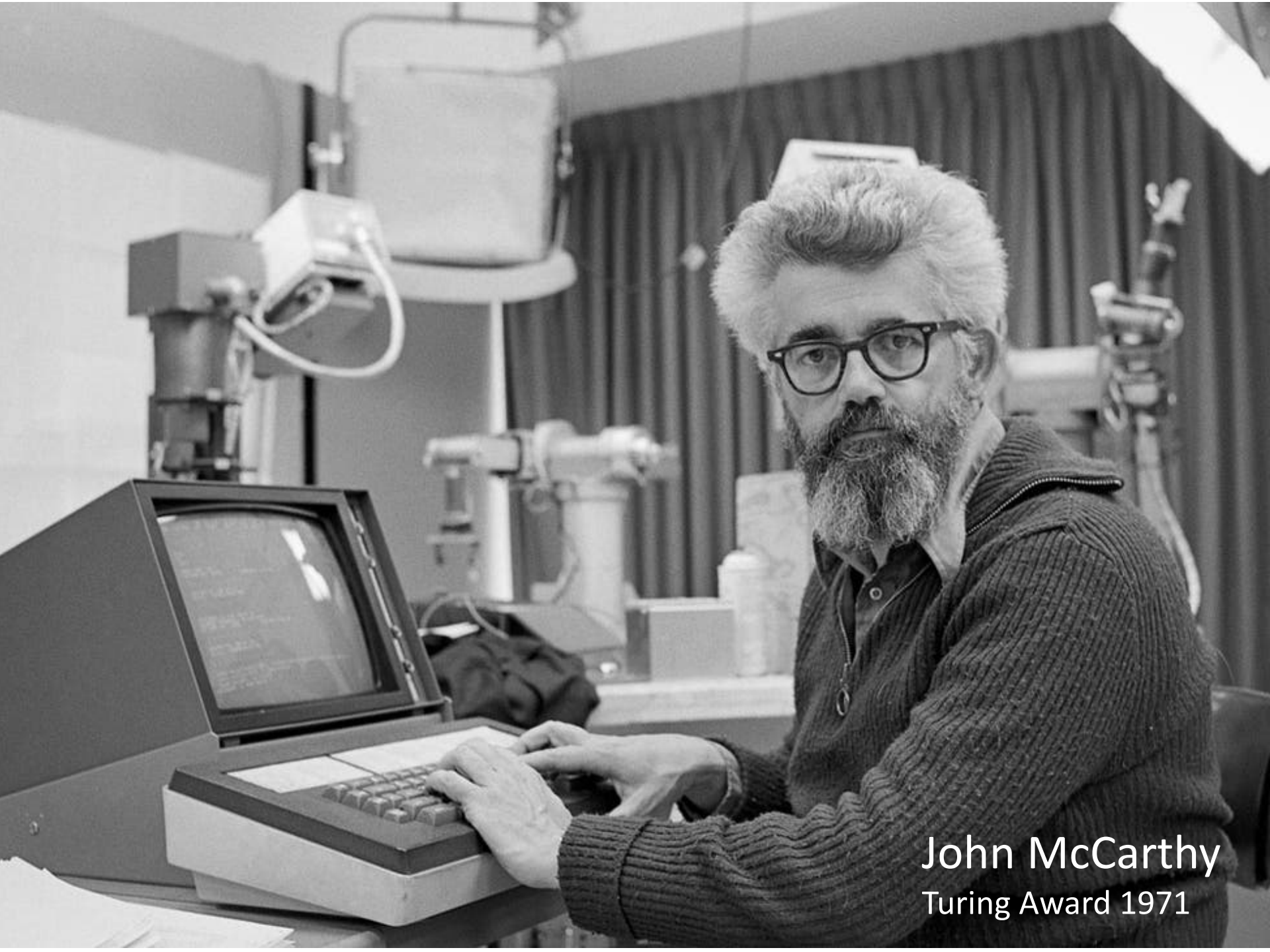
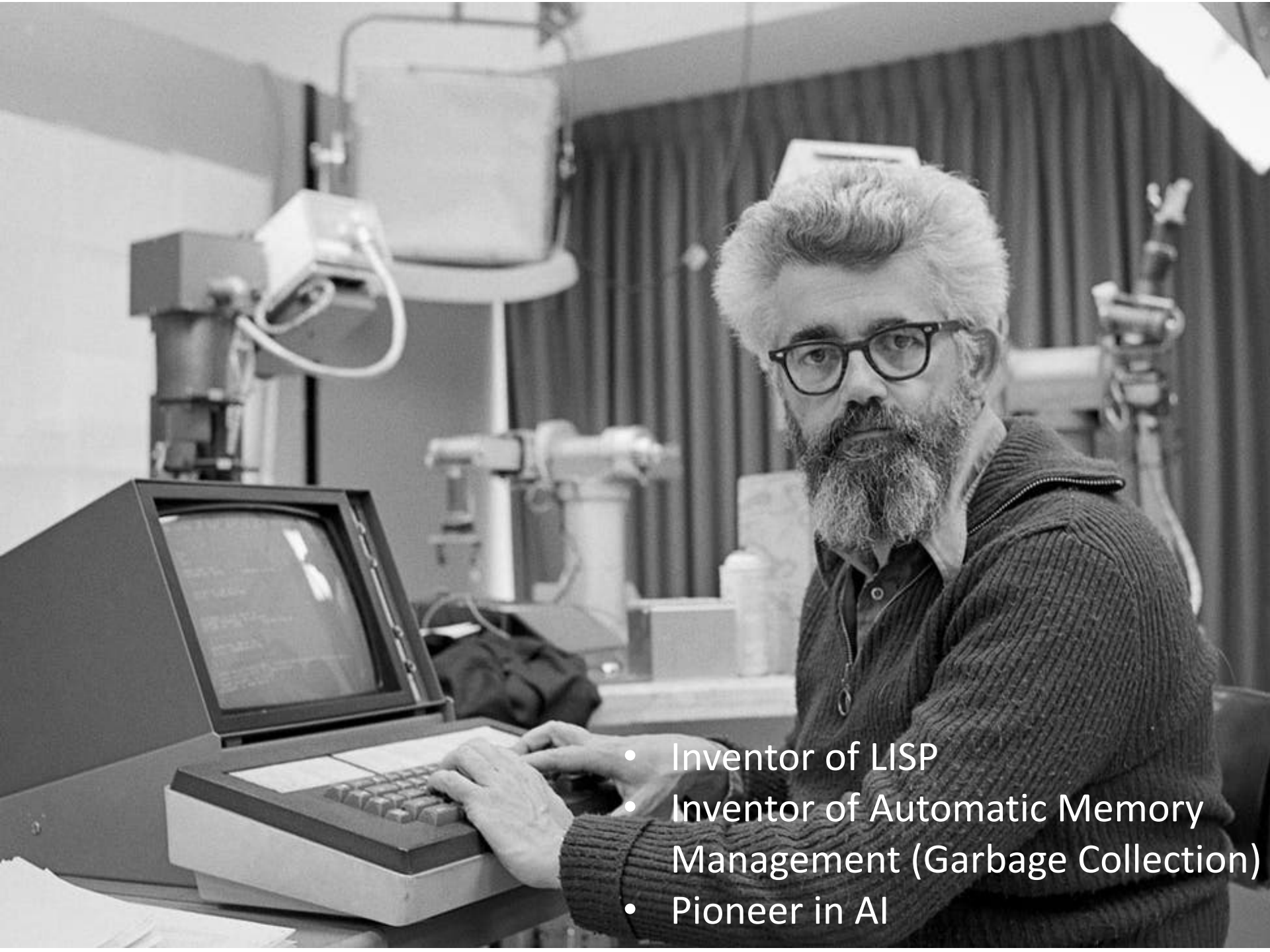
CSCI 1102 Computer Science 2

Meeting 19: Tuesday 4/6/2021

Quiz Review

Implementing Finite Partial Maps; BSTs


John McCarthy
Turing Award 1971


• Inventor of LISP
• Inventor of Automatic Memory Management (Garbage Collection)
• Pioneer in AI

The New York Times

Turing Award Goes to Creators of Computer Programming Building Blocks

Jeffrey Ullman and Alfred Aho developed many of the fundamental concepts that researchers use when they build new software.

Jeffrey Ullman, left, and Alfred Aho were pioneers in creating the "compilers" that translate programming languages into the ones and zeros that computers understand. Stanford University School of Engineering; Eileen Barosso


Google v Oracle
US Supreme Court (6-2) 4/5/2021:
Re-use of APIs (interfaces) are subject
to the *fair use* exception
of copyright law.

ARGUMENT

I. The Decisions Below Reflect the Federal Circuit's Fundamental Misunderstanding of How Interfaces Differ from Programs

The decisions below extend copyright protection to software interfaces—including the Java API—by erroneously equating them with computer programs. Asserting that software interfaces are simply a type of computer program, all of which are "by definition functional," the Federal Circuit misapplied general Ninth Circuit law that recognizes computer programs as copyrightable. *See* Pet. App. 162a. But software interfaces are not computer programs, and no party argues that "one can copy line-for-line someone else's copyrighted computer program." *Id.* at 239a.

