https://algs4.cs.princeton.edu

3.1 SYMBOL TABLES

- *API*
- *elementary implementations*
- *ordered operations*

https://algs4.cs.princeton.edu

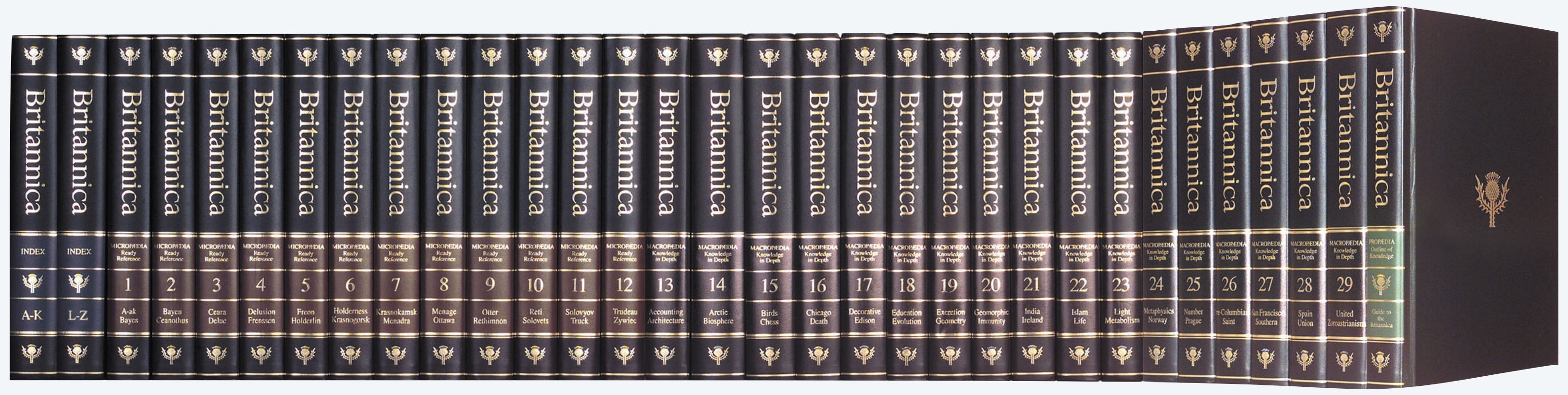
3.1 Symbol Tables

- *API*
- *elementary implementations*
- *ordered operations*

Why are telephone books (and their cousins) obsolete?

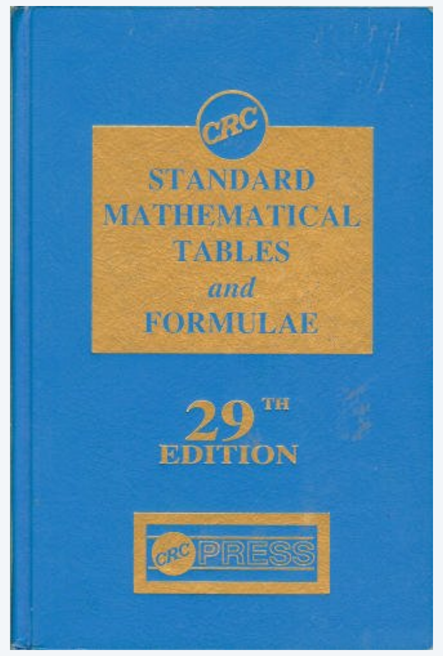
Unsupported phone book operations.

- Add a new name and associated number.
- Remove a given name and associated number.
- Change the number associated with a given name.

