https://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

https://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

Sorting problem

Goal. Given an array of n elements, rearrange in ascending order by sort key.

element →

sort key →

Last	First	House	Year
Longbottom	Neville	Gryffindor	1998
Weasley	Ron	Gryffindor	1998
Abbott	Hannah	Hufflepuff	1998
Potter	Harry	Gryffindor	1998
Chang	Cho	Ravenclaw	1997
Granger	Hermione	Gryffindor	1998
Malfoy	Draco	Slytherin	1998
Diggory	Cedric	Hufflepuff	1996
Weasley	Ginny	Gryffindor	1999
Parkinson	Pansy	Slytherin	1998

sorting hat

Sorting problem

Goal. Given an array of n elements, rearrange in ascending order by sort key.

Last ▾	First	House	Year
Abbott	Hannah	Hufflepuff	1998
Chang	Cho	Ravenclaw	1997
Granger	Hermione	Gryffindor	1998
Diggory	Cedric	Hufflepuff	1996
Longbottom	Neville	Gryffindor	1998
Malfoy	Draco	Slytherin	1998
Parkinson	Pansy	Slytherin	1998
Potter	Harry	Gryffindor	1998
Weasley	Ron	Gryffindor	1998
Weasley	Ginny	Gryffindor	1999

sort key →

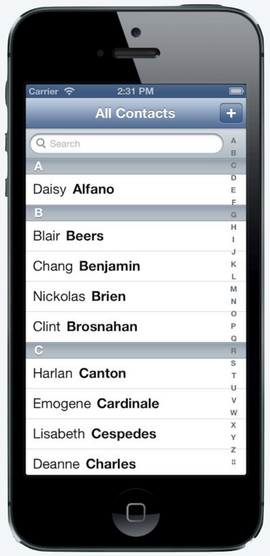
element →

↑ *sorted by key*

sorting hat

Sorting problem

Familiar examples.



alphabetical order

chronological order

Video name	Views (billions) ▲
Gangnam Style	5.46
Uptown Funk	5.48
Phonics Song	6.28
Shape of You	6.41
See You Again	6.56
Bath Song	7.00
Johny John Yes Papa	7.01
Wheels on the Bus	7.10
Despacito	8.65
Baby Shark Dance	15.59

numerical order (ascending)

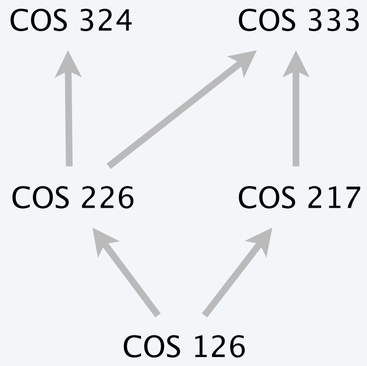
Sorting problem. Is well-defined if there is a binary relation $\preceq$ that satisfies:

- Transitivity: if both $v \preceq w$ and $w \preceq x$, then $v \preceq x$.
- Comparability: either $v \preceq w$ or $w \preceq v$ or both.

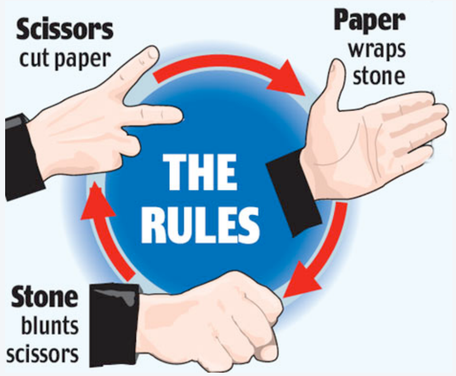
← *mathematically, a "weak order"*
(like a "total order" except can have both $v \preceq w$ and $w \preceq v$ for distinct elements v and w)

Sorting problem

Familiar non-examples.



course prerequisites
(violates comparability)



Ro-sham-bo order
(violates transitivity)

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

- Transitivity: if both $v \preceq w$ and $w \preceq x$, then $v \preceq x$.
- Comparability: either $v \preceq w$ or $w \preceq v$ or both.

Sample sort clients

Goal. General-purpose sorting function.

Ex 1. Sort strings in alphabetical order. ⟵ *lexicographic order (Unicode)*

```
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]);
   }
}
```

```
~/cos226/sort> more words3.txt
BED BUG 💖🦪🦪 DAD YET ZOO … ALL BAD YES

~/cos226/sort> java-algs4 StringSorter < words3.txt
ALL BAD BED BUG DAD … YES YET ZOO 💖🦪🦪
[suppressing newlines]
```

Unicode	value
⋮	⋮
A	65
B	66
C	67
D	68
⋮	⋮
Θ	1,012
⋮	⋮
💖	128,150
⋮	⋮

Unicode character ordering

Sample sort clients

Goal. General-purpose sorting function.

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

```
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-algs4 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

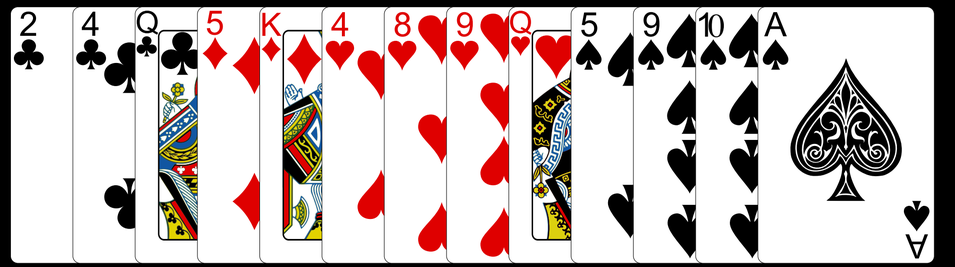
Goal. General-purpose sorting function.

Ex 3. Sort playing cards in suit-major order.