The Federal Circuit's conclusory review fails to appreciate the district court's reasoned—and correct—

Engineer, Sun Microsystems. Designed Java Native Interface, led JVM development. Author, *The Java Native Interface*.

Barbara Liskov. Professor Emeritus, MIT. Created CLU, first programming language to support data abstraction; Argus, first high-level language to support distributed programming. ACM Turing Award, IEEE John von Neumann Medal, ACM SIGPLAN Programming Languages Award, ACM SIGOPS Lifetime Achievement Award. Fellow, ACM, AAoAS, National Academy of Inventors. Member, NAS, NAE.

Douglas McIlroy. Professor, Dartmouth. Headed Bell Labs department that originated Unix.

A2

Matthew Bishop. Professor, UC Davis. Author, *Computer Security: Art and Science*.

Joshua Bloch.† Professor, Carnegie-Mellon. Specialist in API Design. Previously, Chief Java Architect, Google; Distinguished Engineer, Sun Microsystems. Led design, implementation of numerous Java APIs. Author, *Effective Java*.

Gilad Bracha. Creator, Newspeak programming language. Previously, Scientist, Google; VP, SAP Labs; Distinguished Engineer, Sun Microsystems. Co-author, Java Language and VM Specifications. Dahl-Nygaard Prize.

Curtis Schroeder. Computer Scientist, Draper. Served as editor for widely reimplemented SISO CIGI API. Previously, Antycip Simulation, Lockheed Martin.

Robert Sedgewick. Founding Chair and Professor, Princeton CS Department. Co-inventor, Red-Black tree data structure. Author, 20 books including million-selling *Algorithms*. Steele Prize, ACM Karlstrom Award. Fellow, ACM.

Mary Shaw. Professor, Carnegie-Mellon. Specialist in software engineering. National Medal of Technology and Innovation, ACM SIGSOFT Outstanding Research Award, IEEE Distinguished Women in Software Engineering Award. Fellow, ACM, IEEE, AAAS.

```
public interface Queue<T> {
    boolean isEmpty();
    int size();
    void enqueue(T item);
    T dequeue();
    String toString();
}
```

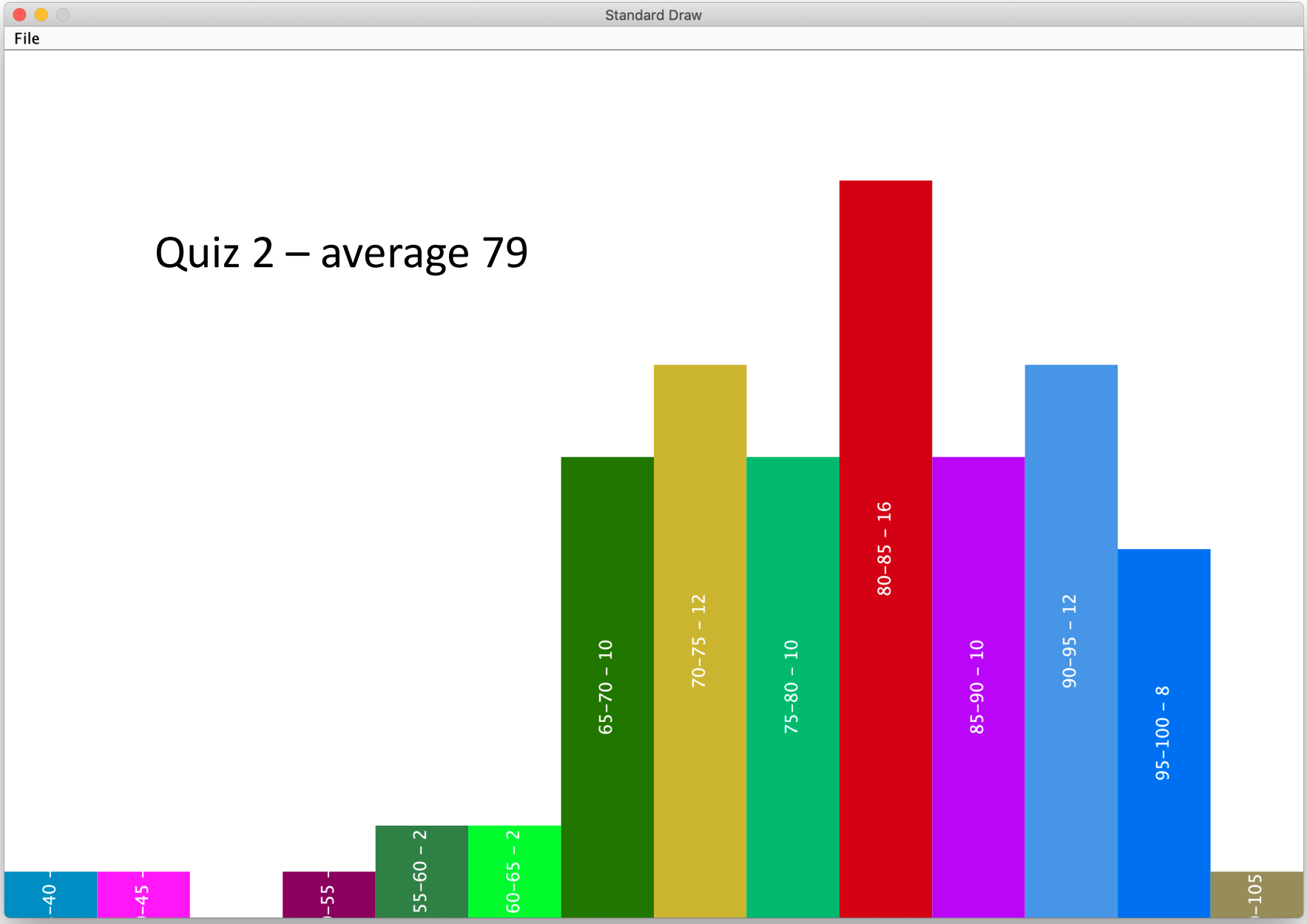
```
public class LinkedQueue<T> implements Queue<T> {
    public boolean isEmpty(){ … }
    public int size() { … }
    public void enqueue(T item) { … }
    public T dequeue() { … }
    public String toString() { … }
}
```

Protected by copyright.

```
public class LinkedQueue<T> implements Queue<T> {
    public boolean isEmpty(){ … }
    public int size() { … }
    public void enqueue(T item) { … }
    public T dequeue() { … }
    public String toString() { … }
}
```

Whether or not protected by copyright, subject to *fair use*.

```
public interface Queue<T> {
    boolean isEmpty();
    int size();
    void enqueue(T item);
    T dequeue();
    String toString();
}
```


Standard Draw
File
Quiz 2 – average 79
55–60 – 2
60–65 – 2
65–70 – 10
70–75 – 12
75–80 – 10
80–85 – 16
85–90 – 10
90–95 – 12
95–100 – 8

1. Java's `==`, `!=`, `<`, `<=`, `>` and `>=` operators can be used with operands of various types. In this and the next question, we're concerned with how they work on integers (i.e., values of type `int`).

   True or false: The `==` operator defines an equivalence relation on integers.

   Answer:

   `True`

2. True or false: The `<` operator defines a total order on integers.

   Answer:

   `False, the < operator isn't reflexive.`

3. We've discussed the idea of *statements* or *assertions* related to sets. For example, the containment statement $2 \in \{1, 3, 5\}$ states (falsely) that 2 is an element of the set $\{1, 3, 5\}$. We've also discussed general statements related to sets using *variables* and their *quantifers* $\forall$ (for all) and $\exists$ (there exists). For example, the statement $\forall n \in \{1, 3, 5\}.n > 4$ can be understood as a conjunction of three statements: $1 > 4$ *and* $3 > 4$ *and* $5 > 4$, another false statement. How can we interpret the statement $\exists n \in \{1, 3, 5\}.n > 4$?

