public TreeNode add(TreeNode root, String value) {
    if (root == null) {
        return new TreeNode(value, null, null);
    }
    int comp = value.compareTo(root.info);
    if (comp <= 0) {
        root.left = add(root.left, value);
    } else {
        root.right = add(root.right, value);
    }
    return root;
}
U is for …

• Unix
  • Basis for Linux and Android and GNU and …
• User Interface/UI, User Experience/UX
  • User is the heart and soul
Google interface

- Marissa Mayer
  - Google – employee number 20
  - Yahoo CEO
  - Lumi Labs
Announcements

• APT-6 due Tuesday, April 7
• APT-7 out this week and due April 16 (note Thursday)
• Assignment P5 Percolation
  • Code due Thursday, April 9, grace thru April 13
• Assignment P6 Huffman out end of week, due April 22

• Discussion 12 Monday, April 13
  • Pre-discussion out at end of week

• Exam 2 is April 10 – Your own work!

• APT Quiz 2 is April 11-15 – Your own work!
PfSWoA

- Exam 2 Review and Logistics
  - O-Notation, Big-Oh, Recurrences
  - Linked Lists
  - Binary Trees

- Comparator Review
  - API for sorting, solving problem review
  - Helpful on next APTs out
Exam 2

• Take any time on Friday April 10. Maybe earlier.
• Take on Gradescope, DO NOT GO TO GRADESCOPE UNTIL YOU ARE READY
  • ONCE YOU CLICK ON IT, YOUR TIMER STARTS
• If you cannot take it on Friday, April 10
  • Fill out form to tell us why, and the soonest day you can take it
• Fill out a reflect form after taking the exam
  • Get 2 extra points
  • Give us feedback
More Exam 2 Details - GradeScope

• MC, short answer, and short code segments.
• You will type in, or click on answers.
• Suggest: write code on paper and then type it in.
• Submit each question as you go.
  • Lose internet, just connect back in.
• Exam is about 1 hour, 15 min.
• You get an extra 1 hour, 15 min for logistics.
• Total time is 2 hr, 30 min.
• Those with accommodations, Kate will email.
Exam 2 – Honor Code

• The exam is your work only
• Use books, open notes, code you have written
• Write code on paper only, then type into Gradescope
• DO NOT write code and run it in IntelliJ, Jshell or other computer means
• DO NOT Search on the web for answers to problems
• DO NOT Talk to any humans about the exam during the exam period
• DO NOT Talk to any humans about the exam until the exam is handed back
Now let’s review for the exam
Runtime and Other Analysis

- Asymptotic complexity of runtime or memory
  - Return value: `bleem(10)=88, bleem(20)=360`
  - \(1+2+\ldots+n=O(n^2)\)

- Runtime and value
  - \(\text{sum += 1}\)
Intuition and Reality

• "fun" x 4 = "funfunfunfun"
  • https://www.youtube.com/watch?v=VbrEsOLu75c
  • Analyze only in terms of n, say with "fun"
    • It's all about line 34, how much does it matter?
• 3, 6, 9, 12, …, 3n = 3(1 + 2 + … + n) so big-Oh?
  • What if length of base is m, then big-Oh

```java
public String appendAlot(String base, int n) {
    String ret = "";
    for (int k=0; k < n; k++) {
        ret += base;
    }
    return ret;
}
```
Buffering … and faster

- "fun" x 4 = "funfunfunfun"
  - It's all about line 41, how much does is matter?
- 3, 3, 3, … ,3 = so 3n and O(n)
  - What if length of base is m, then big-Oh?

```java
public String appendAlot2(String base, int n) {
    StringBuilder sb = new StringBuilder();
    for(int k=0; k < n; k++) {
        sb.append(base);
    }
    return sb.toString();
}
```
WOTO-O Review

public TreeNode add(TreeNode root, String value) {
    if (root == null) {
        return new TreeNode(value, null, null);
    }
    int comp = value.compareTo(root.info);
    if (comp <= 0) {
        root.left = add(root.left, value);
    } else {
        root.right = add(root.right, value);
    }
    return root;
}
Recursive Insert and Recurrences

- Recurrence relation for .add below? (TreeDemo.java)
  - https://coursework.cs.duke.edu/201spring20/classcode/
  - $T(N)$ is time to add one value to tree of $N$ values, what is complexity of adding $N$ values one-at-a-time?

```java
class TreeNode {
    public TreeNode add(String value) {
        if (root == null) {
            return new TreeNode(value);
        }
        int comp = value.compareTo(root.info);
        if (comp <= 0) {
            root.left = add(root.left, value);
        } else {
            root.right = add(root.right, value);
        }
        return root;
    }
}
```
Understand and Analyze

• If tree has N values
  • One recursive call: $T(n) = T(n/2) + O(1)$
  • worst: $T(n) = T(n-1) + O(1)$, solutions?

