http://algs4.cs.princeton.edu

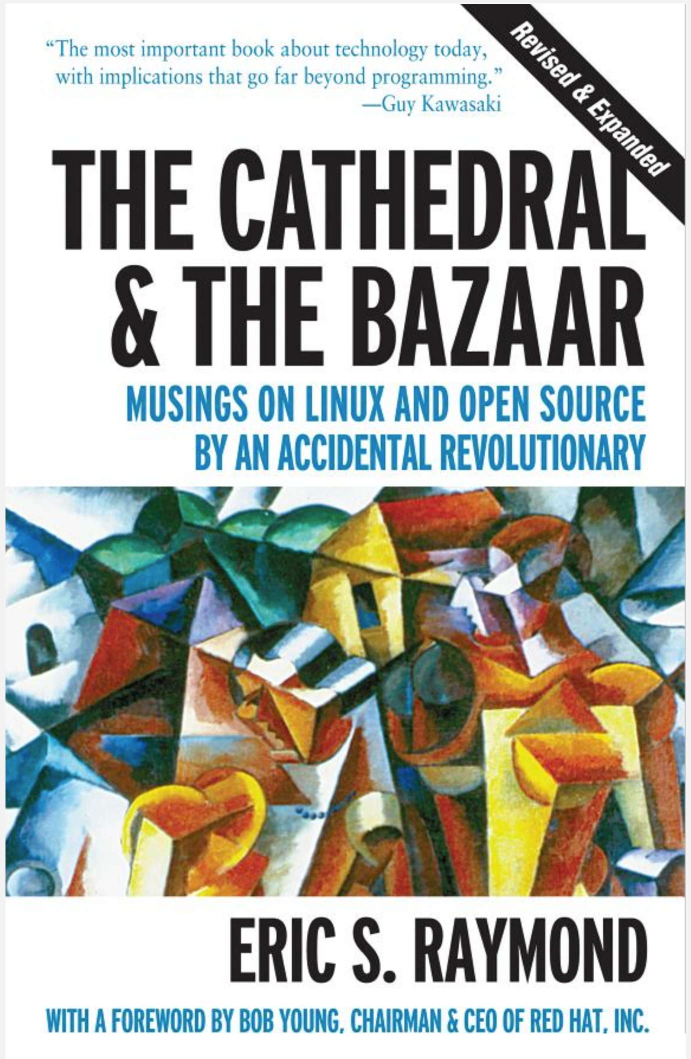
3.1 SYMBOL TABLES

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

Last updated on 10/10/17 7:05 AM

Data structures

> " *Smart data structures and dumb code works a lot better than the other way around.* " – *Eric S. Raymond*


“The most important book about technology today, with implications that go far beyond programming.” —Guy Kawasaki

Revised & Expanded

THE CATHEDRAL & THE BAZAAR

MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY

ERIC S. RAYMOND

WITH A FOREWORD BY BOB YOUNG, CHAIRMAN & CEO OF RED HAT, INC.

http://algs4.cs.princeton.edu

3.1 SYMBOL TABLES

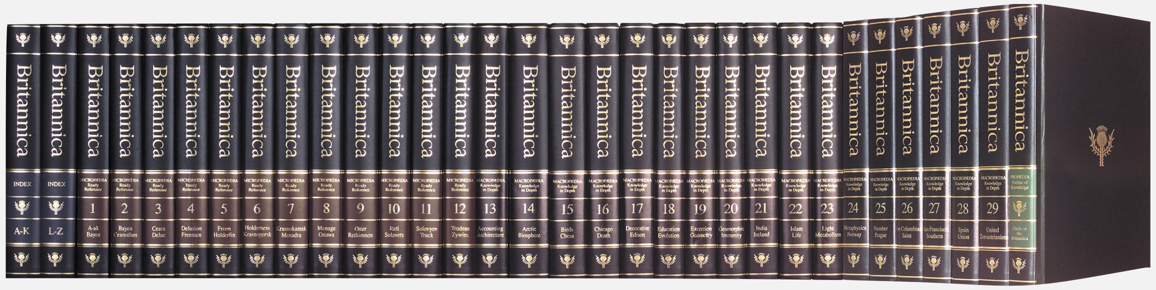
- *API*
- *elementary implementations*
- *ordered operations*

Why are telephone books obsolete?