```
public class HandOfCards {
   ...

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

```
~/cos226/sort> java-algs4 HandOfCards 13
```



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 a function and later executed.

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

← *effectively, client passes* `compareTo()` *method to* `sort()` *function; the callback occurs when* `sort()` *invokes* `compareTo()`

Implementing callbacks.

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

Java interfaces

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

interface (java.lang.Comparable)

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

contract: method with this signature (and prescribed behavior)

Class that implements interface. Must implement all interface methods.

```
public class String implements Comparable<String> {
   ...

   public int compareTo(String that) {
      ...
   }

}
```

class promises to honor the contract

class abides by the contract

Java interfaces: properties

Subtype polymorphism.

- Interfaces are reference types.
- A class that implements an interface is a subtype of that interface.

e.g., can use an object of the subtype instead of interface type in assignment statements, method arguments, return values, …

Dynamic dispatch. Java determines which interface method to call using the type of the referenced object at runtime.

Q. Why useful?

A. Enables callbacks.

- Design a single method that can sort strings, integers, or dates.
- Iterate over a collection without knowing the underlying representation.
- Intercept and process mouse clicks in a Java app.
- …

```
// can assign String object
// to variable of type Comparable
Comparable x = "Hello";
Comparable y = "World";

// calls String compareTo()
int result1 = x.compareTo(y);

// can also assign Date object
// to variable of type Comparable
x = new Date(2025, 02, 11);
y = new Date(1969, 07, 16);

// calls Date compareTo()
int result2 = x.compareTo(y);
```

Callbacks in Java: roadmap

client (StringSorter.java)

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

interface (Comparable.java)

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

`String[]` *is a subtype of* `Comparable[]`

sort implementation (Insertion.java)

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

callback

data type implementation (String.java)

```
public class String implements Comparable<String> {
   ...

   public int compareTo(String that) {
      ...
   }

}
```

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

Elementary sorts: poll 1

Suppose that the Java architects left out the clause `implements Comparable<String>` in the class declaration for `String`. What would be the consequence?

A. Compile-time error in `String.java`.

B. Compile-time error in `StringSorter.java`.

C. Compile-time error in `Insertion.java`.

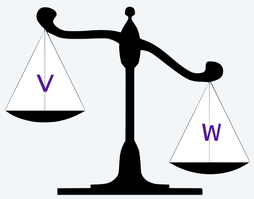
D. Run-time exception in `Insertion.java`.

Comparable API

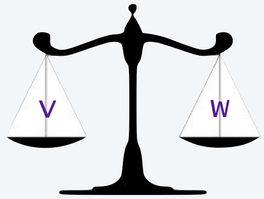
Requirement. Implement `compareTo()` so that `v.compareTo(w)`

- 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 `null`).

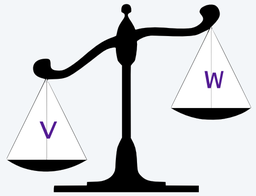
API requirement: the binary relation `v.compareTo(w) <= 0` *is a weak order*



v is less than w
(return negative integer)



v is equal to w
(return 0)



v is greater than w
(return positive integer)

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

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

Implementing the Comparable interface

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

```
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;
   }

}
```

can compare `Date` *objects only to other* `Date` *objects*

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

https://algs4.cs.princeton.edu

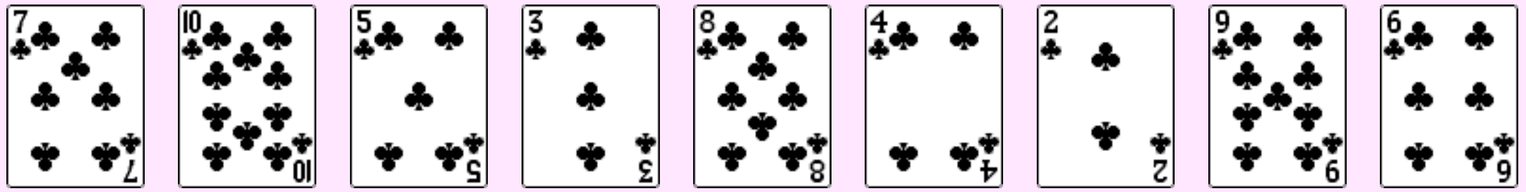
2.1 Elementary Sorts

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

Selection sort demo

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

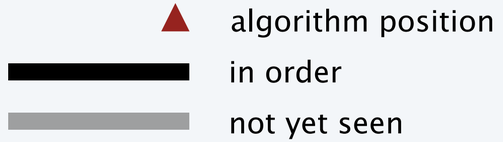
- Find index min of smallest remaining element.
- Swap elements at indices i and min.



initial array

Selection sort: visualization

Visualization. Sort vertical bars by length.