   Answer: $1 > 4$ ***or*** $3 > 4$ ***or*** $5 > 4$**.**

4. Let $A = \{a, b, c\}$. $R = \{(b, b)\}$. Is R a reflexive relation on A?

   Answer:

   No

5. Let $A = \{a, b, c\}$. $R = \{(b, b)\}$. Is R a symmetric relation on A?

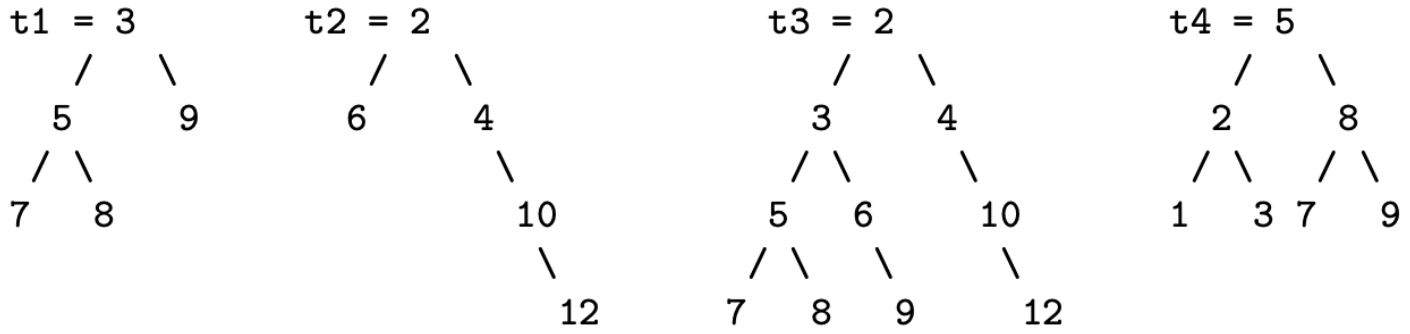
   Answer:

   Yes

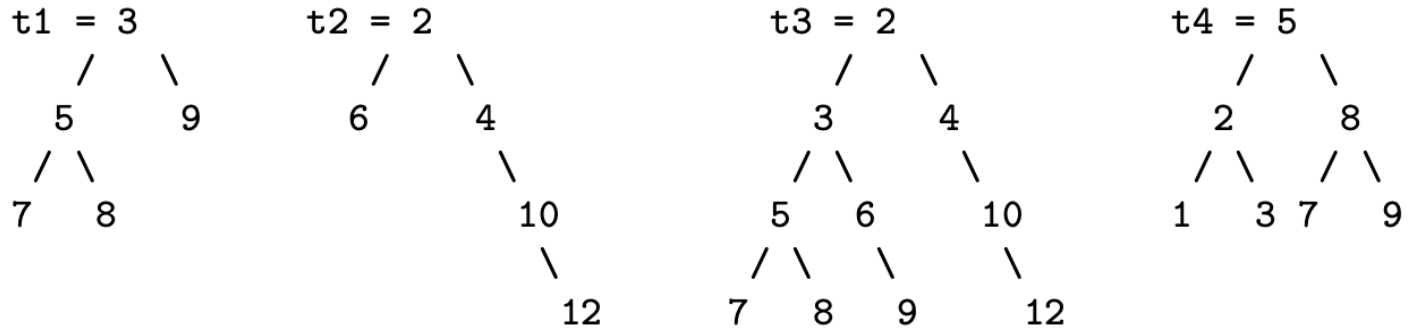
6. Let $A = \{a, b, c\}$. $B = \{0, 1\}$. Show two different partial maps from A to B.

   Answer:

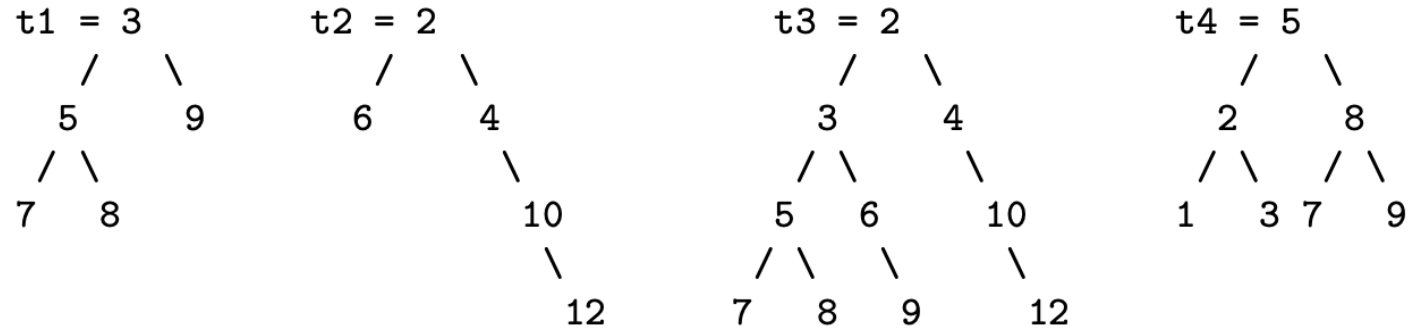
   {}
   {(a, 0)}



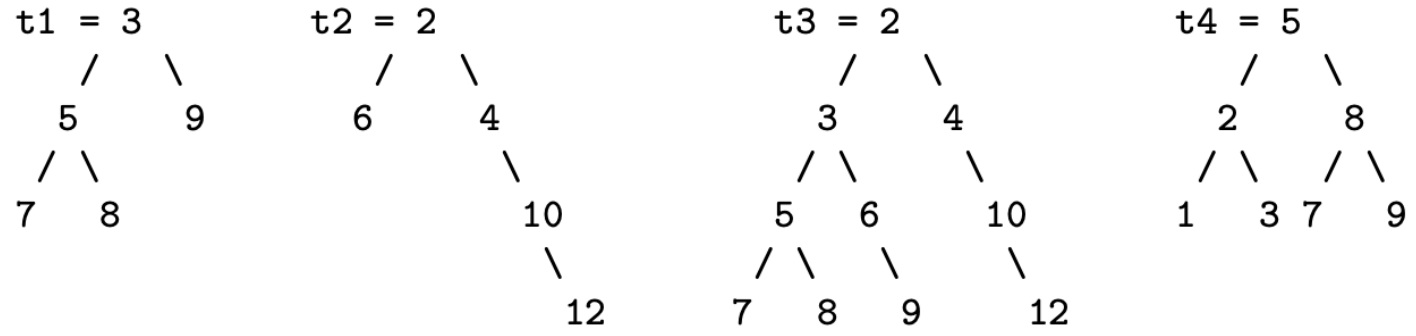

```
class Node {
  int info;
  Node left;
  Node right;
}
```



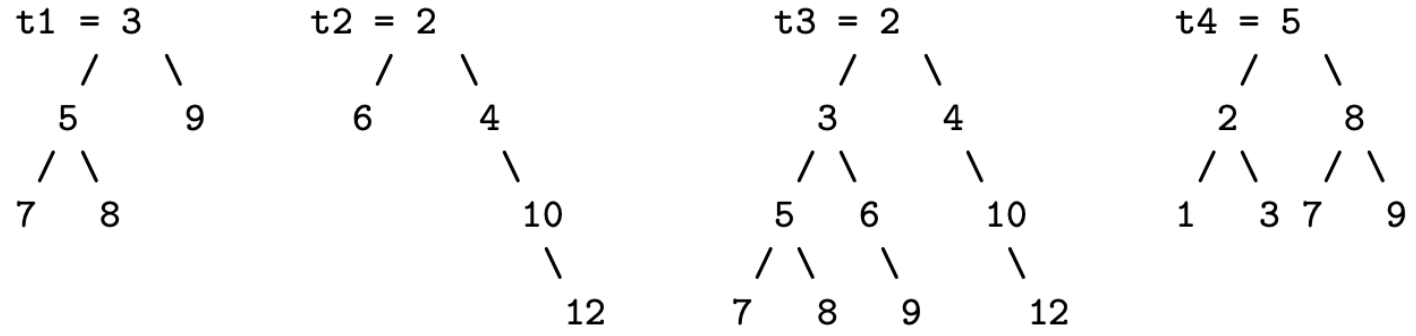
```
String pathTo(int item, Node a) {
  return pathTo(item, a, "start");
}
```



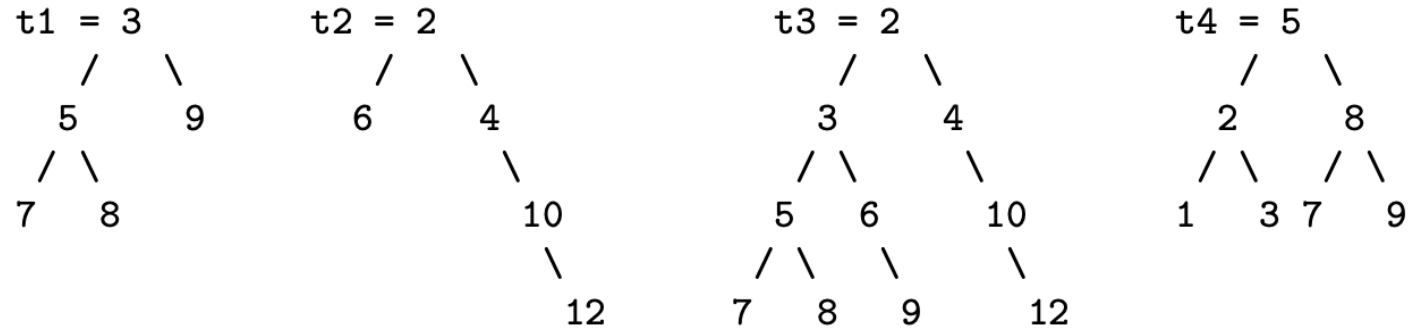

```
String pathTo(int item, Node a) {
  return pathTo(item, a, "start");
}

String pathTo(int item, Node a, String path) {
  if (a == null)
    return "no path";
  else if (a.info == item)
    return path;
  else {









  }
}
```



