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
Sorting problem

**Goal.** Rearrange array of $n$ items 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 hat (now running JDK 11)*
**Sorting problem**

**Goal.** Rearrange array of $n$ items 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>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>
Total preorder

Sorting is a well-defined problem if there is a total preorder.

A total preorder 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>Video name</th>
<th>Views (billions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Despacito&quot;[23]</td>
<td>6.96</td>
</tr>
<tr>
<td>&quot;Baby Shark Dance&quot;[28]</td>
<td>6.55</td>
</tr>
<tr>
<td>&quot;Shape of You&quot;[29]</td>
<td>4.97</td>
</tr>
<tr>
<td>&quot;See You Again&quot;[30]</td>
<td>4.72</td>
</tr>
<tr>
<td>&quot;Masha and the Bear – Recipe for&quot;</td>
<td>4.33</td>
</tr>
<tr>
<td>&quot;Uptown Funk&quot;[38]</td>
<td>3.94</td>
</tr>
</tbody>
</table>

numerical order (descending)    chronological order    lexicographic order
Total preorder

Sorting is a well-defined problem if there is a total preorder.

A total preorder 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 $v \leq x$.

Non-examples.

- course prerequisites (violates totality)
- Ro–sham–bo order (violates transitivity)
- the $\leq$ operator for double (violates totality)
Sample sort clients

**Goal.** Single function that sorts *any* type of data (that has a total preorder).

**Ex 1.** Sort strings in alphabetical order.

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

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

**Goal.** Single function that sorts *any* type of data (that has a total preorder).

**Ex 2.** Sort real numbers in ascending order.

```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]);
    }
}
```

~/Desktop/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.** Single function that sorts any type of data (that has a total preorder).

**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());
    }
}
```
How can a single function sort any type of data?

**Goal.** Single function that sorts any type of data (that has a total preorder).

**Solution.** Callback = reference to executable code.

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 total preorder.
Callbacks

**Goal.** Single function that sorts *any* type of data (that has a total preorder).

**Solution.** *Callback* = reference to executable code.

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

**Implementing callbacks.**

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

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

```java
public 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)
    {
        ...
    }
}
```

**Enforcement.** Compile-time error if a class fails to define the requisite methods.
Callbacks in Java: roadmap

client (StringSorter.java)

```java
public class StringSorter
{
    public static void main(String[] args)
    {
        String[] a = StdIn.readString().split(
        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

java.lang.Comparable interface

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

data type implementation (String.java)

```java
public class String implements Comparable<String>
{
    ... 
    public int compareTo(String that)
    {
        ...
    }
}
```
Suppose that the Java architects left out \texttt{implements Comparable<String>}

in the class declaration for \texttt{String}. What would be the effect?

\begin{itemize}
\item[A.] \texttt{String.java} won't compile.
\item[B.] \texttt{StringSorter.java} won't compile.
\item[C.] \texttt{Insertion.java} won't compile.
\item[D.] \texttt{Insertion.java} will throw an exception.
\end{itemize}
Comparable API