• We do this n times. Upper bound $n \log n$, but …
  • $\log(1) + \log(2) + \ldots + \log(n) = \log(1 \times 2 \times \ldots \times n)$
  • $\log(n!)$
  • $n \log n$
  • Stirling's Formula

```java
public TreeNode add(TreeNode root, String value) {
    if (root == null) {
        return new TreeNode(value, null, null);
    }
    int comp = value.compareTo(root.info);
    if (comp <= 0) {
        root.left = add(root.left, value);
    } else {
        root.right = add(root.right, value);
    }
    return root;
}
```
Recurrence Relations

No need to derive, remember or look up

<table>
<thead>
<tr>
<th>Recurrence</th>
<th>Algorithm</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) = T(n/2) + O(1)$</td>
<td>binary search</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>$T(n) = T(n-1) + O(1)$</td>
<td>sequential search</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$T(n) = 2T(n/2) + O(1)$</td>
<td>tree traversal</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$T(n) = T(n/2) + O(n)$</td>
<td>quicksort partition</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>mergesort, quicksort</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$T(n) = T(n-1) + O(n)$</td>
<td>selection or bubble sort</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
WOTO Midterm Tree Review

public TreeNode add(TreeNode root, String value) {
    if (root == null) {
        return new TreeNode(value, null, null);
    }
    int comp = value.compareTo(root.info);
    if (comp <= 0) {
        root.left = add(root.left, value);
    } else {
        root.right = add(root.right, value);
    }
    return root;
}
When I was a kid, I was, ..., very much fascinated by Star Wars, and what fascinated me the most is this notion of the Force, this energy that connects all people and all objects and allows you to feel people that you can't even see. And I remember many nights, I would be sitting at home, just, like, concentrating and focusing, trying to feel the Force, and I didn't feel anything, don't worry.
Linked List - LowLevelLinkDemo.java

• Convert array to linked-list, values in same order
  • [Link: https://coursework.cs.duke.edu/201spring20/classcode/]
• Choices: add to front or back of list being built
  • If we add to front? loop over array values …
  • If we add to back? Need pointer to front too
• Invariant?
  • last ...

```java
private Node createList(String[] array) {
    if (array.length == 0) return null;
    Node first = new Node(array[0], null);
    Node last = first;
    for (int k = 1; k < array.length; k += 1) {
        last.next = new Node(array[k], null);
        last = last.next;
    }
    return first;
}
```
Add to Back Invariant

- **Choices:** add to front or back of list being built
  - If we add to back? Need pointer to front too
- **Invariant**
  - last is last node of list with values in range \([0,k)\)
  - True before loop guard first tested? \([0,1)\)
  - Updated in loop?

```java
private Node createList(String[] array) {
    if (array.length == 0) return null;

    Node first = new Node(array[0], null);
    Node last = first;
    for (int k = 1; k < array.length; k += 1) {
        last.next = new Node(array[k], null);
        last = last.next;
    }
    return first;
}
```
Linked List Concepts Reviewed

- Loop with statement `list = list.next`
- Recurse with call `doit(list.next)`
- Must assign to `.next` field to create a linked list
  - Can be via node constructor

- Verify code works: empty list and last node of list
  - `void` null-pointer dereference
WOTO Link Review

public TreeNode add(TreeNode root, String value) {
    if (root == null) {
        return new TreeNode(value, null, null);
    }
    int comp = value.compareTo(root.info);
    if (comp <= 0) {
        root.left = add(root.left, value);
    }
    else {
        root.right = add(root.right, value);
    }
    return root;
}
Sorting APTs

• Creating inner/nested classes to help
  • https://www2.cs.duke.edu/csed/newapt/clientlist.html
  • https://www2.cs.duke.edu/csed/newapt/medaltable.html

• Sorting Map.Entry<Key,Value> pairs
  • https://www2.cs.duke.edu/csed/newapt/sortbyfreqs.html

• Simple Use of API
  • https://www2.cs.duke.edu/csed/newapt/lengthsort.html
ClientsList APT