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

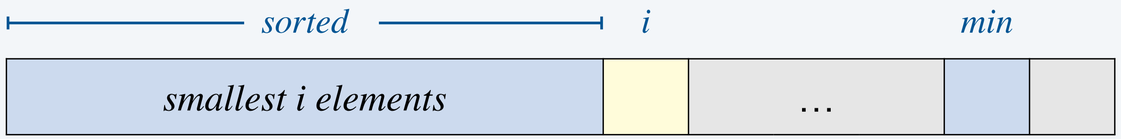
Selection sort invariants

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

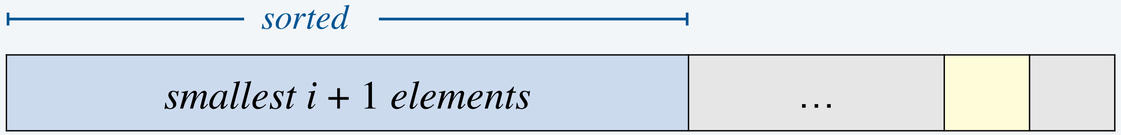
- Find index *min* of smallest remaining element.
- Swap elements at indices i and *min*.

Invariants. ← *logical conditions that hold repeatedly at well-defined steps of an algorithm (or program)*

at start of iteration i



at end of iteration i



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 element `v` strictly less than element `w` ?

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

← `less("aardvark", "zebra")` *returns* `true`

dynamic dispatch: Java calls the object's `compareTo()` *method*

use interface type as argument: can call `less()` *with any class that implements the* `Comparable` *interface*

Exchange. Swap array entries `a[i]` and `a[j]`.

```
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[]`)

← 1 *exchange makes* 4 *array accesses*

Selection sort: Java implementation

```
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

Elementary sorts: poll 2

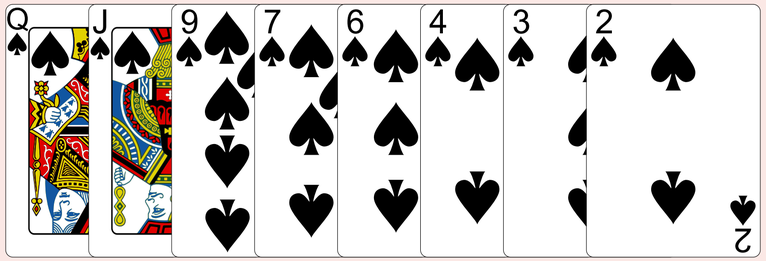
How many compares to selection sort an array of n distinct elements in reverse order?

A. $\sim n$

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

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

D. $\sim n^2$



$n = 8$

Selection sort: mathematical analysis

Proposition. Selection sort makes $\sim \frac{1}{2} n^2$ compares and n exchanges to sort any array of n elements.

Pf. Exactly $(n - i - 1)$ compares and 1 exchange in iteration i.

$(n-1) + (n-2) + \ldots + 1 + 0 \sim \frac{1}{2} n^2$

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
0	6	S	O	R	T	E	X	A	M	P	L	E
1	4	A	O	R	T	E	X	S	M	P	L	E
2	10	A	E	R	T	O	X	S	M	P	L	E
3	9	A	E	E	T	O	X	S	M	P	L	R
4	7	A	E	E	L	O	X	S	M	P	T	R
5	7	A	E	E	L	M	X	S	O	P	T	R
6	8	A	E	E	L	M	O	S	X	P	T	R
7	10	A	E	E	L	M	O	P	X	S	T	R
8	8	A	E	E	L	M	O	P	R	S	T	X
9	9	A	E	E	L	M	O	P	R	S	T	X
10	10	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

entries in black are examined to find the minimum

entries in red are a[min]

entries in gray are in final position

Running time insensitive to input. Makes $\Theta(n^2)$ compares. ← *even if input array is sorted*

Data movement is minimal. Makes $\Theta(n)$ exchanges.

In place. Uses $\Theta(1)$ extra space.

https://algs4.cs.princeton.edu

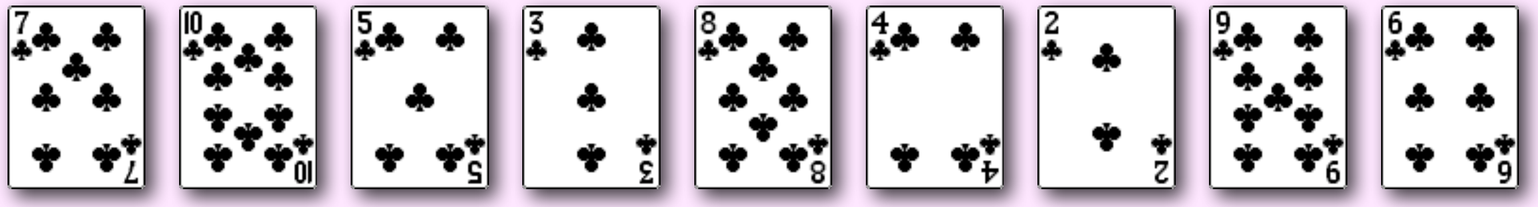
2.1 Elementary Sorts

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

Insertion sort demo

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.



initial array

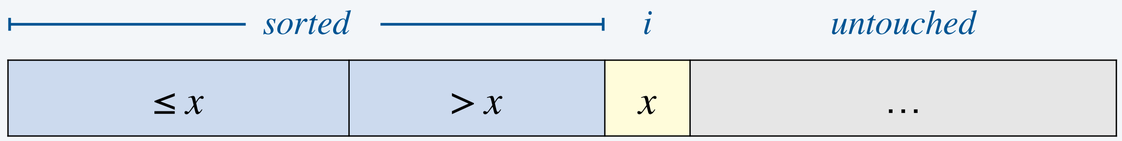
Insertion sort invariants

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

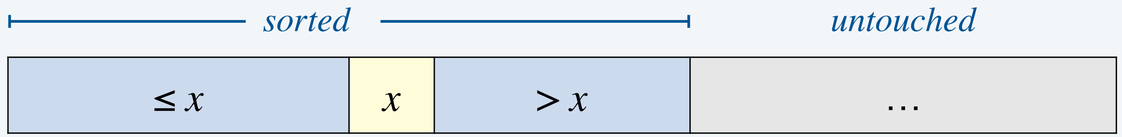
- Let x be the element at index i.
- Repeatedly exchange x with each larger element to its immediate left.