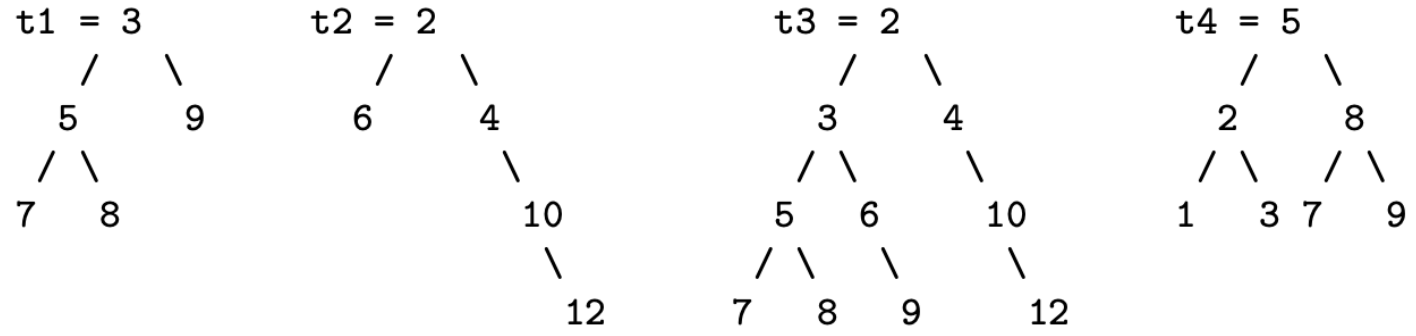

```
String pathTo(int item, Node a) {
  return pathTo(item, a, "start");
}

String pathTo(int item, Node a, String path) {
  if (a == null)
    return "no path";
  else if (a.info == item)
    return path;
  else {
    String leftPath = pathTo(item, a.left, String.format("%s left", path));
    if (!leftPath.equals("no path"))
      return leftPath;
    else
      return pathTo(item, a.right, String.format("%s right", path));
  }
}
```



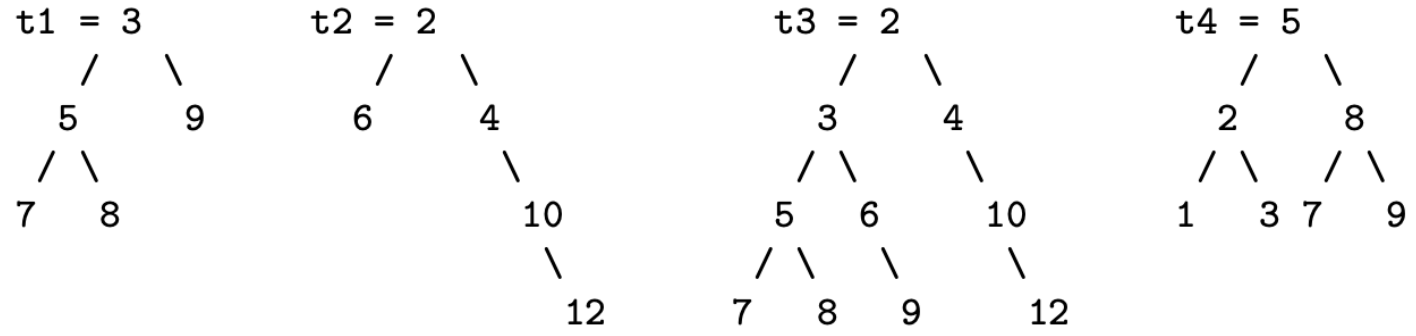

```
boolean isBalanced(Node a) {
  if (a == null || isLeaf(a))
    return true;
  int n = nodes(a);
  return height(a) <= (int) Math.floor(log2(n));
}
```




```
boolean isBalanced(Node a) {
  if (a == null || isLeaf(a))
    return true;
  int n = nodes(a);
  return height(a) <= (int) Math.floor(log2(n));
}

int nodes(Node a) {
  if (a == null)
    return 0;
  else
    return 1 + nodes(a.left) + nodes(a.right);
}
```

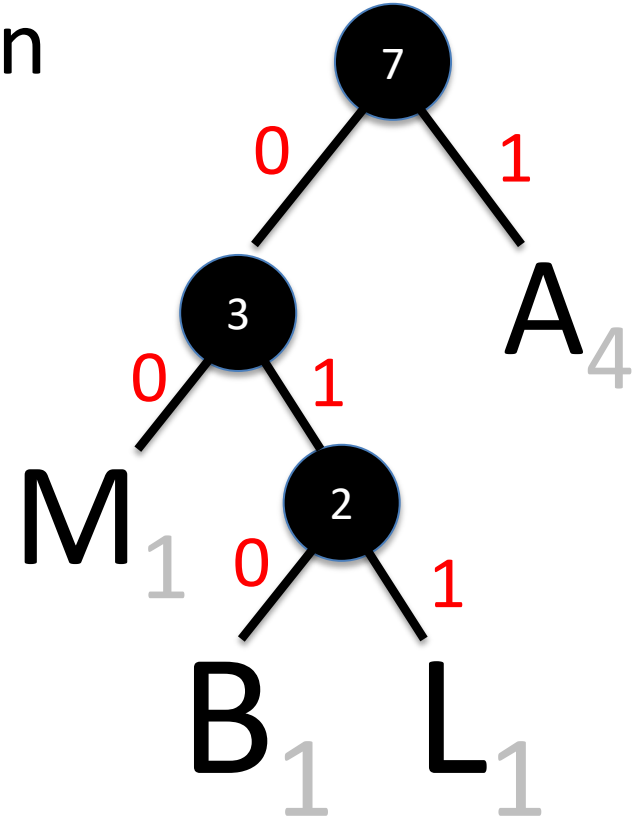



```
boolean isBalanced(Node a) {
  if (a == null || isLeaf(a))
    return true;
  int n = nodes(a);
  return height(a) <= (int) Math.floor(log2(n));
}

int nodes(Node a) {
  if (a == null)
    return 0;
  else
    return 1 + nodes(a.left) + nodes(a.right);
}

int height(Node a) {
  if (a == null || isLeaf(a))
    return 0;
  else
    return 1 + Math.max(height(a.left), height(a.right));
}
```




```
Node merge(Node a, Node b) {
  if (a == null) return b;
  if (b == null) return a;
  if (a.info < b.info) {
    if (StdRandom.uniform(2) == 0)
      return new Node(a.info, a.left, merge(a.right, b));
    else
      return new Node(a.info, merge(a.left, b), a.right);
  }
  else if (StdRandom.uniform(2) == 0)
    return new Node(b.info, b.left, merge(a, b.right));
  else
    return new Node(b.info, merge(a, b.left), b.right);
}
```

The Huffman Coding Tree



M = 00, B = 010; L = 011; A = 1

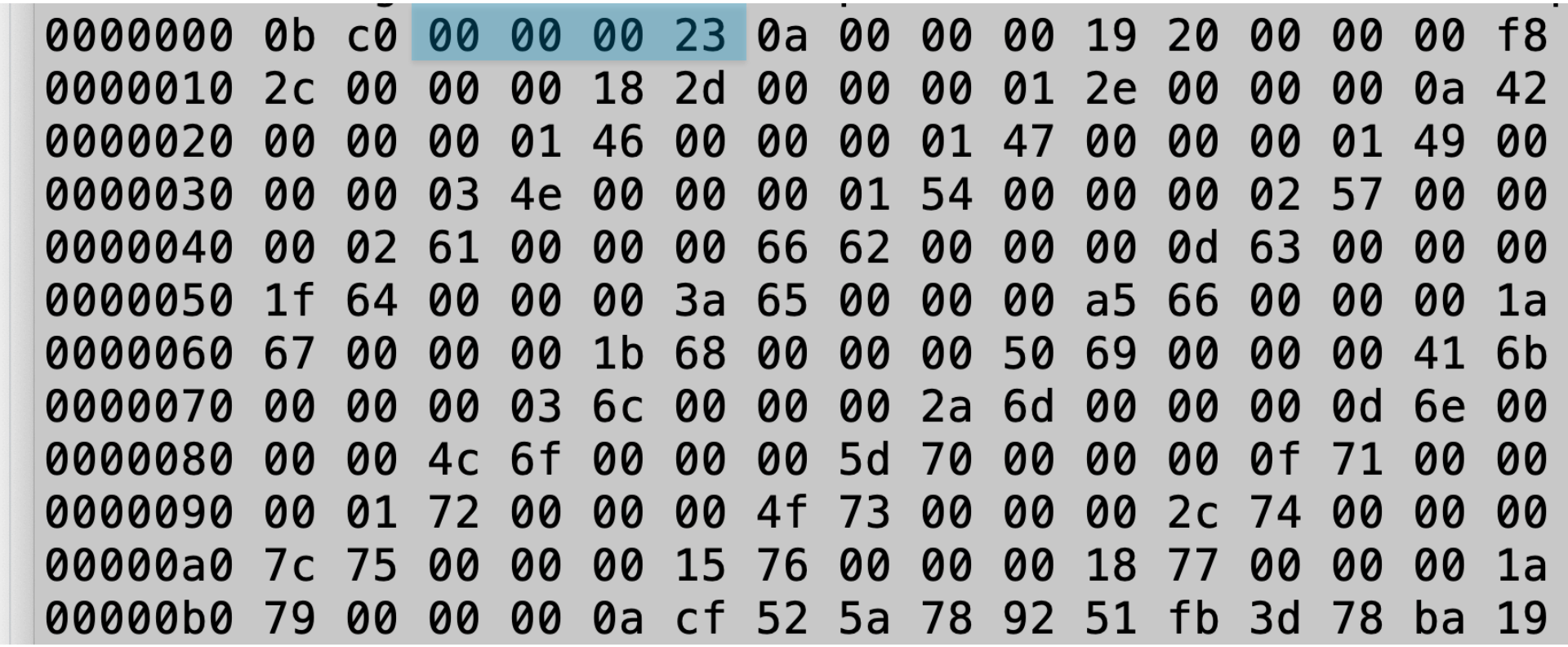
Unix hexdump of lincoln.zip

```
0000000 0b c0 00 00 00 23 0a 00 00 00 19 20 00 00 00 f8
0000010 2c 00 00 00 18 2d 00 00 00 01 2e 00 00 00 0a 42
0000020 00 00 00 01 46 00 00 00 01 47 00 00 00 01 49 00
0000030 00 00 03 4e 00 00 00 01 54 00 00 00 02 57 00 00
0000040 00 02 61 00 00 00 66 62 00 00 00 0d 63 00 00 00
0000050 1f 64 00 00 00 3a 65 00 00 00 a5 66 00 00 00 1a
0000060 67 00 00 00 1b 68 00 00 00 50 69 00 00 00 41 6b
0000070 00 00 00 03 6c 00 00 00 2a 6d 00 00 00 0d 6e 00
0000080 00 00 4c 6f 00 00 00 5d 70 00 00 00 0f 71 00 00
0000090 00 01 72 00 00 00 4f 73 00 00 00 2c 74 00 00 00
00000a0 7c 75 00 00 00 15 76 00 00 00 18 77 00 00 00 1a
00000b0 79 00 00 00 0a cf 52 5a 78 92 51 fb 3d 78 ba 19
```