- Nested class that can be compared: Person
  - Two approaches
    - Implement Comparable and a newer approach
  - Solve? Translate, Process, Translate
How to sort: old and new

• This is the OLD way to sort in Java

```java
@Override
public int compareTo(Person p) {
    int ldiff = myLast.compareTo(p.myLast);
    if (ldiff != 0) return ldiff;
    return myFirst.compareTo(p.myFirst);
}
```

• Here is newer way, no need for Comparable

```java
Arrays.sort(list, Comparator.comparing(Person::getLast)
    .thenComparing(Person::getFirst));
```
APT SortByFreqs

Problem Statement

The frequency with which data occurs is sometimes an important statistic. In this problem you are given an array of strings and must determine how frequently the strings occur. Return an array of strings that is sorted (ordered) by frequency. The first element of the returned array is the most frequently occurring string, the last element is the least frequently occurring. Ties are broken by listing strings in lexicographic/alphabetical order. The returned array contains one occurrence of each unique string from the array parameter.

Consider these strings (quotes for clarity, they're not part of the strings).

{"apple", "pear", "cherry", "apple", "pear", "apple", "banana"}

The array returned is:

{ "apple", "pear", "banana", "cherry" }

since the most frequently occurring string is "apple" which occurs 3 times; the string "pear" occurs twice and the other strings each occur once so they are returned in alphabetical order.
Problem Statement

The frequency with which data occurs is sometimes an important statistic. In this problem you'll determine how frequently strings occur and return an array representing the frequencies of each different/unique string. The array returned contains as many frequencies as there are unique strings. The returned frequencies represent an alphabetic/lexicographic ordering of the unique words, so the first frequency is how many times the alphabetically first word occurs and the last frequency is the number of times the alphabetically last word occurs.

Consider these strings (quotes for clarity, they're not part of the strings).

{"apple", "pear", "cherry", "apple", "cherry", "pear", "apple", "banana"}

The array returned is \{3,1,2,2\} since the alphabetically first word is "apple" which occurs 3 times; the second word alphabetically is "banana" which occurs once, and the other words each occur twice.
SortByFreqs, SortedFreqs

• Can use TreeMap to keep count of each string
  • `Map.Entry` class is like inner class `Person`
  • However, does efficiency matter?

• Suppose for SortedFreqs we create a sorted set of the unique strings, why `TreeSet`?

```java
public int[] freqs(String[] data) {
    TreeSet<String> set = new TreeSet<>(Arrays.asList(data));
    // Simply return count of each string
}```
Efficiency for APTs?

- If we have M unique values from a set of N
  - This code is $O(MN)$, we can do better

```java
int[] ret = new int[set.size()];
int index = 0;
for(String s : set) {
    ret[index] = Collections.frequency(Arrays.asList(data), s);
    index++;
}
return ret;
```

- Extract values: `TreeMap<String, Integer>`
SortByFreqs

• TreeMap <String, Integer>, keys in order
  • Keys are in sorted order
  • What song will you use for green dance?
• [https://www.youtube.com/watch?v=XjVNlG5cZyQ](https://www.youtube.com/watch?v=XjVNlG5cZyQ)

```java
TreeMap<String, Integer> map = new TreeMap<>();
for (String s : data) {
    map.putIfAbsent(s, 0);
    map.put(s, map.get(s) + 1);
}
```
Using TreeMap and Map.Entry

- Map.Entry is like a pair class for maps
  - Use `Map.Entry<Key, Value>`
  - Getting this right is not pretty

```java
ArrayList<Map.Entry<String, Integer>> list =
    new ArrayList<>(map.entrySet());
Collections.sort(list,
    Map.Entry.comparingByValue(Comparator.reverseOrder()));
```

```cpp
ArrayList<Map.Entry<String, Integer>> list =
    new ArrayList<>(map.entrySet());
Collections.sort(list,
    Collections.reverseOrder(
        Comparator.comparing(
            Map.Entry::getValue)));
```
Using Lambda to sort in Java

• Perhaps more intuitive: sort strings using corresponding value in a map
  • Given string, find value
  • Leverage alphabetical order of strings and stable sort: "equal" keys don't change order, where equal is count of occurrences from map

```java
ArrayList<String> list = new ArrayList<>(map.keySet());
Comparator<String> comp =
    Comparator.comparingInt(s -> map.get(s));
Collections.sort(list, comp.reversed());
return list.toArray(new String[0]);
```