Invariants.

at start of iteration i



at end of iteration i



Insertion sort: Java implementation

```
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: poll 4

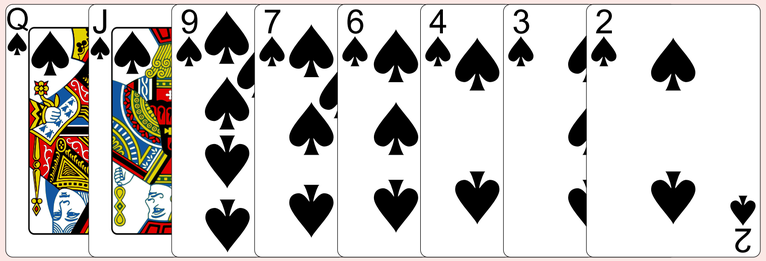
How many compares to insertion sort an array of n distinct elements in reverse order?

A. $\sim n$

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

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

D. $\sim n^2$



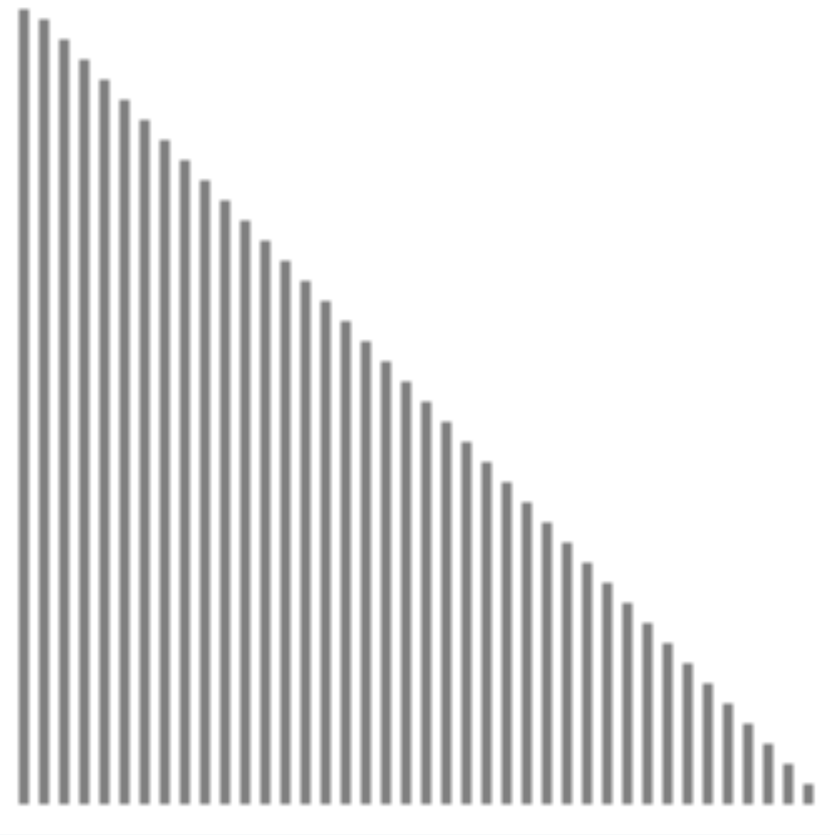
$n = 8$

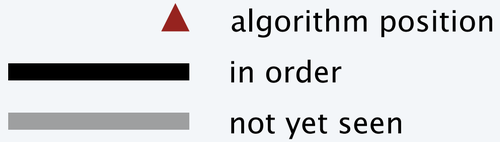
Insertion sort: running time analysis

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 elements in reverse order.

Pf. Exactly i compares and exchanges in iteration i.

$$0 + 1 + \ldots + (n-2) + (n-1) \sim \frac{1}{2} n^2$$



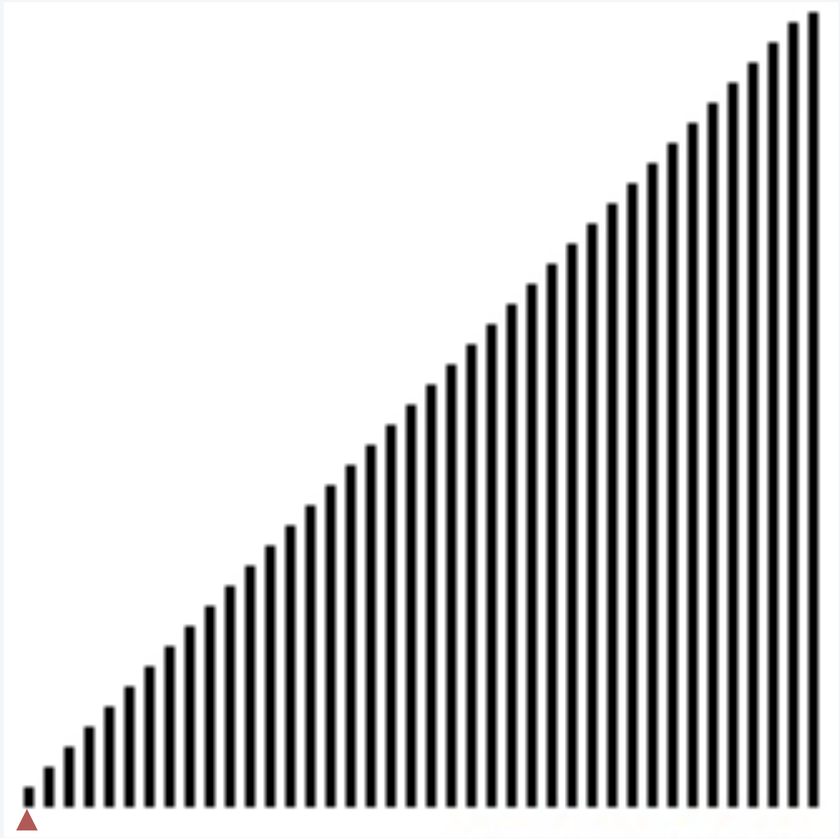


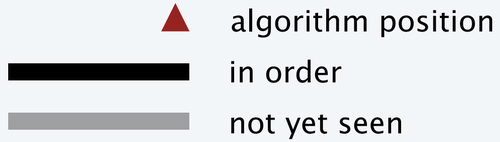
https://www.toptal.com/developers/sorting-algorithms/insertion-sort

Insertion sort: running time analysis