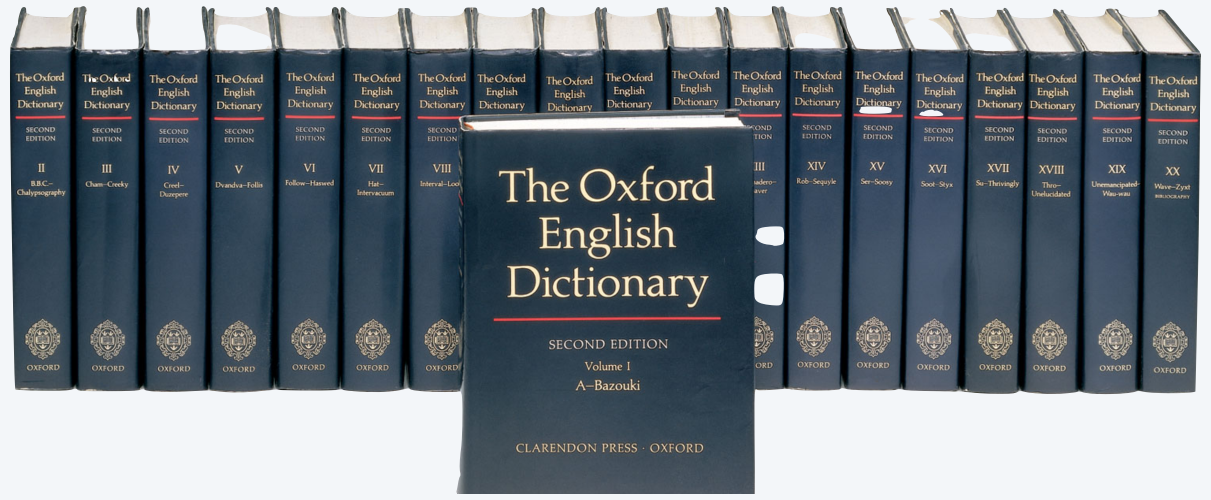


key = term
value = article

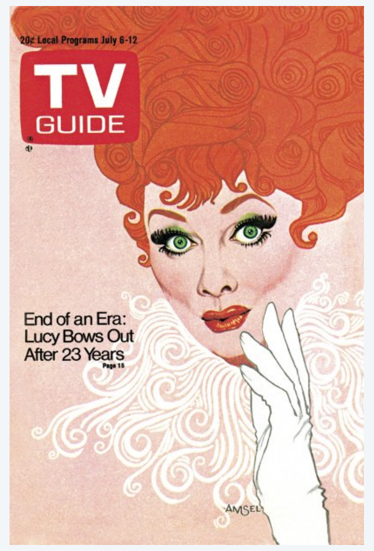
key = name
value = phone number



key = math function and input
value = function output



key = word
value = definition



key = time and channel
value = TV show

Symbol tables

Key-value pair abstraction.

- Insert a value with specified key.
- Given a key, search for the corresponding value.

Ex. DNS lookup.

- Insert domain name with specified IP address.
- Given domain name, find corresponding IP address.

domain name	IP address
www.cs.princeton.edu	128.112.136.61
goprincetontigers.com	67.192.28.17
wikipedia.com	208.80.153.232
google.com	172.217.11.46

↑ *key* ↑ *value*

Symbol table applications

application	purpose of search	key	value
dictionary	*find definition*	word	definition
book index	*find relevant pages*	term	list of page numbers
file share	*find song to download*	name of song	computer ID
financial account	*process transactions*	account number	transaction details
web search	*find relevant web pages*	keyword	list of page names
compiler	*find properties of variables*	variable name	type and value
routing table	*route Internet packets*	destination	best route
DNS	*find IP address*	domain name	IP address
reverse DNS	*find domain name*	IP address	domain name
genomics	*find markers*	DNA string	known positions
file system	*find file on disk*	filename	location on disk

Symbol tables: context

Also known as: maps, dictionaries, associative arrays.

Generalizes arrays. Keys need not be integers between 0 and $n - 1$.

Language support.

- External libraries: C, VisualBasic, Standard ML, bash, …
- Built-in libraries: Java, C#, C++, Scala, …
- Built-in to language: Awk, Perl, PHP, Tcl, JavaScript, Python, Ruby, Lua, …

```
has_nice_syntax_for_dictionaries['Python'] = True
has_nice_syntax_for_dictionaries['Java']   = False
```

legal Python code

Basic symbol table API

Associative array abstraction. Associate key-value pairs.

two generic type parameters

```
public class ST<Key extends Comparable<Key>, Value>
```

	`ST()`	*create an empty symbol table*	
`void`	`put(Key key, Value val)`	*insert key–value pair*	← `a[key] = val;`
`Value`	`get(Key key)`	*value paired with key*	← `a[key]`
`Iterable<Key>`	`keys()`	*all the keys in the symbol table*	
`boolean`	`contains(Key key)`	*is there a value paired with key?*	
`void`	`delete(Key key)`	*remove key (and associated value)*	
`boolean`	`isEmpty()`	*is the symbol table empty?*	
`int`	`size()`	*number of key–value pairs*	

Conventions

- Method `put()` overwrites old value with new value.
- Method `get()` returns `null` if key not present.
- Values are not `null`. ← `java.util.Map` *allows* `null` *values*

> *“ Careless use of null can cause a staggering variety of bugs. Studying the Google code base, we found that something like 95% of collections weren’t supposed to have any null values in them, and having those fail fast rather than silently accept null would have been helpful to developers.”*

GUAVA™

https://code.google.com/p/guava-libraries/wiki/UsingAndAvoidingNullExplained

Key and value types

Value type. Any generic type.