magic number 0x0bc0

```
0000000 0b c0 00 00 00 23 0a 00 00 00 19 20 00 00 00 f8
0000010 2c 00 00 00 18 2d 00 00 00 01 2e 00 00 00 0a 42
0000020 00 00 00 01 46 00 00 00 01 47 00 00 00 01 49 00
0000030 00 00 03 4e 00 00 00 01 54 00 00 00 02 57 00 00
0000040 00 02 61 00 00 00 66 62 00 00 00 0d 63 00 00 00
0000050 1f 64 00 00 00 3a 65 00 00 00 a5 66 00 00 00 1a
0000060 67 00 00 00 1b 68 00 00 00 50 69 00 00 00 41 6b
0000070 00 00 00 03 6c 00 00 00 2a 6d 00 00 00 0d 6e 00
0000080 00 00 4c 6f 00 00 00 5d 70 00 00 00 0f 71 00 00
0000090 00 01 72 00 00 00 4f 73 00 00 00 2c 74 00 00 00
00000a0 7c 75 00 00 00 15 76 00 00 00 18 77 00 00 00 1a
00000b0 79 00 00 00 0a cf 52 5a 78 92 51 fb 3d 78 ba 19
```

Frequency table size 0x23 = 35_{10}

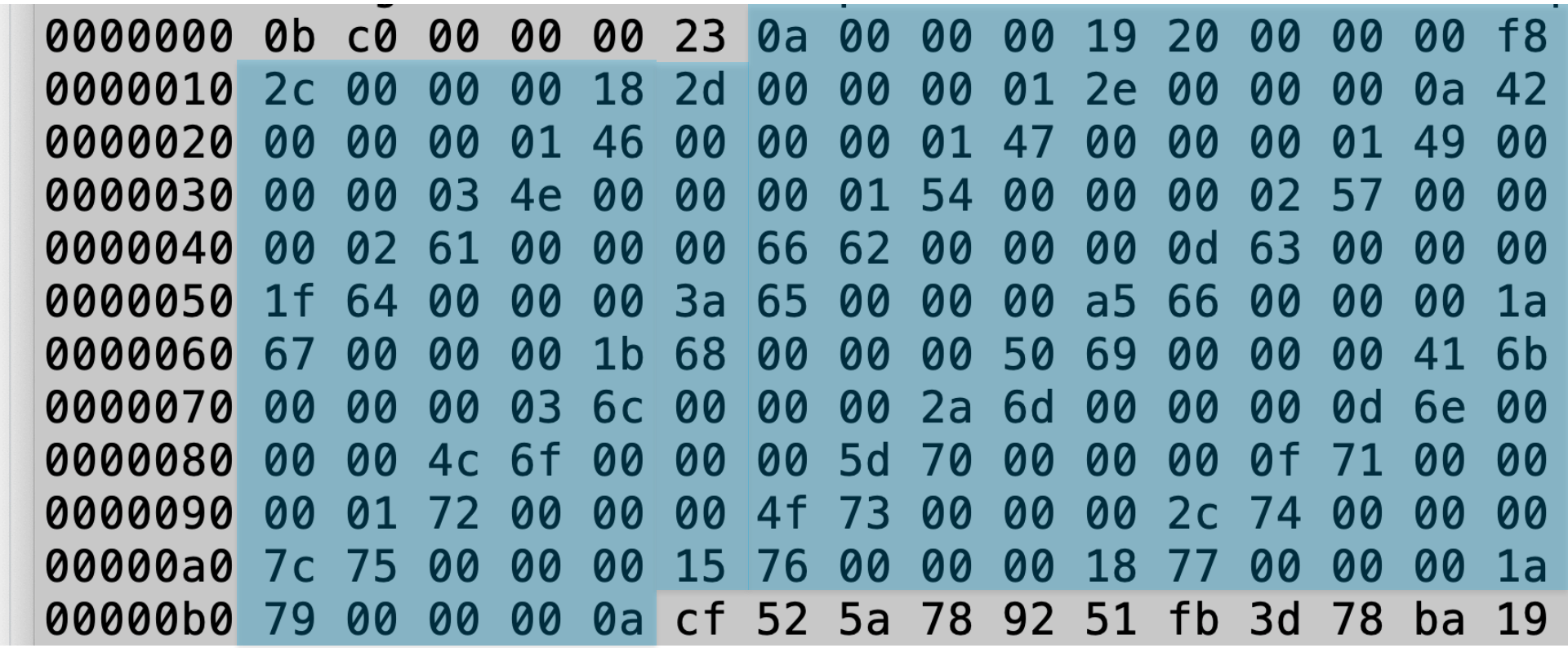


```
0000000 0b c0 00 00 00 23 0a 00 00 00 19 20 00 00 00 f8
0000010 2c 00 00 00 18 2d 00 00 00 01 2e 00 00 00 0a 42
0000020 00 00 00 01 46 00 00 00 01 47 00 00 00 01 49 00
0000030 00 00 03 4e 00 00 00 01 54 00 00 00 02 57 00 00
0000040 00 02 61 00 00 00 66 62 00 00 00 0d 63 00 00 00
0000050 1f 64 00 00 00 3a 65 00 00 00 a5 66 00 00 00 1a
0000060 67 00 00 00 1b 68 00 00 00 50 69 00 00 00 41 6b
0000070 00 00 00 03 6c 00 00 00 2a 6d 00 00 00 0d 6e 00
0000080 00 00 4c 6f 00 00 00 5d 70 00 00 00 0f 71 00 00
0000090 00 01 72 00 00 00 4f 73 00 00 00 2c 74 00 00 00
00000a0 7c 75 00 00 00 15 76 00 00 00 18 77 00 00 00 1a
00000b0 79 00 00 00 0a cf 52 5a 78 92 51 fb 3d 78 ba 19
```

