
https://algs4.cs.princeton.edu

# 3.1 Symbol Tables

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


Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

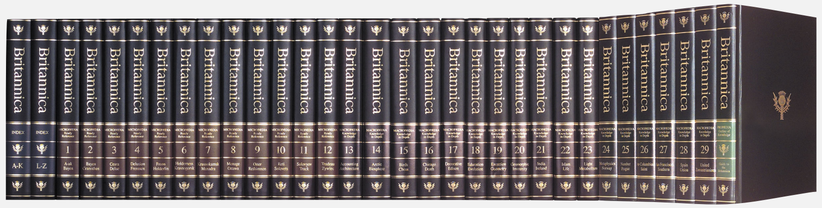
# 3.1 Symbol Tables

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

# Why are telephone books obsolete?

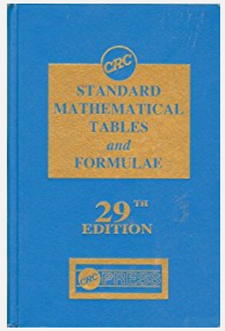
## Unsupported operations.

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

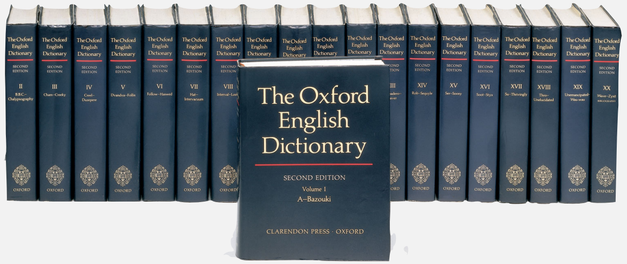



key = term
value = article


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

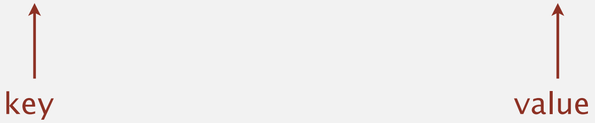
Key–value pair abstraction.

- 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 |



# 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]` |
| `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 `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

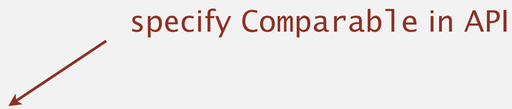
> *“ Careless use of null can cause a staggering variety of bugs. Studying the Google code base, we found that something like 95% of collections weren’t supposed to have any null values in them, and having those fail fast rather than silently accept null would have been helpful to developers.”*


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

# Key and value types

Value type. Any generic type.

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, ...

# 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]);

      compute frequencies
      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);
      print a string with max frequency
      for (String word : st.keys())
         if (st.get(word) > st.get(max))
            max = word;
      StdOut.println(max + " " + st.get(max));
   }
}
```

create ST

read string and update frequency

iterate over key–value pairs


https://algs4.cs.princeton.edu

# 3.1 Symbol Tables

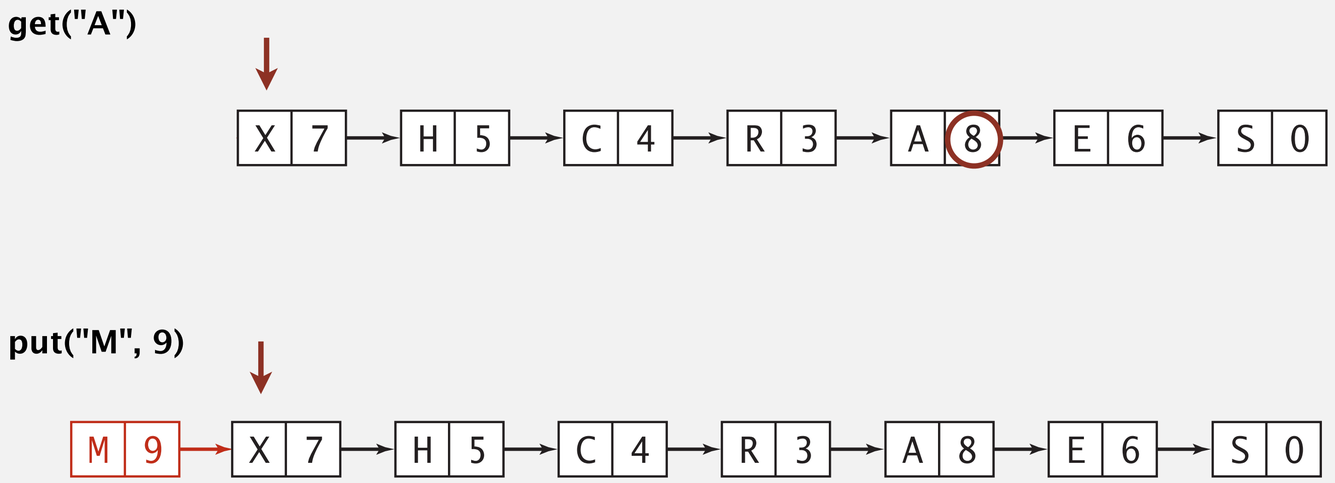
- *API*
- *elementary implementations*
- *ordered operations*

# Sequential search in a linked list

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

Search. Scan through all keys until find a match.

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



# 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 a sorted array

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

Search. Use binary search to find key.

Proposition. At most $1 + \log_2 n$ compares to search a sorted array of length $n$.

**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 |

# 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.