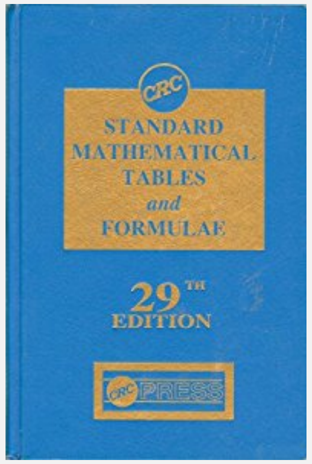
Unsupported operations.

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

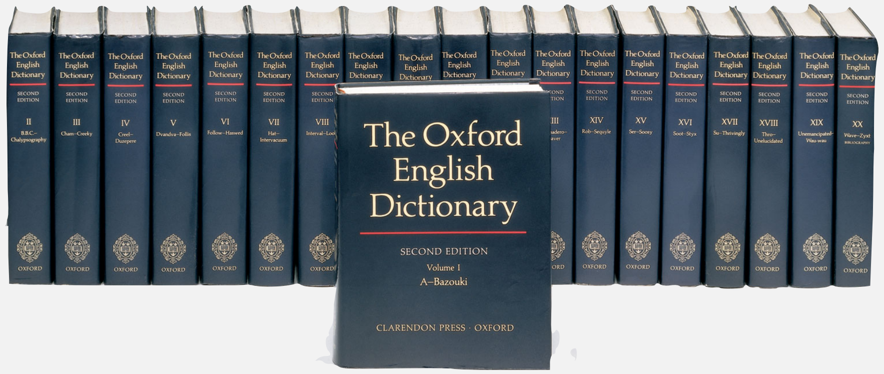
key = name
value = phone number



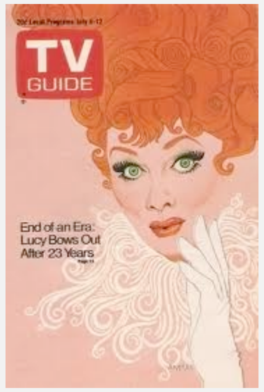
key = term
value = article



key = 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	128.112.136.11
www.princeton.edu	128.112.128.15
www.yale.edu	130.132.143.21
www.harvard.edu	128.103.060.55

↑ 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 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.

every array is an associative array

every object is an associative array

table is the only "primitive" data structure

```
has_nice_syntax_for_associative_arrays["Python"] = True
has_nice_syntax_for_associative_arrays["Java"]   = False
```

legal Python code

Basic symbol table API

Associative array abstraction. Associate one value with each key.

`public class ST<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]`**
`boolean`	`contains(Key key)`	*is there a value paired with key?*	
`Iterable<Key>`	`keys()`	*all the keys in the symbol table*	
`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 `get()` returns `null` if key not present.
- Method `put()` overwrites old value with new value.
- 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-libraries

Guava: Google Core Libraries for Java 1.6+

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

Conventions

- Method `get()` returns `null` if key not present.
- Method `put()` overwrites old value with new value.
- Values are not `null`.

Intended consequences.

- Easy to implement `contains()`.

```
public boolean contains(Key key)
{  return get(key) != null;  }
```

- Can implement lazy version of `delete()`.

```
public void delete(Key key)
{  put(key, null);  }
```

Keys and values

Value type. Any generic type.

specify `Comparable` in API.

Key type: several natural assumptions.

