In place.** \(\Theta(1)\) extra space.
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- binary search
Algorithm. For each index $i = 0$ to $n - 1$:

- Let $x$ be the element at index $i$.
- Repeatedly exchange $x$ with each larger element to its immediate left.
Insertion sort invariants

Algorithm. For each index $i = 0$ to $n - 1$:

- Let $x$ be the element at index $i$.
- Repeatedly exchange $x$ with each larger element to its immediate left.

Invariants.

before iteration $i$

<table>
<thead>
<tr>
<th>sorted</th>
<th>$i$</th>
<th>untouched</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leq x$</td>
<td>$&gt; x$</td>
<td>$x$</td>
</tr>
</tbody>
</table>

after iteration $i$

<table>
<thead>
<tr>
<th>sorted</th>
<th>untouched</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leq x$</td>
<td>$x$</td>
</tr>
</tbody>
</table>
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(Object[] a, int i, int j) {
        /* as before */
    }
}

https://algs4.cs.princeton.edu/21elementary/Insertion.java.html
How many compares to insertion sort an array of \( n \) distinct keys in reverse order?

A. \( \sim n \)

B. \( \sim \frac{1}{4} n^2 \)

C. \( \sim \frac{1}{2} n^2 \)

D. \( \sim n^2 \)
Insertion sort: running time analysis

**Worst case.** Insertion sort makes \( \sim \frac{1}{2} n^2 \) compares and \( \sim \frac{1}{2} n^2 \) exchanges to sort an array of \( n \) distinct keys in reverse order.

**Pf.** Exactly \( i \) compares and exchanges in iteration \( i \).

\[
0 + 1 + 2 + \ldots + (n-1) \sim \frac{1}{2} n^2
\]
Best case. Insertion sort makes $n - 1$ compares and 0 exchanges to sort an array of $n$ distinct keys in ascending order.
Insertion sort: running time analysis

**Good case.** Insertion sort takes $\Theta(n)$ time on “partially sorted” arrays.

**Q.** Can we formalize what we mean by partially sorted?

**A.** Yes, in terms of “inversions” (see textbook).

[Image: https://www.toptal.com/developers/sorting-algorithms/insertion-sort]
Insertion sort: practical improvements

**Half exchanges.** Shift items over (instead of exchanging).
- Same compares; fewer array accesses.
- No longer uses only `less()` and `exch()` to access data.

```
A C H H I M N P Q X Y K B I N A R Y
```

**Binary insertion sort.** Use binary search to find insertion point.
- Now, worst-case number of compares \( \sim n \log_2 n \).
- But still makes \( \Theta(n^2) \) array accesses in worst case.

```
A C H H I M N P Q X Y K B I N A R Y
```

*binary search for first key > K*
1.4 Analysis of Algorithms

- rules of the game
- selection sort
- insertion sort
- binary search
Binary search

Goal. Given a sorted array and a search key, find index of the search key in the array?

Binary search. Compare search key with middle entry.

• Too small, go left.
• Too big, go right.
• Equal, found.

<table>
<thead>
<tr>
<th>lo</th>
<th>hi</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>51</td>
</tr>
<tr>
<td>6</td>
<td>53</td>
</tr>
<tr>
<td>7</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>84</td>
</tr>
<tr>
<td>10</td>
<td>93</td>
</tr>
<tr>
<td>11</td>
<td>95</td>
</tr>
<tr>
<td>12</td>
<td>96</td>
</tr>
<tr>
<td>13</td>
<td>97</td>
</tr>
</tbody>
</table>
Binary search: implementation

Trivial to implement?

- First binary search published in 1946.
- First bug-free one in 1962.
- Bentley experiment: 90% of programmers implement it incorrectly.
- Bug in Java's `Arrays.binarySearch()` discovered in 2006.

Extra, Extra - Read All About It: Nearly All Binary Searches and Mergesorts are Broken

Friday, June 02, 2006

Posted by Joshua Bloch, Software Engineer

I remember vividly Jon Bentley’s first Algorithms lecture at CMU, where he asked all of us incoming Ph.D. students to write a binary search, and then dissected one of our implementations in front of the class. Of course it was broken, as were most of our implementations. This made a real impression on me, as did the treatment of this material in his wonderful Programming Pearls (Addison-Wesley, 1986, Second Edition, 2000). The key lesson was to carefully consider the invariants in your programs.

https://ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html
Binary search: implementation

**Invariant.** If key appears in array a[], then a[lo] ≤ key ≤ a[hi].

```java
public static int binarySearch(String[] a, String key) {
    int lo = 0, hi = a.length - 1;
    while (lo <= hi) {
        int mid = (lo + hi) >>> 1;
        int compare = key.compareTo(a[mid]);
        if (compare < 0) hi = mid - 1;
        else if (compare > 0) lo = mid + 1;
        else return mid;
    }
    return -1;
}
```

**Binary search: analysis**

**Proposition.** Binary search makes at most $1 + \log_2 n$ compares to search in any sorted array of length $n$.

**Pf.**

- Each iteration of `while` loop:
  - calls `compareTo()` once
  - decreases the length of remaining subarray by at least a factor of 2


can happen at most $1 + \log_2 n$ times. Why?

$n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow \cdots \rightarrow 2 \rightarrow 1$

slightly better than $2x$, due to elimination of `a[mid]` from subarray (or early termination of `while` loop)
Binary search vs. sequential search

(R. Hendricks 112) int index = 0;
(R. Hendricks 113) while (!element.equals(sortedList.get(index))
(R. Hendricks 114)     && sortedList.size() > ++index);
(R. Hendricks 115) return index < sortedList.size() ? index : -1;
3-Sum

3-SUM. Given an array of \( n \) distinct integers, count number of triples that sum to 0.

Version 0. \( \Theta(n^3) \) time in worst case. ✔
Version 1. \( \Theta(n^2 \log n) \) time in worst case.
Version 2. \( \Theta(n^2) \) time in worst case.

Note. For full credit, use only \( \Theta(1) \) extra space.
Summary

**Comparable interface.** Java framework for comparing items.

**Selection sort.** $\Theta(n^2)$ compares; $\Theta(n)$ exchanges.

**Insertion sort.** $\Theta(n^2)$ compares and exchanges in the worst case.

**Binary search.** Search a sorted array using $\Theta(\log n)$ compares in worst case.
<table>
<thead>
<tr>
<th>image/video</th>
<th>source</th>
<th>license</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting Hat</td>
<td>Hannah Hill</td>
<td>CC BY-NC 4.0</td>
</tr>
<tr>
<td>Airport Departures</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>iPhone Contacts</td>
<td>StackOverflow</td>
<td></td>
</tr>
<tr>
<td>Playing Cards</td>
<td>Google Code</td>
<td>public domain</td>
</tr>
<tr>
<td>Rock, Paper, Scissors</td>
<td>Daily Mail</td>
<td></td>
</tr>
<tr>
<td>Anime Boy</td>
<td>freesvg.org</td>
<td>public domain</td>
</tr>
<tr>
<td>Anime Girl</td>
<td>freesvg.org</td>
<td>public domain</td>
</tr>
<tr>
<td>Balance</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Jon Bentley</td>
<td>Amazon</td>
<td></td>
</tr>
<tr>
<td>Binary vs. Sequential Search</td>
<td>Silicon Valley S6E4</td>
<td></td>
</tr>
<tr>
<td>Insertion Sort Dance</td>
<td>AlgoRythmics</td>
<td></td>
</tr>
</tbody>
</table>
Insertion sort with Romanian folk dance

https://www.youtube.com/watch?v=ROaiU379I3U