Table entry: 0x0a = newline, 0x19 =25_{10}

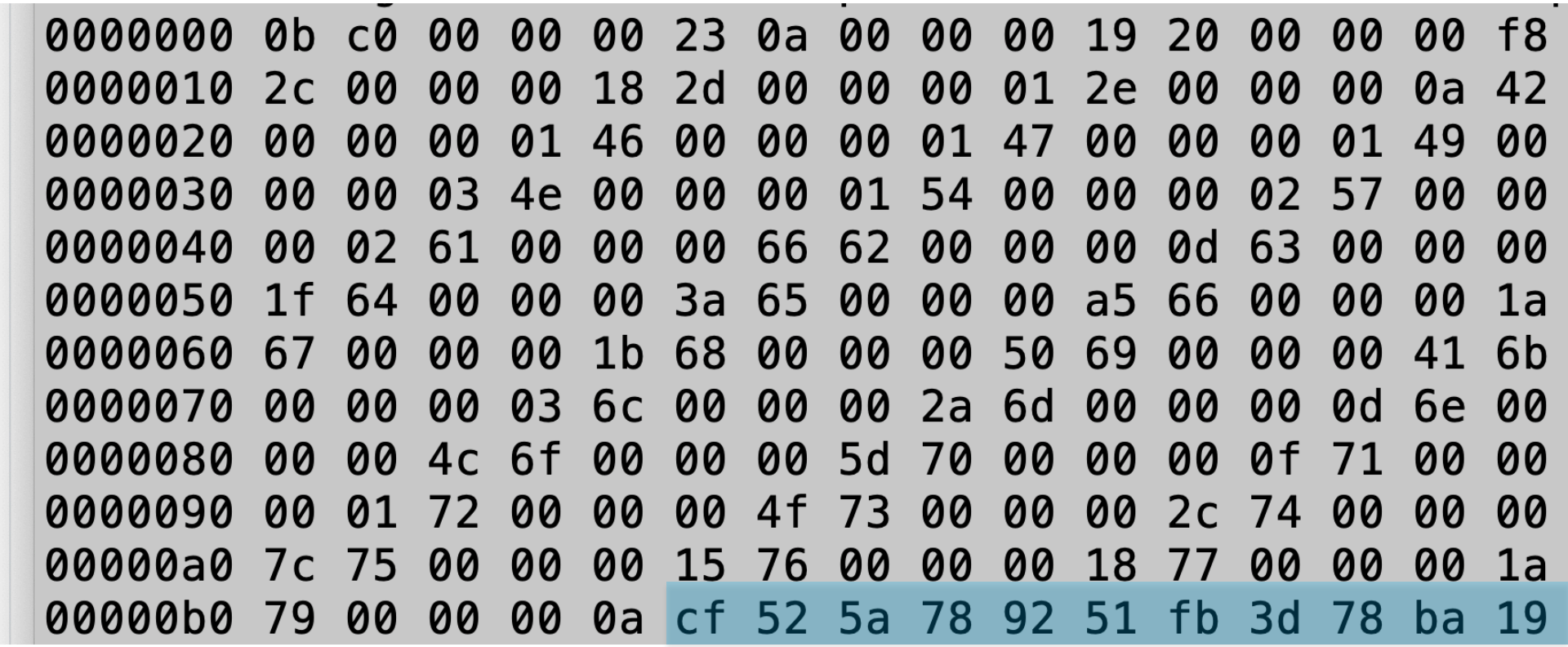
```
0000000 0b c0 00 00 00 23 0a 00 00 00 19 20 00 00 00 f8
0000010 2c 00 00 00 18 2d 00 00 00 01 2e 00 00 00 0a 42
0000020 00 00 00 01 46 00 00 00 01 47 00 00 00 01 49 00
0000030 00 00 03 4e 00 00 00 01 54 00 00 00 02 57 00 00
0000040 00 02 61 00 00 00 66 62 00 00 00 0d 63 00 00 00
0000050 1f 64 00 00 00 3a 65 00 00 00 a5 66 00 00 00 1a
0000060 67 00 00 00 1b 68 00 00 00 50 69 00 00 00 41 6b
0000070 00 00 00 03 6c 00 00 00 2a 6d 00 00 00 0d 6e 00
0000080 00 00 4c 6f 00 00 00 5d 70 00 00 00 0f 71 00 00
0000090 00 01 72 00 00 00 4f 73 00 00 00 2c 74 00 00 00
00000a0 7c 75 00 00 00 15 76 00 00 00 18 77 00 00 00 1a
00000b0 79 00 00 00 0a cf 52 5a 78 92 51 fb 3d 78 ba 19
```

Frequency Table



```
0000000 0b c0 00 00 00 23 0a 00 00 00 19 20 00 00 00 f8
0000010 2c 00 00 00 18 2d 00 00 00 01 2e 00 00 00 0a 42
0000020 00 00 00 01 46 00 00 00 01 47 00 00 00 01 49 00
0000030 00 00 03 4e 00 00 00 01 54 00 00 00 02 57 00 00
0000040 00 02 61 00 00 00 66 62 00 00 00 0d 63 00 00 00
0000050 1f 64 00 00 00 3a 65 00 00 00 a5 66 00 00 00 1a
0000060 67 00 00 00 1b 68 00 00 00 50 69 00 00 00 41 6b
0000070 00 00 00 03 6c 00 00 00 2a 6d 00 00 00 0d 6e 00
0000080 00 00 4c 6f 00 00 00 5d 70 00 00 00 0f 71 00 00
0000090 00 01 72 00 00 00 4f 73 00 00 00 2c 74 00 00 00
00000a0 7c 75 00 00 00 15 76 00 00 00 18 77 00 00 00 1a
00000b0 79 00 00 00 0a cf 52 5a 78 92 51 fb 3d 78 ba 19
```

Start of Bit Stream



Finite Partial Maps

aka

Dictionaries

Symbol Tables

Finite Partial Maps

Let R be a binary relation on A, B.

R is a *partial map* from A to B if and only iff

$$\forall a \in A.\, b, b' \in B.\, \text{if } (a, b) \in R \text{ and } (a, b') \in R \text{ then } b = b'.$$

A – a set of *keys*
B – a set of *values*

{ “Alice” : 12, “Bob” : 8 }

General Considerations

- Must be able to find a key => need a reasonable equivalence relation on keys.

General Considerations

- Must be able to find a key => need a reasonable equivalence relation on keys.

- What to do if key isn't in the map?
  - Return null – heads up: using null for 2nd purpose here
  - Throw a java.util.NoSuchElementException
  - Return java.util.Optional.empty()

General Considerations

- Which operations need to be fast?
  - Worst case? Average case? Amortized?
  - What about storage?
  - Locality?

General Considerations

- Which operations need to be fast?
  - Worst case? Average case? Amortized?
  - What about storage?
  - Locality?
- Should the implementation be mutable or immutable?

java.util

Scree

Interface Map<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

All Known Subinterfaces:

Bindings, ConcurrentMap<K,V>, ConcurrentNavigableMap<K,V>, LogicalMessageContext, MessageContext, NavigableMap<K,V>, SOAPMessageContext, SortedMap<K,V>

All Known Implementing Classes:

AbstractMap, Attributes, AuthProvider, ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, HashMap, Hashtable, IdentityHashMap, LinkedHashMap, PrinterStateReasons, Properties, Provider, RenderingHints, SimpleBindings, TabularDataSupport, TreeMap, UIDefaults, WeakHashMap

---

public interface **Map<K,V>**

Sedgewick & Wayne's "API"

API. Here is the API.

public class ST<Key, Value>		
	ST()	*create a symbol table*
void	put(Key key, Value val)	*put key-value pair into the table (remove* key *from table if value is* null*)*
Value	get(Key key)	*value paired with* key *(*null *if* key *is absent)*
void	delete(Key key)	*remove* key *(and its value) from table*
boolean	contains(Key key)	*is there a value paired with* key*?*
boolean	isEmpty()	*is the table empty?*
int	size()	*number of key-value pairs in the table*
Iterable<Key>	keys()	*all the keys in the table*

API for a generic basic symbol table

```java
public interface UnorderedMap<Key, Value> {
  void put(Key key, Value value);

  Value get(Key key);

  void delete(Key key);

  boolean contains(Key key);

  boolean isEmpty();

  int size();

  Iterable<Key> keys();
}
```

We use interfaces for APIs

Ordered or Sorted Maps

- If there is a total order on keys – *Ordered (or Sorted) Maps*
  - can implement additional operations related to the order and
  - can consider data structures supporting efficient operations leveraging the order.

java.util

Scree

Interface SortedMap<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

All Superinterfaces:

Map<K,V>

All Known Subinterfaces:

ConcurrentNavigableMap<K,V>, NavigableMap<K,V>

All Known Implementing Classes:

ConcurrentSkipListMap, TreeMap

---

public interface **SortedMap<K,V>**
extends Map<K,V>

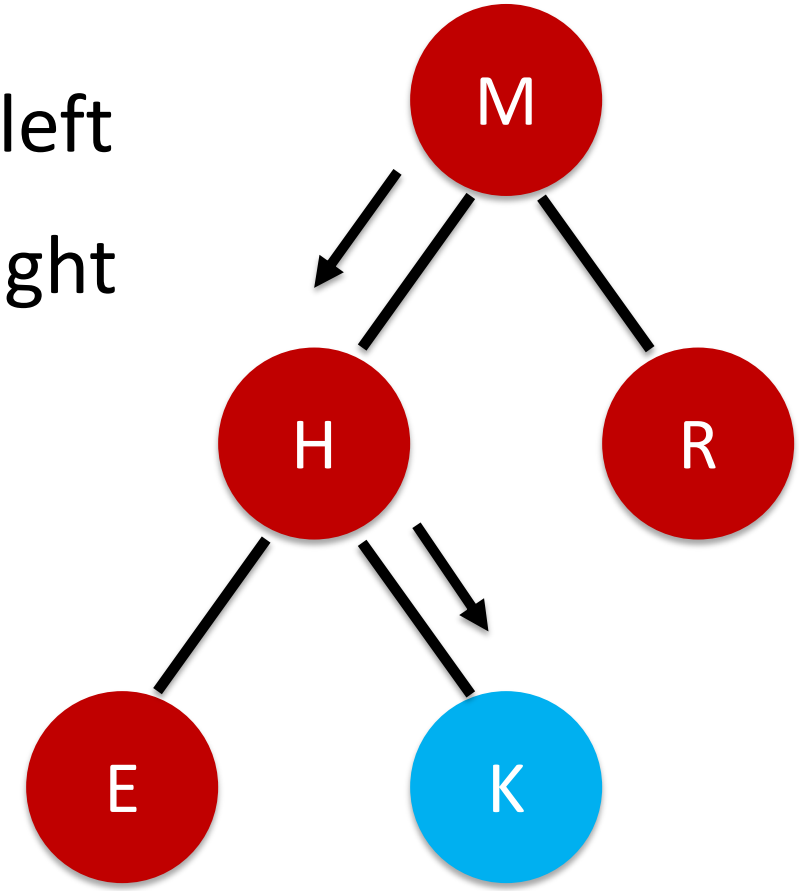
Implementations

- Simple binary search trees (Ordered, SW 3.2)
- Balanced binary search trees (Ordered, SW 3.3)
- Hash Tables (Unordered, SW 3.4)
- Skip Lists (Ordered, not covered in SW)