- Assume keys are `Comparable`, use `compareTo()`.
- Assume keys are any generic type, use `equals()` to test equality.
- Assume keys are any generic type, use `equals()` to test equality; use `hashCode()` to scramble key.

built-in to Java
(stay tuned)

Best practices. Use immutable types for symbol table keys.

- Immutable in Java: `Integer`, `Double`, `String`, `java.io.File`, …
- Mutable in Java: `StringBuilder`, `java.net.URL`, arrays, ...

Equality test

All Java classes inherit a method `equals()`.

Java requirements. For any references `x`, `y` and `z`:

- Reflexive: `x.equals(x)` is `true`.
- Symmetric: `x.equals(y)` iff `y.equals(x)`.
- Transitive: if `x.equals(y)` and `y.equals(z)`, then `x.equals(z)`.
- Non-null: `x.equals(null)` is `false`.

equivalence relation

do x and y refer to the same object?

Default implementation. `(x == y)`

Customized implementations. `Integer`, `Double`, `String`, `java.io.File`, …

User-defined implementations. Some care needed.

Implementing equals for user-defined types

Seems easy.

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

   public boolean equals(Date that)
   {








      if (this.day   != that.day  ) return false;
      if (this.month != that.month) return false;
      if (this.year  != that.year ) return false;
      return true;
   }
}
```

check that all significant fields are the same

Implementing equals for user-defined types

Seems easy, but requires some care.

typically unsafe to use equals() with inheritance (would violate symmetry)

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

   public boolean equals(Object y)
   {
      if (y == this) return true;

      if (y == null) return false;

      if (y.getClass() != this.getClass())
         return false;

      Date that = (Date) y;
      if (this.day   != that.day  ) return false;
      if (this.month != that.month) return false;
      if (this.year  != that.year ) return false;
      return true;
   }
}
```

must be Object.
Why? Experts still debate.

optimization (for reference equality)

check for null

objects must be in the same class
(religion: getClass() vs. instanceof)

cast is now guaranteed to succeed

check that all significant
fields are the same

Equals design

"Standard" recipe for user-defined types.

- Optimization for reference equality.
- Check against `null`.
- Check that two objects are of the same type; cast.
- Compare each significant field:
  - if field is a primitive type, use ==
  - if field is an object, use `equals()`
  - if field is an array, apply to each entry

but use `Double.compare()` for double (to deal with –0.0 and NaN)

apply rule recursively

can use `Arrays.deepEquals(a, b)` but not `a.equals(b)`

Best practices.

e.g., cached Manhattan distance

- No need to use calculated fields that depend on other fields.
- Compare fields mostly likely to differ first.
- Make `compareTo()` consistent with `equals()`.

`x.equals(y)` if and only if `(x.compareTo(y) == 0)`

ST test client for analysis

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

```
% 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

% java FrequencyCounter 3 < tinyTale.txt
the 10

% java FrequencyCounter 8 < tale.txt
business 122

% java FrequencyCounter 10 < leipzig1M.txt
government 24763
```

tiny example
(60 words, 20 distinct)

real example
(135,635 words, 10,769 distinct)

real example
(21,191,455 words, 534,580 distinct)

Frequency counter implementation

```
public class FrequencyCounter
{
   public static void main(String[] args)
   {
      int minLength = Integer.parseInt(args[0]);
      ST<String, Integer> st = new ST<String, Integer>();
      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);
      }
      String max = "";
      st.put(max, 0);
      for (String word : st.keys())
         if (st.get(word) > st.get(max))
            max = word;
      StdOut.println(max + " " + st.get(max));
   }
}
```

