CompSci 201, L14: Sorting
Logistics, Coming up

- Project 3: DNA due today 10/17

- Project 4: Autocomplete will release tomorrow, 10/18, will be due Monday after next 10/31.

- APT Quiz 1 due this Wednesday 10/19
  - Takes 2 hours, finish by 11:59
  - No regular APTs this week, just the quiz
Mergesort

High level idea:
• Base case: size 1
  • Return list
• Recursive case:
  • Mergesort(first half)
  • Mergesort(second half)
  • …
Mergesort

High level idea:
• Base case: size 1
  • Return list
• Recursive case:
  • Mergesort(first half)
  • Mergesort(second half)
  • Merge the sorted halves
  • Return sorted
Runtime complexity of mergesort?

```
public static List<String> mergeSortList(List<String> list) {
    if (list.size() <= 1) {
        return list;
    }
    int mid = list.size()/2;
    List<String> firstHalfSorted = mergeSortList(list.subList(0, mid));
    List<String> secondHalfSorted = mergeSortList(list.subList(mid, list.size()));
    return merge(firstHalfSorted, secondHalfSorted);
}
```
Recursion tree

\[ T(N) = N + T(N / 2) + T(N / 2) \]

Depth of the recursion tree: Number of recursive calls before base case.

\[ \log_2(N) \]

Total complexity of each level across all of the recursive calls.

\[ T(N) = O(N \log N) \]
Sorting in Java: Comparable, Comparator
Java API Sort Algorithms

• Collections.sort (for a List)
• Arrays.sort (for an Array)

• Both implement *Timsort* a variant of Mergesort.
  • O(N log(N)), *nearly* linear runtime complexity.
  • Sorts in-place, mutates the input rather than return a new List/Array.
    • Collections.sort(myList); causes myList to be sorted.
  • Stable, does not reorder elements if not needed (e.g., if two elements are equal).
What can be compared and sorted in Java?

• Class objects that implement the `Comparable` interface. Has a natural order.

• Requires implementing a `.compareTo()` method

Should return an int:

• `< 0` if this comes before the parameter.

• `0` if this and the parameter are equal.

• `> 0` if this comes after the parameter.

```java
private static class Person implements Comparable<Person> {
    String first;
    String last;
    public Person(String s) {...}
    public String getLast() { return last; }
    public String getFirst() { return first; }
    public String toString() { return first + " " + last; }

    @Override
    public int compareTo(Person p){
        int diff = last.compareTo(p.last);
        if (diff != 0) return diff;
        return first.compareTo(p.first);
    }
}
```
Strings are Comparable

• What is the equivalent of < for Strings?
• Use the `compareTo` method for the natural lexicographic (dictionary/sorted) ordering.

```
  jshell> "a".compareTo("b");
  $30  =>  -1

  jshell> "b".compareTo("b");
  $31  =>  0

  jshell> "b".compareTo("a");
  $32  =>  1

  jshell> "az".compareTo("cb");
  $37  =>  -2
```

- Negative for “less than”
- Zero for “equal”
- Positive for “less than”
- Lexicographic, check first character, second if equal, third if still equal, ...
Comparable for other classes?

• Can implement Comparable interface and compareTo method when defining your own class.
  • Defines a natural ordering, can sort
  • Collections.sort, Arrays.sort

• What if it’s someone else’s class and they didn’t implement Comparable?

• Or what if you want to sort by something other than the “natural order” defined by Comparable?
Comparable vs. Comparator

- `a.compareTo(b)`
  - What is method signature? One parameter
  - Method in class of which object `a` is an instance
  - `a` is `this`, `b` is a parameter

- Create a Comparator `c`, use `c.compare(a,b)`
  - Method has two parameters
  - Part of `Comparator` (Java API link)
  - Returns an int:
    - `< 0` (means `a` comes before `b`)
    - `== 0` (means `a` equals `b`)
    - `> 0` (means `a` comes after `b`)
java.util.Comparator Examples

- **Comparator.naturalOrder**

  ```java
  jshell> Comparator<String> c = Comparator.naturalOrder()
  c ==> INSTANCE
  
  jshell> c.compare("a","b")
  $12 ==> -1
  
  jshell> c.reversed().compare("a","b")
  $13 ==> 1
  ```

- **Comparator.comparing**

  ```java
  jshell> Comparator<String> c = Comparator.comparing(String::length)
  c ==> java.util.Comparator$Lambda$27/0x000000800b97c462b71fc7e
  
  jshell> c.compare("this", "is")
  $15 ==> 1
  
  jshell> c.compare("is", "it")
  $16 ==> 0
  ```

  Syntax is: `<Type>::<method name>` to sort something of the Type by the result of some getter method that returns something Comparable.
Sorting Comparable objects by naturalOrder

["sloth", "house", "owl", "ant", "mice", "kelp"]
["ant", "house", "kelp", "mice", "owl", "sloth"]

• naturalOrder for Strings is lexicographic (alphabetical or dictionary order)

```java
String[] a = {"sloth", "house", "owl", "ant", "mice", "kelp"};
System.out.println(Arrays.toString(a));

String[] copy = Arrays.copyOf(a, a.length);
Arrays.sort(copy);
System.out.println(Arrays.toString(copy));
```
Comparator to sorting strings by length