Binary Search Trees

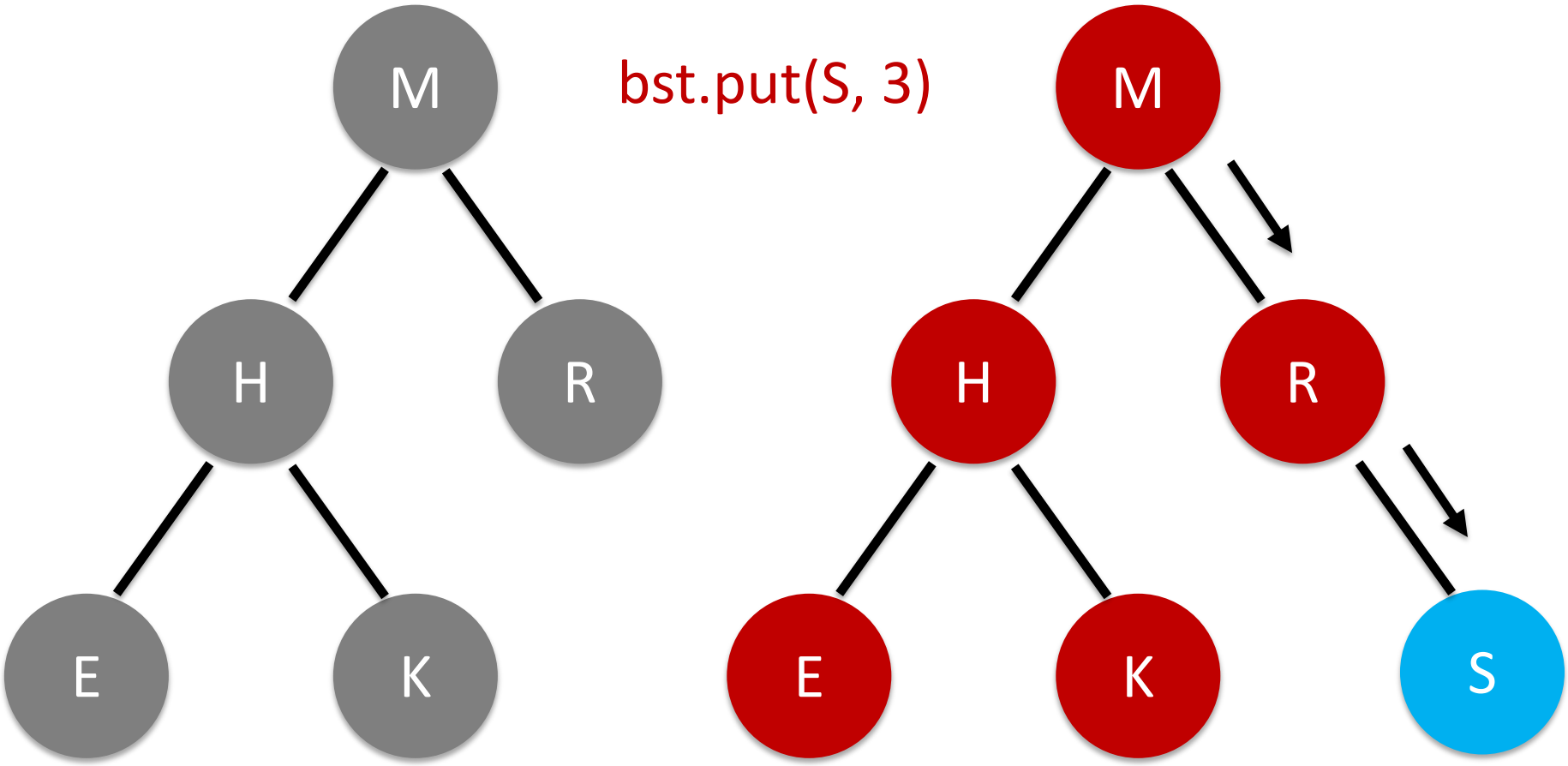
- Smaller keys to the left
- Larger keys to the right

bst.get(K)