specify `Comparable` *in API*

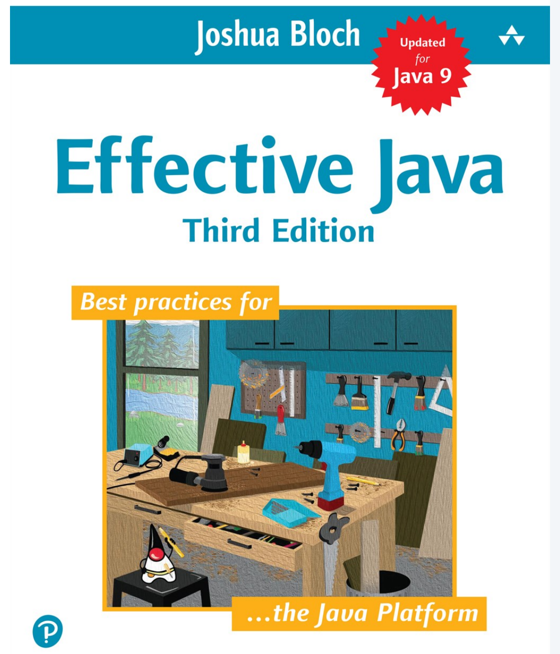
Key type: different assumptions.

- This lecture: keys are `Comparable`; use `compareTo()`.
- Hashing lecture: keys are any generic type;
  use `equals()` to test equality and `hashCode()` to scramble key.

Best practices. Use immutable types for symbol-table keys.

- Immutable in Java: `String`, `Integer`, `Double`, `Color`, …
- Mutable in Java: `StringBuilder`, `Stack`, `URL`, arrays, …

> " *Classes should be immutable unless there's a very good reason to make them mutable.… If a class cannot be made immutable, you should still limit its mutability as much as possible.* "
>
> — Joshua Bloch (Java Collections architect)



ST test client for analysis

Frequency counter. Read a sequence of strings from standard input; print one that occurs most often.

```
~/cos226/st> more tinyTale.txt
it was the best of times
it was the worst of times
it was the age of wisdom
it was the age of foolishness
it was the epoch of belief
it was the epoch of incredulity
it was the season of light
it was the season of darkness
it was the spring of hope
it was the winter of despair

~/cos226/st> java FrequencyCounter 3 < tinyTale.txt      ← tiny example (60 words, 20 distinct)
the 10

~/cos226/st> java FrequencyCounter 8 < tale.txt          ← real example (135,635 words, 10,769 distinct)
business 10

~/cos226/st> java FrequencyCounter 10 < leipzig1M.txt    ← real example (21,191,455 words, 534,580 distinct)
government 24763
```

Frequency counter implementation

```
public class FrequencyCounter {
   public static void main(String[] args) {
      int minLength = Integer.parseInt(args[0]);
```

compute frequencies

```
      ST<String, Integer> st = new ST<>();
      while (!StdIn.isEmpty()) {
         String word = StdIn.readString();
         if (word.length() < minLength) continue;
         if (!st.contains(word)) st.put(word, 1);
         else                    st.put(word, st.get(word) + 1);
      }
```

identify and print a string with max frequency

```
      String champ = "";
      st.put(champ, 0);
      for (String word : st.keys()) {
         if (st.get(word) > st.get(champ))
            champ = word;
      }
      StdOut.println(champ + " " + st.get(champ));
   }
}
```