create ST

ignore short strings

read string and
update frequency

print a string with max frequency

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

3.1 Symbol Tables

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

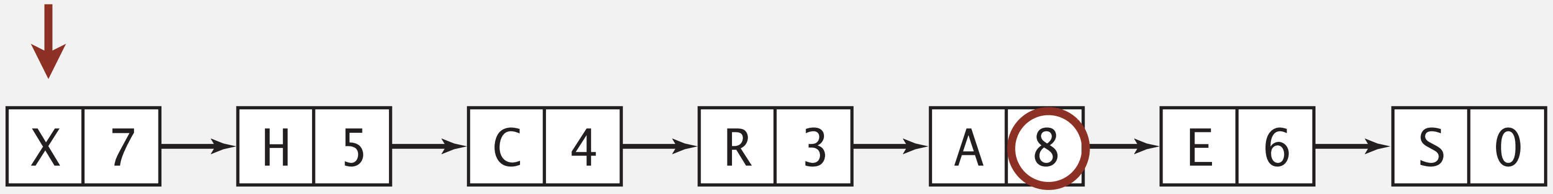
Sequential search in a linked list

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

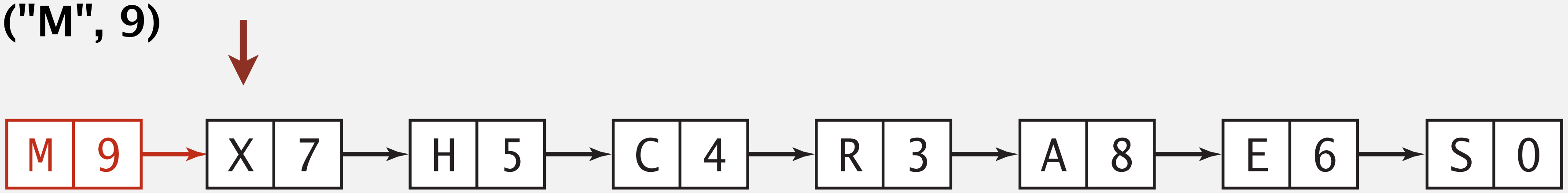
Search. Scan through all keys until find a match.

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

get("A")



put("M", 9)



Elementary ST implementations: summary

implementation	guarantee		average case		operations on keys
	search	insert	search hit	insert	
sequential search (unordered list)	n	n	n	n	`equals()`

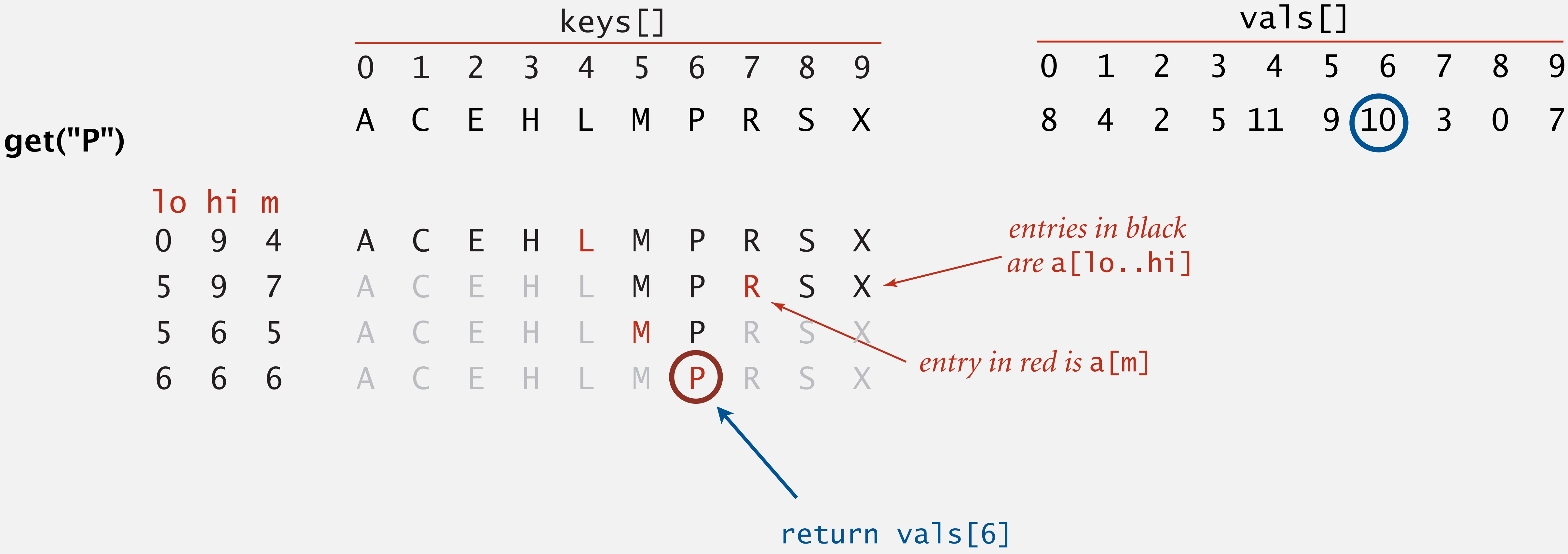
Challenge. Efficient implementations of both search and insert.