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

- Returns a
  - negative integer if `v` is less than `w`
  - positive integer if `v` is greater than `w`
  - zero if `v` is equal to `w`
- Induces a total preorder.
- 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;
    }
}
```

https://algs4.cs.princeton.edu/12oop/Date.java.html

can compare Date objects only to other Date objects
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- binary search
Selection sort demo

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

\begin{center}
\begin{tabular}{cccccccc}
7 & 10 & 5 & 3 & 8 & 4 & 2 & 9 & 6 \\
\end{tabular}
\end{center}

initial array
Selection sort

**Algorithm.** ↑ scans from left to right.

**Invariants.**
- Entries the left of ↑ (including ↑) fixed and in ascending order.
- No entry to right of ↑ is smaller than any entry to the left of ↑.

![Diagram showing selection sort process](image-url)
Selection sort inner loop

To maintain algorithm invariants:

- Move the pointer to the right.
  
  ```c
  i++;  
  ```

- Identify index of minimum entry on right.
  
  ```c
  int min = i;
  for (int j = i+1; j < n; j++)
      if (less(a[j], a[min]))
          min = j;
  ```

- Exchange into position.
  
  ```c
  exch(a, i, min);
  ```
Two useful sorting primitives (and a cost model)

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

use as our cost model for sorting

**Compare.** Is item v less than w?

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

polymorphic method call

**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[])
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) {
        /* 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
Selection sort: animations

20 random items

- algorithm position
- in final order
- not in final order

http://www.sorting-algorithms.com/selection-sort
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 \( (n - 1) + (n - 2) + \ldots + 1 + 0 \sim n^2 / 2 \) compares and \( n \) exchanges to sort any array of \( n \) items.

\[
\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}
\]

---

**Running time insensitive to input.** \( \Theta(n^2) \) compares, even if input is sorted.

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

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

![](initial_array.png)
Insertion sort

Algorithm. \(\uparrow\) scans from left to right.

Invariants.
- Entries to the left of \(\uparrow\) (including \(\uparrow\)) are in ascending order.
- Entries to the right of \(\uparrow\) have not yet been seen.
Insertion sort: inner loop

To maintain algorithm invariants:

- Move the pointer to the right.

```c
i++;
```

- Moving from right to left, exchange `a[i]` with each larger entry to its left.

```c
for (int j = i; j > 0; j--)
    if (less(a[j], a[j-1]))
        exch(a, j, j-1);
    else break;
```
```java
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
Elementary sorts: quiz 3

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: 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 $i$. 

$$0 + 1 + 2 + \ldots + (n - 1)$$

http://www.sorting-algorithms.com/insertion-sort
Best case. Insertion sort makes $n - 1$ compares and 0 exchanges to sort an array of $n$ distinct keys in ascending order.
**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).
Insertion sort: practical improvements

Half exchanges. Shift items over (instead of exchanging).
- Same compares but 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 can still make $\Theta(n^2)$ array accesses.

```
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 key, find index of the key in the array?

Binary search. Compare key against middle entry.

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

sorted array

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

Trivial to implement?

- First binary search published in 1946.
- First bug-free one in 1962.
- 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: Java implementation

**Invariant.** If key appears in array a[], then $a[lo] \leq key \leq a[hi]$.

```java
public static int binarySearch(String[] a, String key) {
    int lo = 0, hi = a.length - 1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        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?

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

  \[1 + \log_2 n\]

  slightly better than $2x$,
  due to elimination of $a[mid]$ from subarray
  (or early termination of `while` loop)
3-SUM. Given an array of $n$ distinct integers, count number of triples that sum to 0.

Version 0. $\Theta(n^3)$ time. ✓

Version 1. $\Theta(n^2 \log n)$ time.

Version 2. $\Theta(n^2)$ time.

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

- Step 1: Sort the \( n \) (distinct) numbers.
- Step 2: For each pair \( a[i] \) and \( a[j] \):
  
  binary search for \( -(a[i] + a[j]) \).

Analysis. Running time is \( \Theta(n^2 \log n) \).

- Step 1: \( \Theta(n^2) \) with selection sort.
- Step 2: \( \Theta(n^2 \log n) \) with binary search.

\[ \Theta(n^2) \text{ binary searches in an array of length } n \]
3-SUM. Given an array of $n$ distinct integers, find three such that $x + y + z = 0$.

Version 0. $\Theta(n^3)$ time. ✔
Version 1. $\Theta(n^2 \log n)$ time. ✔
Version 2. $\Theta(n^2)$ time. [not much harder]

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

Open research problem 1. Design algorithm that takes $\Theta(n^{1.999})$ time or better.
Open research problem 2. Prove that no $\Theta(n)$ time algorithm is possible.
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 worst case.

Binary search. Search a sorted array using $\Theta(\log n)$ compares.