https://algs4.cs.princeton.edu

3.1 Symbol Tables

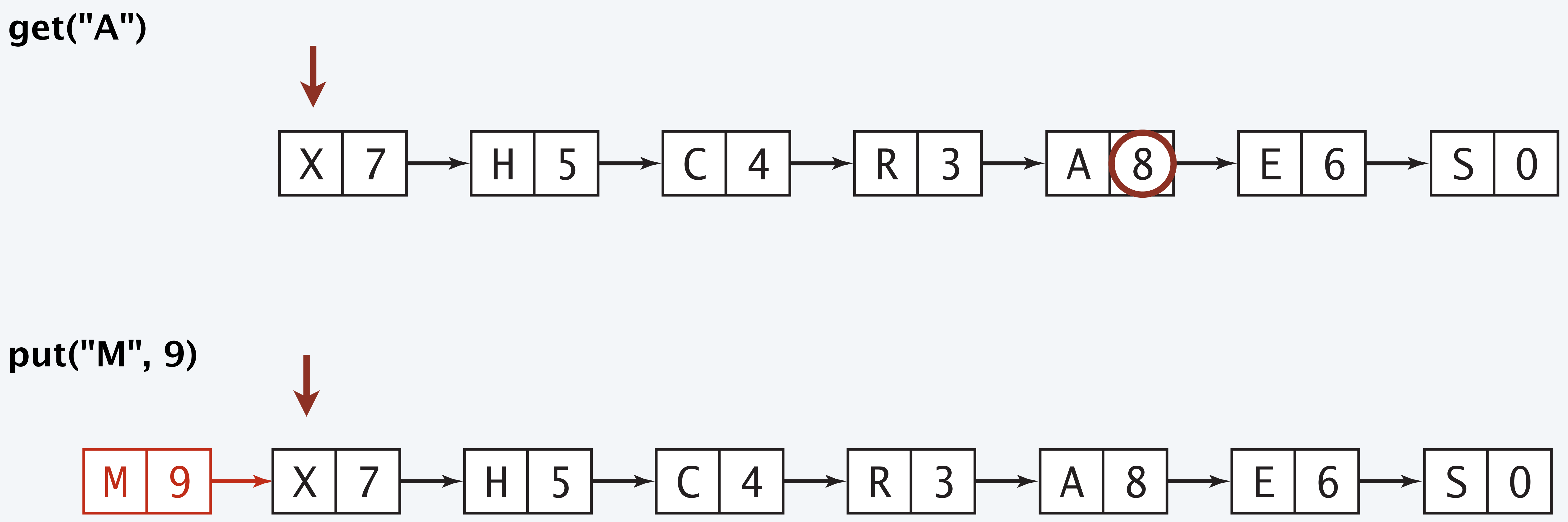
- API
- elementary implementations
- ordered operations

Sequential search in a linked list

Data structure. Maintain an (unordered) linked list of key-value pairs.

Search. Scan through all keys until finding a match.

Insert. Scan through all keys until finding a match; if no match add to front.



Proposition. In the worst case, search and insert take $\Theta(n)$ time.

Elementary symbol tables: quiz 1

Data structure. Maintain parallel arrays for keys and values, sorted by key.

keys[]

0	1	2	3	4	5	6	7	8	9
A	C	E	H	L	M	P	R	S	Z

vals[]

0	1	2	3	4	5	6	7	8	9
8	4	2	5	11	9	10	3	0	7

What are the worst-case running times for search and insert?

A. $\Theta(\log n)$ and $\Theta(\log n)$

B. $\Theta(n)$ and $\Theta(\log n)$

C. $\Theta(\log n)$ and $\Theta(n)$

D. $\Theta(n)$ and $\Theta(n)$

Binary search in a sorted array

Data structure. Maintain parallel arrays for keys and values, sorted by key.

Search. Use binary search to find key.

Insert. Use binary search to find place to insert; shift all larger keys over.

get("P")

keys[]

0	1	2	3	4	5	6	7	8	9
A	C	E	H	L	M	P	R	S	Z

vals[]

0	1	2	3	4	5	6	7	8	9
8	4	2	5	11	9	10	3	0	7

put("P", 10)

keys[]

0	1	2	3	4	5	6	7	8	9
A	C	E	H	M	R	S	X	-	-

vals[]

0	1	2	3	4	5	6	7	8	9
8	4	6	5	9	3	0	7	-	-

Elementary symbol tables: quiz 2

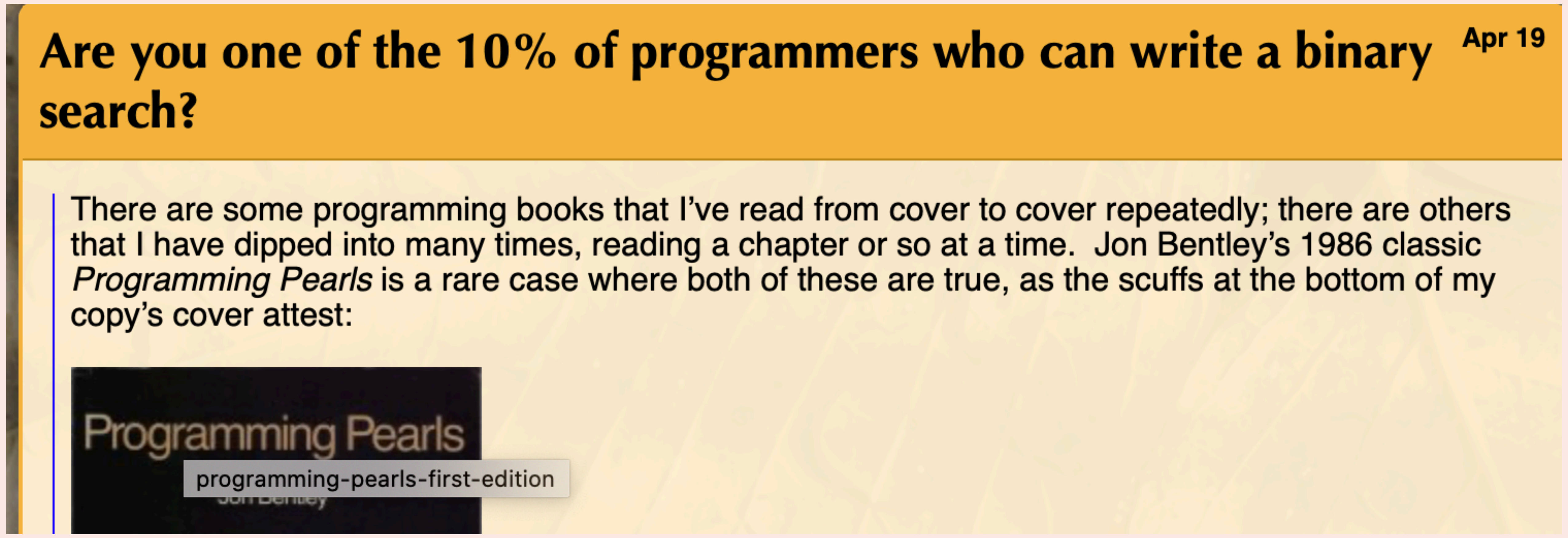
When I first submitted `BinarySearchDeluxe.java` to TigerFile, the autograder identified a …