Binary search in an ordered array

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

Search. Use binary search to find key.

Proposition. At most $\sim \lg n$ compares to search a sorted array of length n.



keys[]

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

vals[]

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

get("P")

lo	hi	m										
0	9	4	A	C	E	H	L	M	P	R	S	X
5	9	7	A	C	E	H	L	M	P	R	S	X
5	6	5	A	C	E	H	L	M	P	R	S	X
6	6	6	A	C	E	H	L	M	P	R	S	X

entries in black are `a[lo..hi]`

entry in red is `a[m]`

`return vals[6]`

Binary search in an ordered array

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

Search. Use binary search to find key.

```
 public Value get(Key key)
 {
    int lo = 0, hi = n-1;
    while (lo <= hi)
    {
        int mid = lo + (hi - lo) / 2;
        int cmp = key.compareTo(keys[mid]);
        if      (cmp  < 0) hi = mid - 1;
        else if (cmp  > 0) lo = mid + 1;
        else if (cmp == 0) return vals[mid];
   }
   return null;
}
```

← no matching key

Elementary symbol tables: quiz 1

Implementing binary search was

- **A.** Much easier than I thought.
- **B.** Easier than I thought.
- **C.** About what I expected.
- **D.** Harder than I thought.
- **E.** Much harder than I thought.

FIND THE FIRST 1

Problem. Given an array with all 0s in the beginning and all 1s at the end, with more 1s thanks 0s, find the index in the array where the 1s begin.

input	0	0	0	0	…	0	0	0	1	1	1	1	1	1	…	1	1	1	1

Binary search: insert

Data structure. Maintain an ordered array of key–value pairs.

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

Proposition. Takes linear time in the worst case.

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 ST implementations: summary

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

† can do with log n compares, but requires n array accesses

Challenge. Efficient implementations of both search and insert.

http://algs4.cs.princeton.edu

3.1 SYMBOL TABLES

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

Examples of ordered symbol table API

keys	*values*
09:00:00	Chicago
09:00:03	Phoenix
09:00:13	Houston
09:00:59	Chicago
09:01:10	Houston
09:03:13	Chicago
09:10:11	Seattle
09:10:25	Seattle
09:14:25	Phoenix
09:19:32	Chicago
09:19:46	Chicago
09:21:05	Chicago
09:22:43	Seattle
09:22:54	Seattle
09:25:52	Chicago
09:35:21	Chicago
09:36:14	Seattle
09:37:44	Phoenix

`min()`

`get(09:00:13)`

`floor(09:05:00)`

`select(7)`

`keys(09:15:00, 09:25:00)`

`ceiling(09:30:00)`

`max()`

`size(09:15:00, 09:25:00)` *is* 5

`rank(09:10:25)` *is* 7

Ordered symbol table API

`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.

Binary search: ordered symbol table operations summary

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

order of growth of the running time for ordered symbol table operations