Best case. Insertion sort makes $n - 1$ compares and 0 exchanges to sort an array of n distinct elements in ascending order.

Pf. Exactly 1 compares and 0 exchanges in each iteration (except first).





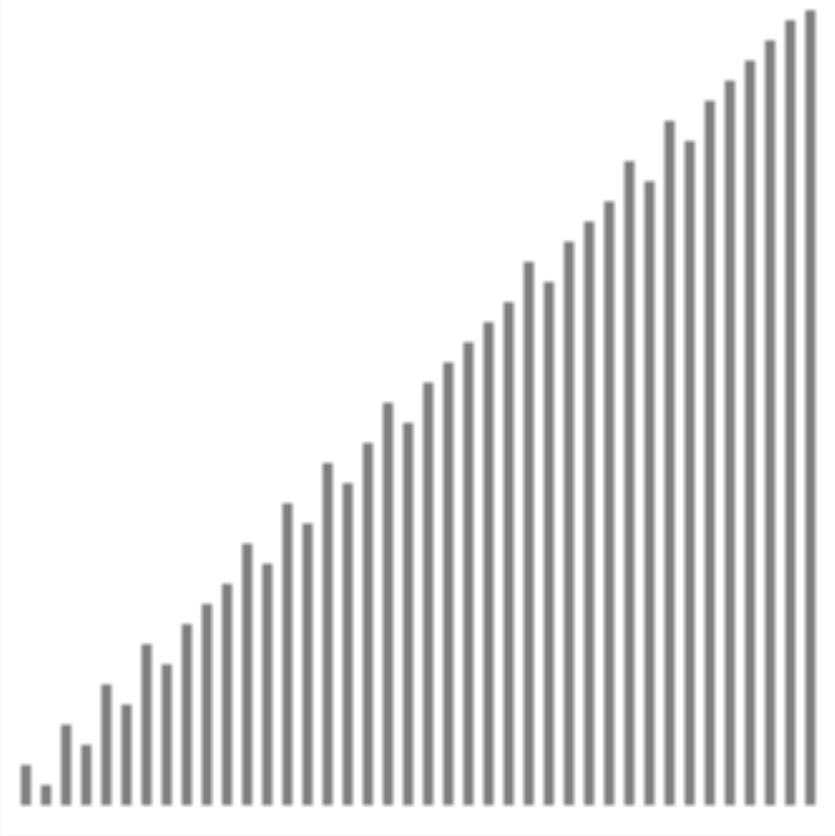
https://www.toptal.com/developers/sorting-algorithms/insertion-sort

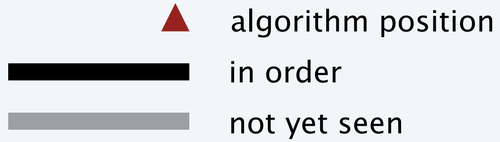
Insertion sort: running time analysis

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).



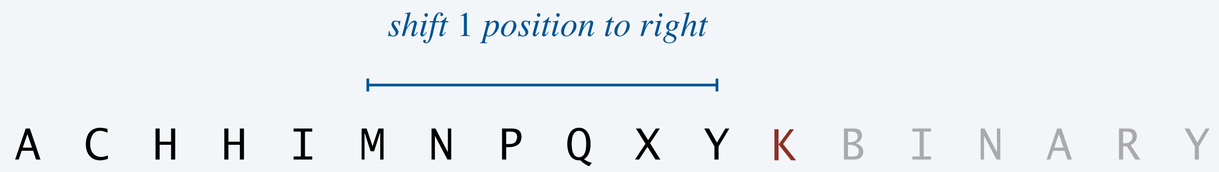


https://www.toptal.com/developers/sorting-algorithms/insertion-sort

Insertion sort: practical improvements

Half exchanges. Shift elements over (instead of exchanging).

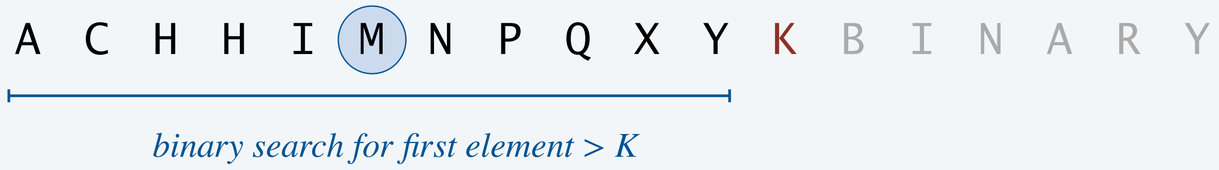
- Same compares; fewer array accesses.
- No longer uses only `less()` and `exch()` to access data.



Binary insertion sort. Use binary search to locate insertion point.

- Now, worst-case number of compares is $\sim n \log_2 n$.
- But worst-case number of array access is still $\Theta(n^2)$.