A. Correctness bug (false positive or false negative).

B. Performance bug (or infinite loop).

C. Both A and B.

D. Neither A nor B.



Are you one of the 10% of programmers who can write a binary search? Apr 19

There are some programming books that I've read from cover to cover repeatedly; there are others that I have dipped into many times, reading a chapter or so at a time. Jon Bentley's 1986 classic *Programming Pearls* is a rare case where both of these are true, as the scuffs at the bottom of my copy's cover attest:

Elementary ST implementations: summary

implementation	guarantee		operations on keys
	search	insert	
sequential search (unordered list)	n	n	`equals()`
binary search (sorted array)	$\log n$	n †	`compareTo()`

† *can do with* $\Theta(\log n)$ *compares, but still requires* $\Theta(n)$ *array accesses*

Challenge. Efficient implementations of both search and insert.

https://algs4.cs.princeton.edu

3.1 Symbol Tables

- API
- elementary implementations
- ordered operations

Examples of ordered symbol table API

	keys	values	
min() →	9:00:00	Chicago	
	9:00:03	Phoenix	
	9:00:13	Houston	← get(9:00:13)
	9:00:59	Chicago	
	9:01:10	Houston	
floor(9:05:00) →	9:03:13	Chicago	
	9:10:11	Seattle	
select(7) →	9:10:25	Seattle	
rank(9:10:25) = 7	9:14:25	Phoenix	
	9:19:32	Chicago	
	9:19:46	Chicago	
	9:21:05	Chicago	
	9:22:43	Seattle	
	9:22:54	Seattle	
	9:25:52	Chicago	
ceiling(9:30:00) →	9:35:21	Chicago	
	9:36:14	Seattle	
max() →	9:37:44	Phoenix	

Ordered symbol table API

Symbol table API. Add these ordered operations when keys are *Comparable*.

```
public class ST<Key extends Comparable<Key>, Value>
```

	⋮	
`Key`	`min()`	*smallest key*
`Key`	`max()`	*largest key*
`Key`	`floor(Key key)`	*largest key less than or equal to key*
`Key`	`ceiling(Key key)`	*smallest key greater than or equal to key*
`int`	`rank(Key key)`	*number of keys less than key*
`Key`	`select(int k)`	*key of rank k*

Rank in a sorted array

Problem. Given a sorted array of n distinct keys,

find the number of keys strictly less than a given query key.

Q. What if duplicate keys are allowed?

Ordered symbol table operations: summary

	sequential search	binary search	goal
search	n	$\log n$	$\log n$
insert	n	n	$\log n$
min / max	n	1	$\log n$
floor / ceiling	n	$\log n$	$\log n$
rank	n	$\log n$	$\log n$
select	n	1	$\log n$

order of growth of worst-case running time for ordered symbol table operations

Challenge. Efficient implementations of all operations.

Credits

image	source	license
Encyclopedias	Encyclopædia Britannica	
Stack of Phone Books	James Joyner	
CRC Standard Mathematical Tables	CRC Press	
Oxford English Dictionary	Oxford University Press	
TV Guide	TV Guide Magazine	
Effective Java	Addison-Wesley Professional	
Binary Search	The Reinvigorated Programmer	