```
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 return vals[mid];
   }
   return null; <------ no matching key
}
```

# Binary search: insert

Data structure. Maintain a sorted array of key–value pairs.

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

Proposition. Takes $\Theta(n)$ 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 (sorted array)** | $\log n$ | $n$ † | $\log n$ | $n$ † | `compareTo()` |

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

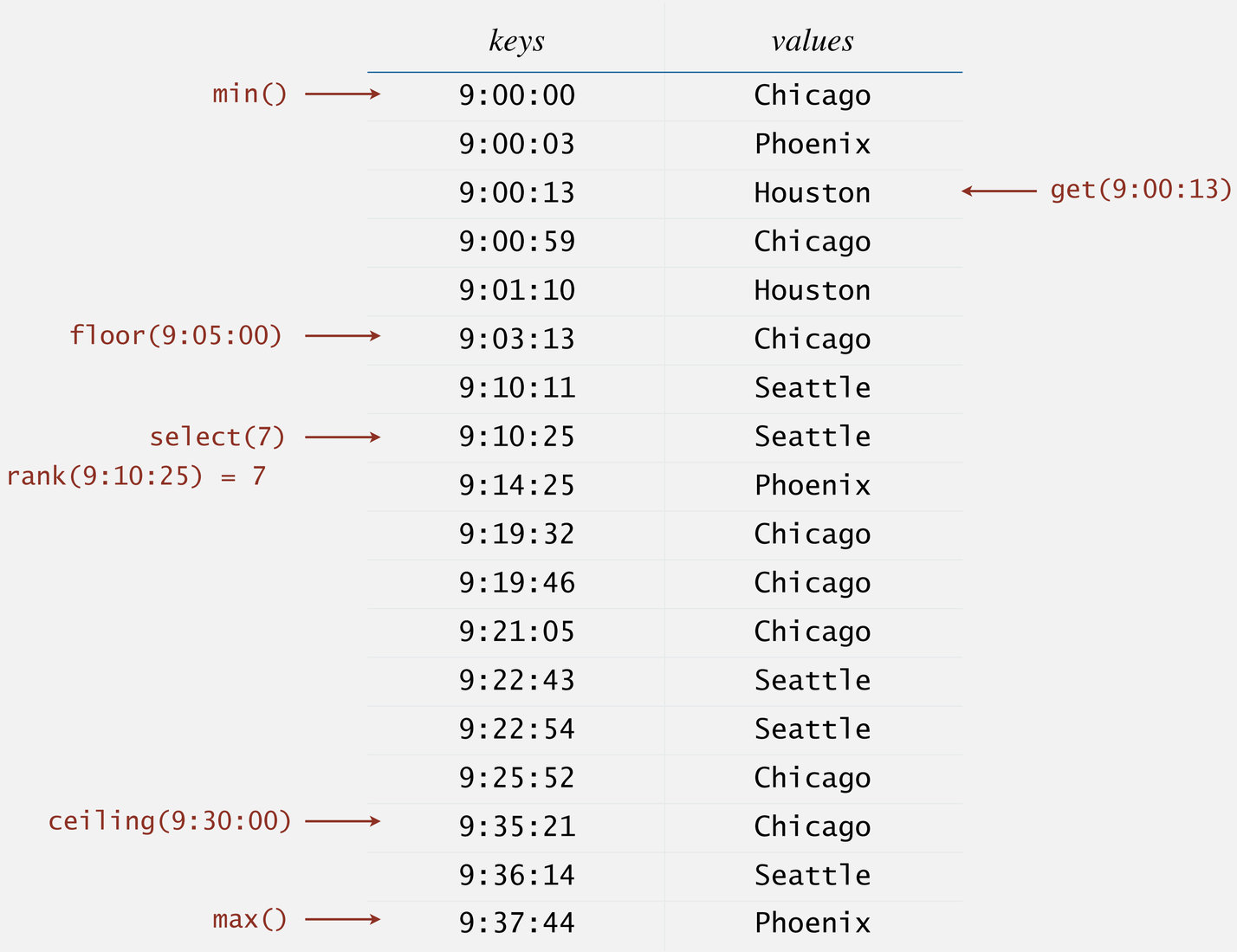
Challenge. Efficient implementations of both search and insert.


https://algs4.cs.princeton.edu

# 3.1 Symbol Tables

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

# Examples of ordered symbol table API



# 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.

# 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

Challenge. Efficient implementations of all operations.