compares can be very expensive (relative to data movement), especially in interpreted languages (such as Python)



https://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

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

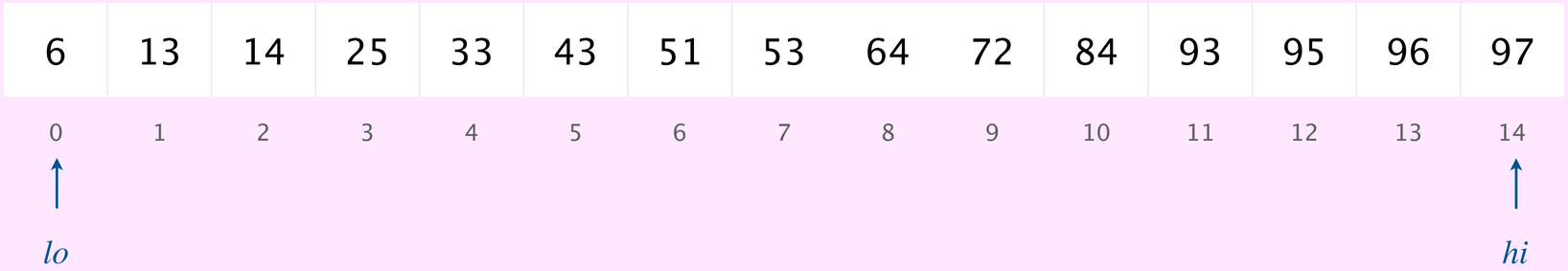
Binary search

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 element.

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

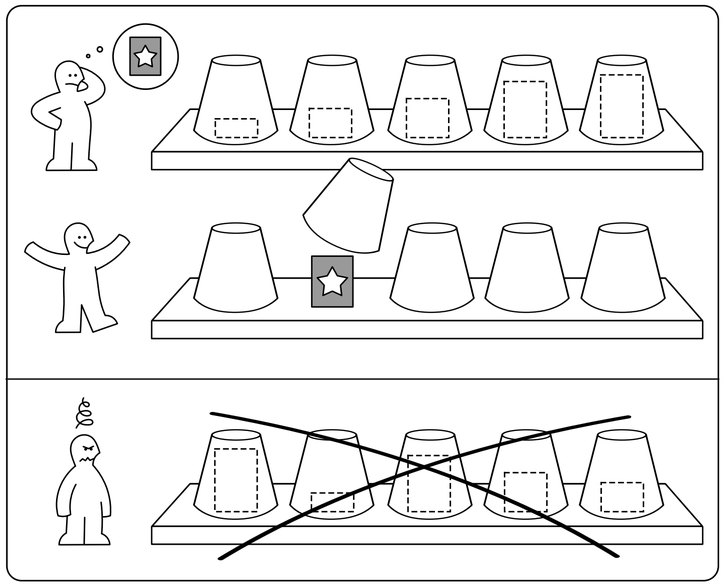
sorted array



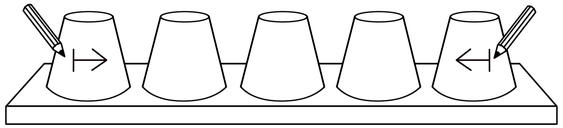
Binary search: nonverbal algorithm assembly instructions

BINÄRY SEARCH

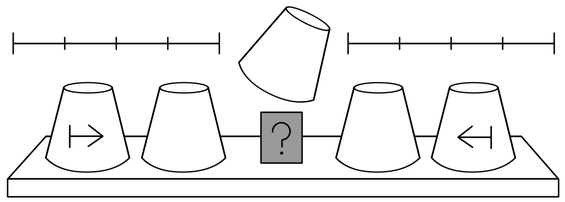
idea-instructions.com/binary-search/
v1.1, CC by-nc-sa 4.0



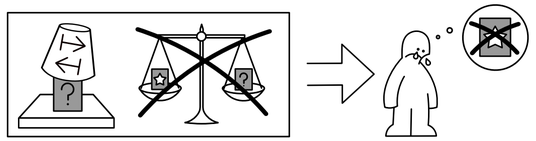
1



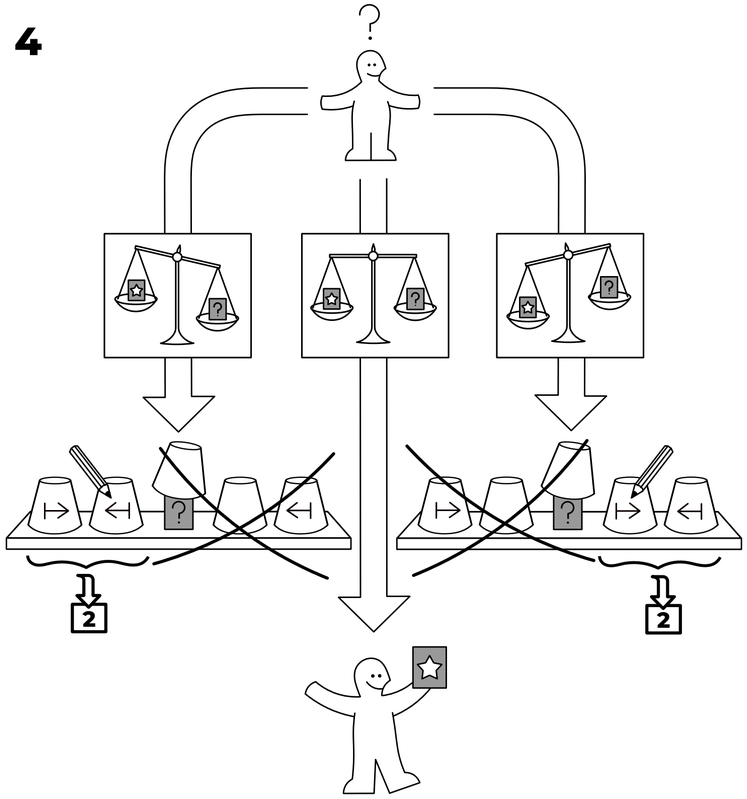
2



3



4



Binary search: implementation

Trivial to implement?

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

and in C, C++, …

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]`.