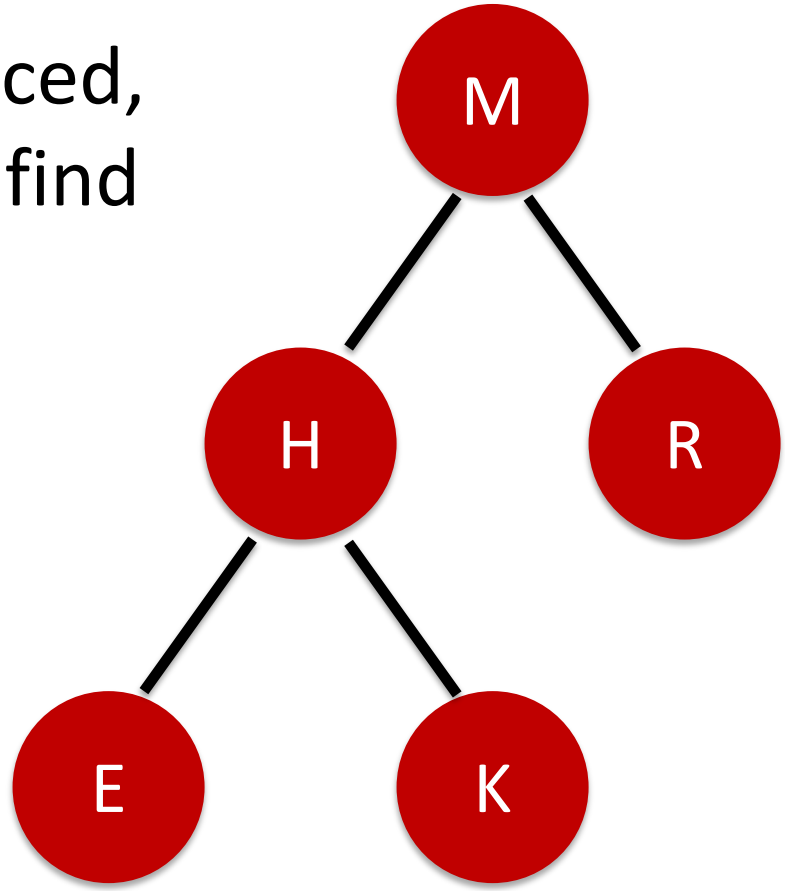


Operations on BSTs must Preserve the Invariants

BSTs: Smaller to the left, larger to the right

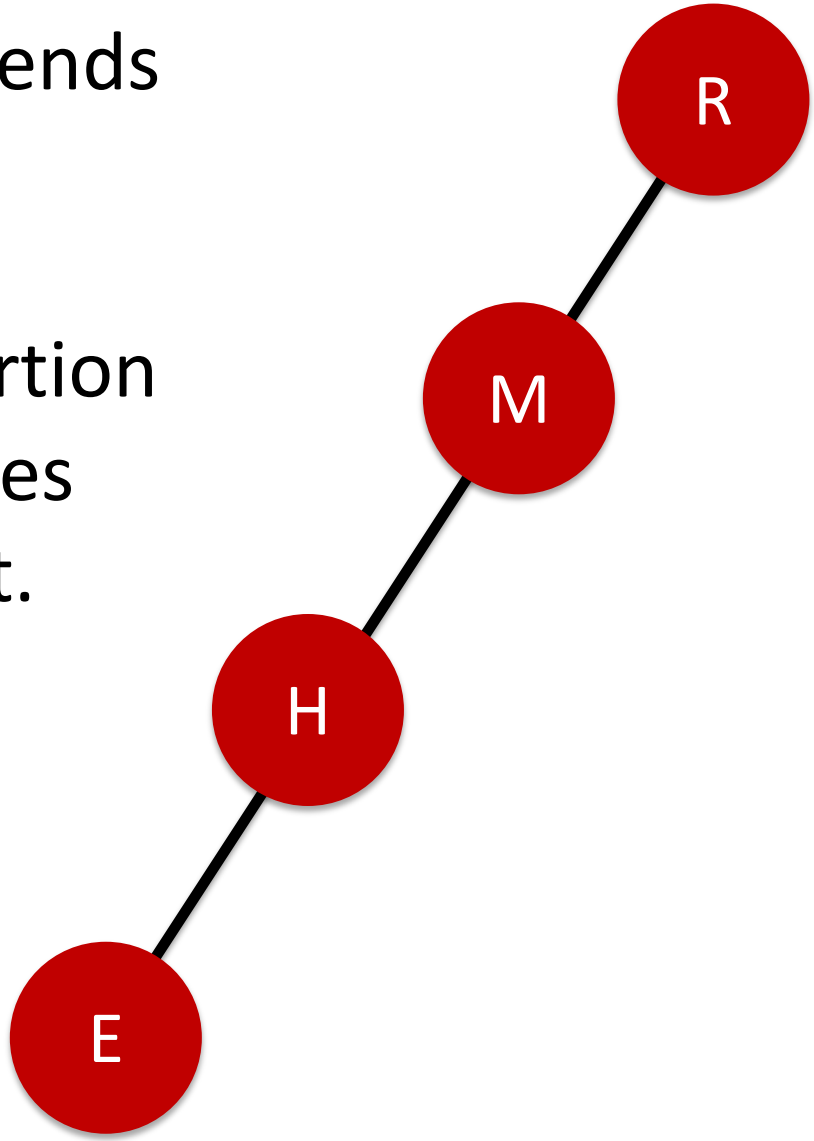


If the tree is balanced, $\log_2$ insertion and find time.

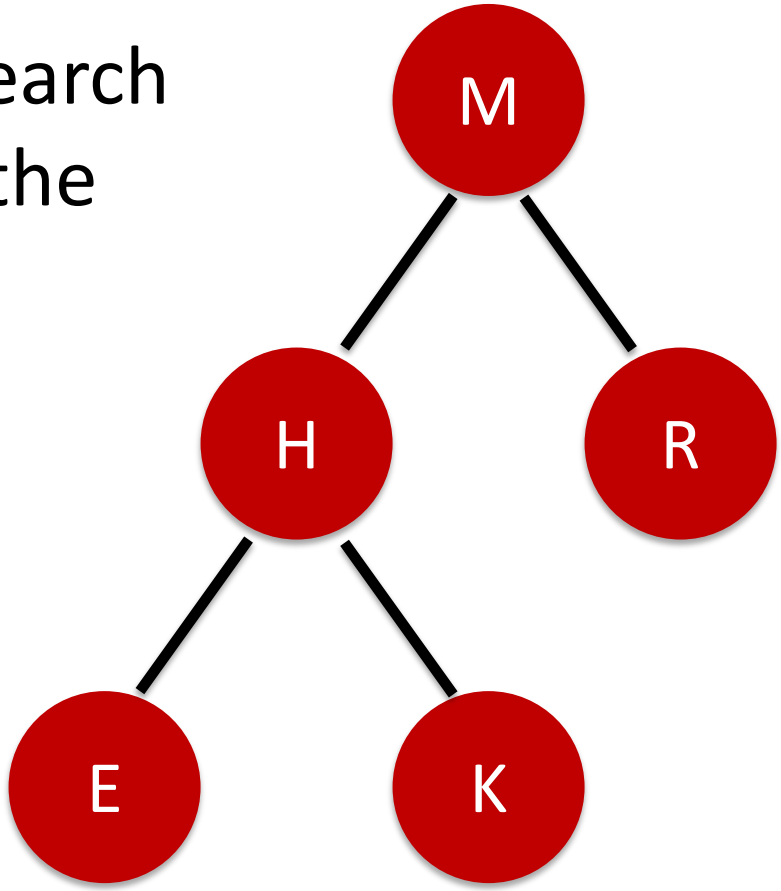


Structure of tree depends on insertion order.

With an unlucky insertion order, the tree behaves like a singly-linked list.



Balanced binary search trees ensure that the tree remains well-balanced.

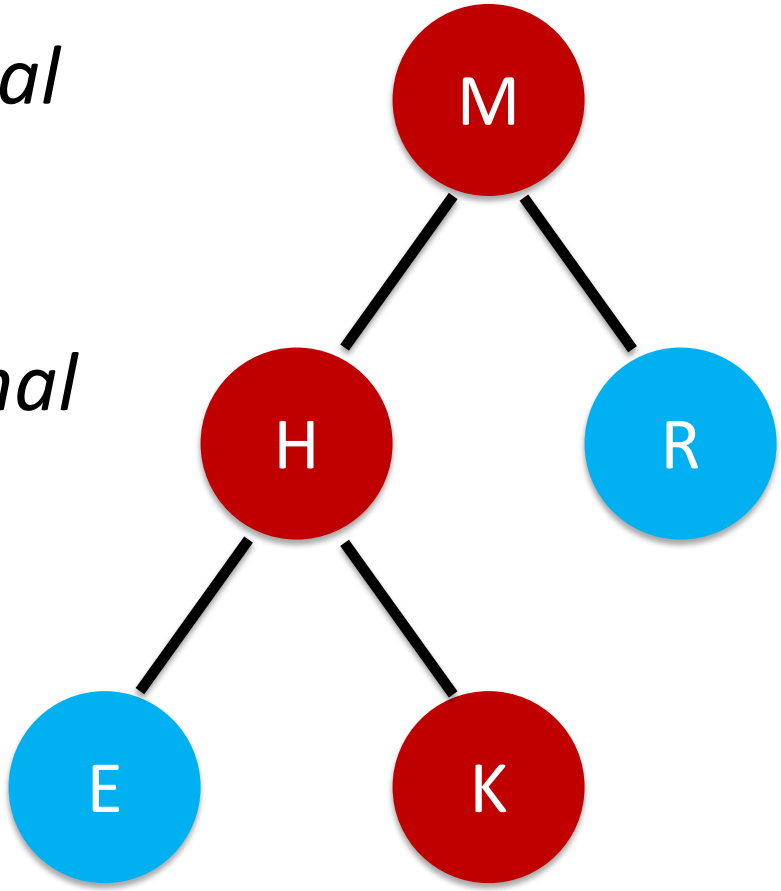


```java
public interface OrderedMap<Key extends Comparable<Key>, Value> {
  void put(Key key, Value value);

  Value get(Key key);

  void delete(Key key);

  boolean contains(Key key);

  boolean isEmpty();

  int size();

  Iterable<Key> keys();

  Key ceiling(Key key);

  Key floor(Key key);

  Key max();

  Key min();
}
```

```java
public class BST<K extends Comparable<K>, V> implements OrderedMap<K, V> {

  private Node root;

  private class Node {
    private K key;
    private V value;
    private Node left;
    private Node right;
    private int size;
```

Finding the *minimal* key E – go left.

Finding the *maximal* key R – go right.

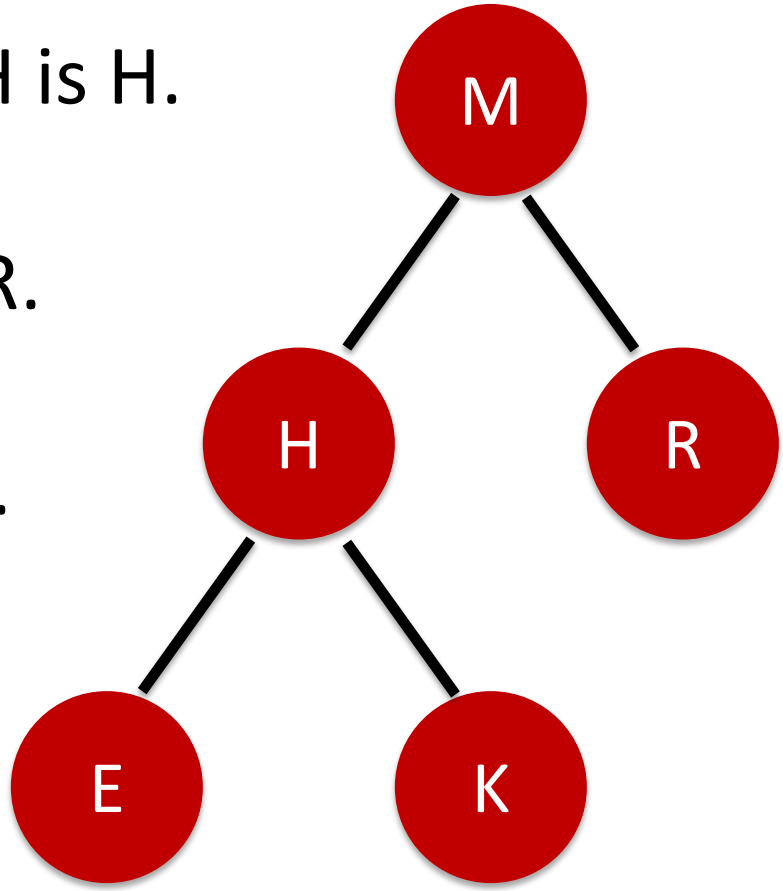


The *ceiling* of key H is H.

The ceiling of N is R.

The ceiling of J is K.

The ceiling of S doesn't exist.



Next time:
How to delete a key?