```
// precondition: a[] is sorted
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;
}
```

why not `mid = (lo + hi) / 2` ?

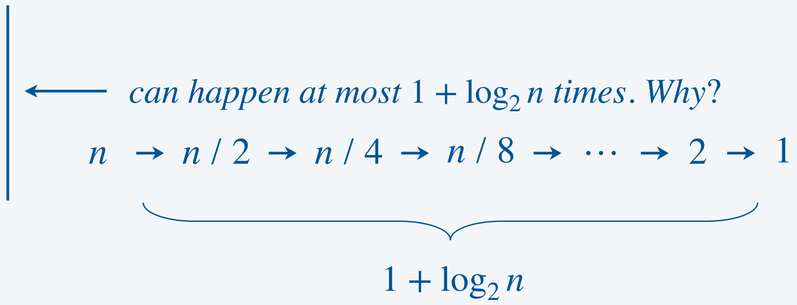
https://algs4.cs.princeton.edu/11model/BinarySearch.java.html

Binary search: analysis

Proposition. Binary search makes at most $1 + \log_2 n$ calls to `compareTo()` to search in any sorted array of length n.

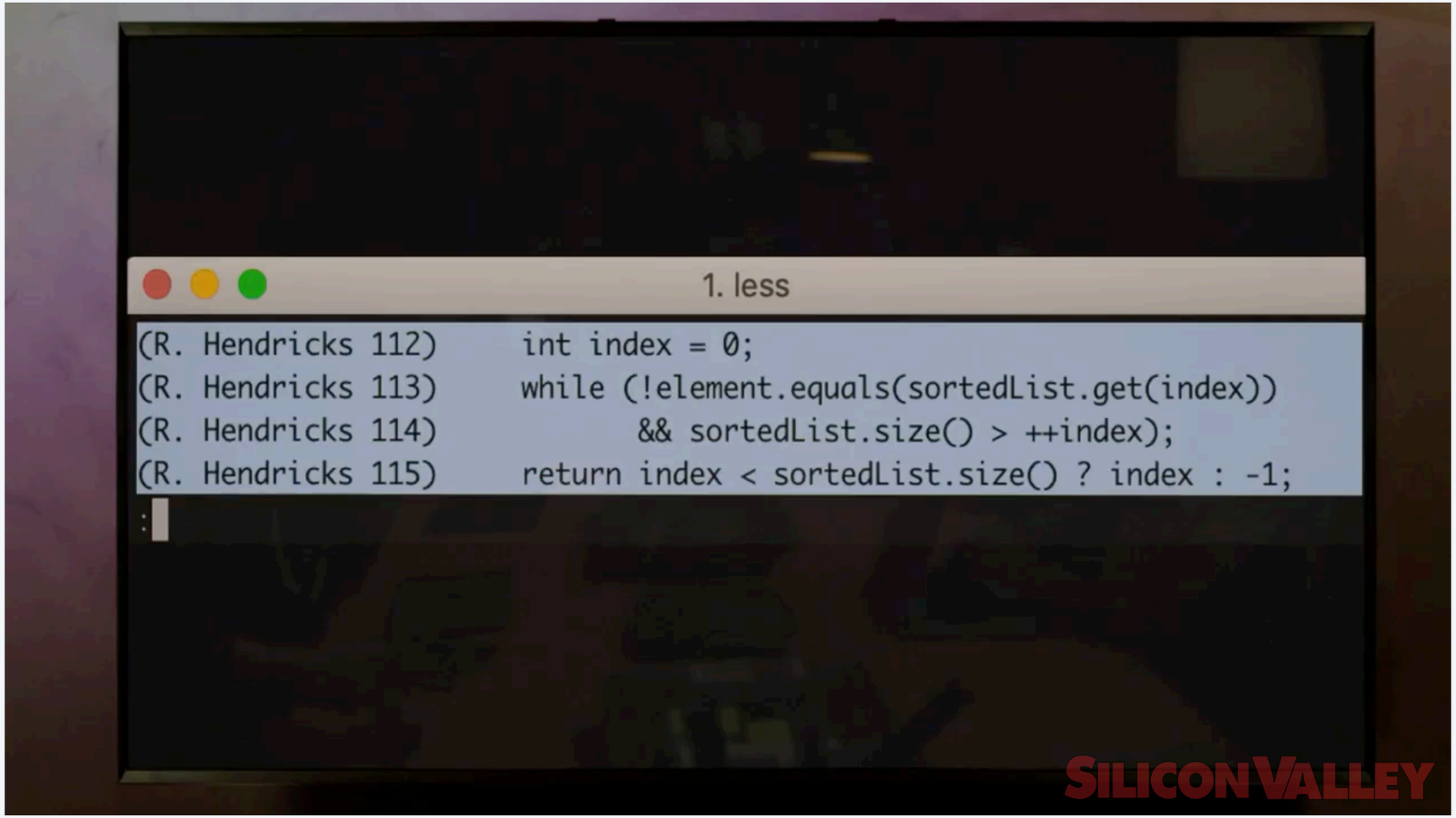
Pf.

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



slightly better than 2×, *due to elimination of* `a[mid]` *from subarray (or early termination of* `while` *loop)*

Binary search vs. sequential search



3-Sum

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

Version 0. Takes $\Theta(n^3)$ time in worst case. ✅

Version 1. Takes $\Theta(n^2 \log n)$ time in worst case.

Version 2. Takes $\Theta(n^2)$ time in worst case.

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

3-Sum: a $\Theta(n^2 \log n)$ algorithm

Algorithm.

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

Analysis. Running time is $\Theta(n^2 \log n)$ in worst case.

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

$\Theta(n^2)$ *binary searches in an array of length n*

input array a[]

30 -40 -20 -10 40 0 10 5

sorted array a[]

-40 -20 -10 0 5 10 30 40

binary search

i	j	a[i]	a[j]	x
0	1	-40	-20	60
0	2	-40	-10	50
0	3	-40	0	40
0	4	-40	5	35
0	5	-40	10	30
⋮	⋮	⋮	⋮	⋮
1	2	-20	-10	30
⋮	⋮	⋮	⋮	⋮
2	3	-10	0	10
⋮	⋮	⋮	⋮	⋮
5	6	10	30	~~-40~~
5	7	10	40	-50
6	7	30	40	-70

count only if $i < j < k$ to avoid both triple counting and $10 + 10 + -20$

3-Sum

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

Version 0. Takes $\Theta(n^3)$ time in worst case. ✅

Version 1. Takes $\Theta(n^2 \log n)$ time in worst case. ✅

Version 2. Takes $\Theta(n^2)$ time in worst case. [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

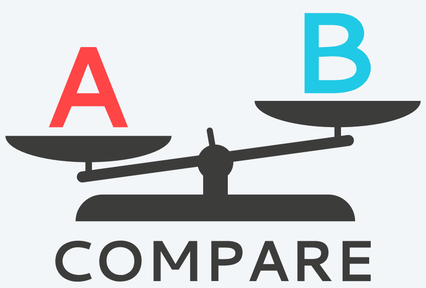
Comparable interface. Java framework for comparing elements.

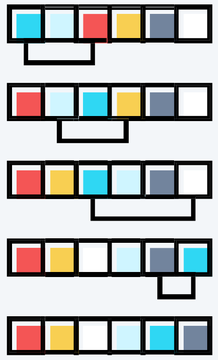
Selection sort. Makes $\Theta(n^2)$ compares; $\Theta(n)$ exchanges.

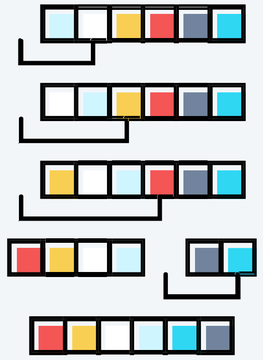
Insertion sort. Makes $\Theta(n^2)$ compares and exchanges in the worst case.

Binary insertion sort. Makes $\Theta(n \log n)$ compares; $\Theta(n^2)$ exchanges in the worst case.

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







Credits

media	source	license
Sorting Hat	Hannah Hill	CC BY-NC 4.0
Airport Departures	Adobe Stock	education license
iPhone Contacts	StackOverflow	
Playing Cards	Google Code	public domain
Rock, Paper, Scissors	Daily Mail	
Anime Boy	freesvg.org	public domain
Anime Girl	freesvg.org	public domain
Call Back Icon		
Balance	Adobe Stock	education license
Selection Sort Visualization	Toptal	
Insertion Sort Visualization	Toptal	

Credits

media	source	license
Jon Bentley	Amazon	
Binary Search IDEA	idea-instructions.com	CC BY-NC-SA 4.0
Binary vs. Sequential Search	Silicon Valley S6E4	
Compare A and B	Adobe Stock	education license
Selection Sort Graphic	Adobe Stock	education license
Insertion Sort Graphic	Adobe Stock	education license
Insertion Sort Dance	AlgoRythmics	

Insertion sort with Romanian folk dance

https://www.youtube.com/watch?v=ROalU379l3U