[sloth, house, owl, ant, mice, kelp]
[owl, ant, mice, kelp, sloth, house]

• Why does "owl" come before "ant"?
  • Stable sort respects order of equal keys

```java
    copy = Arrays.copyOf(a, a.length);
    Arrays.sort(copy, Comparator.comparing(String::length));
    System.out.println(Arrays.toString(copy));
```
Stable, Stability

• Respect order of equal keys when sorting
  • First sort by shape, then by color: Stable!
    • Triangle < Square < Circle; Yellow < Green < Red
Sorting by length and then naturally

[sloth, house, owl, ant, mice, kelp]
[ant, owl, kelp, mice, house, sloth]

- Could put this on one line, easier to read?
  - First sort by length, if same? naturally

Arrays.sort(copy, Comparator.
  comparing(String::length).
  thenComparing(Comparator.naturalOrder()));
Comparator with “lambdas”

• What if you want to sort by something that doesn’t have a getter method?

• Or you just want to do something different than sort by a getter method?

• Can also define a comparator with a “lambda” expression.

• Example: Sort an Array of Arrays

```java
int[] ar1 = {2, 0, 1};  // int[] ar2 = {1, 0, 2, 3};
int[][] bothAr = {ar1, ar2};
```
Comparator with lambda to sort Array of Arrays by first elements

```java
type ar1 = {2, 0, 1};    int[] ar2 = {1, 0, 2, 3};
int[][] bothAr = {ar1, ar2};

{ int[3] { 2, 0, 1 }, int[4] { 1, 0, 2, 3 } }

Comparator<int[][]> comp =
    (a, b) -> (a[0] - b[0]);
```

Type we want to compare

Given an a and a b of that type...

comp.compare(a,b) should return this expression
Comparator with lambda to sort Array of Arrays by first elements

```java
int[] ar1 = {2, 0, 1};    int[] ar2 = {1, 0, 2, 3};
int[][] bothAr = {ar1, ar2};

{ int[3] { 2, 0, 1 }, int[4] { 1, 0, 2, 3 } }

Comparator<int[]> comp =
    (a, b) -> (a[0] - b[0]);

comp.compare(ar1, ar2) returns (2-1) = 1.
comp.compare(ar1, ar1) returns (2-2)=0
comp.compare(ar2, ar1) returns (1-2)=-1
```
Comparator with lambda to sort Array of Arrays by first elements

```java
int[] ar1 = {2, 0, 1};    int[] ar2 = {1, 0, 2, 3};
int[][] bothAr = {ar1, ar2};

    { int[3] { 2, 0, 1 }, int[4] { 1, 0, 2, 3 } }

Comparator<int[][]> comp =
    (a, b) -> (a[0] - b[0]);

Arrays.sort(bothAr, comp);
    { int[4] { 1, 0, 2, 3 }, int[3] { 2, 0, 1 } }
```
Go to duke.is/pra6v

Not graded for correctness, just participation.

Try to answer without looking back at slides and notes.

But do talk to your neighbors!
Summary: Sorting Blobs

```java
public class Blob implements Comparable {
    public String name;
    public String shape;
    public String color;

    public Blob(String n, String c, String s) {
        name = n;
        shape = s;
        color = c;
    }

    public String getShape() {
        return shape;
    }

    @Override
    public int compareTo(Object o) {
        Blob other = (Blob) o;
        return name.compareTo(other.name);
    }
}
```
Sorting Blobs many ways

```java
List<Blob> myBlobs = ...;

// Sorting by natural order (names)
Collections.sort(myBlobs);
System.out.println(myBlobs);

[(al, round, red), (bo, square, blue), (cj, triangle, green), (di, square, red)]

// Sorting by shape with Comparator using getShape method
Comparator<Blob> comp = Comparator.comparing(Blob::getShape);
Collections.sort(myBlobs, comp);
System.out.println(myBlobs);

[(al, round, red), (bo, square, blue), (di, square, red), (cj, triangle, green)]
```
Sorting Blobs even more ways

25  // Sorting by color with Comparator using lambda
26  Comparator<Blob> comp2 = (b1, b2) -> (b1.color.compareTo(b2.color));
27  Collections.sort(myBlobs, comp2);
28  System.out.println(myBlobs);

[(bo, square, blue), (cj, triangle, green),
 (al, round, red), (di, square, red)]

30  // Sorting by with multiple comparators
31  Collections.sort(myBlobs, comp.thenComparing(comp2));
32  System.out.println(myBlobs);

[(al, round, red), (bo, square, blue),
 (di, square, red), (cj, triangle, green)]
Algorithms using sortedness
Binary Search

• Given a **sorted** list of N elements and a target, in just \( O(\log(N)) \) time, return:
  • Index \( i \) such that \( \text{list.get}(i) \) equals target, or
  • -1 if target not in list

• Example:
  • If we search for ‘h’, should return 4
  • If we search for ‘c’, should return -1

<table>
<thead>
<tr>
<th>value</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘d’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘j’</th>
<th>‘k’</th>
<th>‘m’</th>
<th>‘p’</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Java API Binary Search

Arrays.binarySearch (for arrays) and Collections.binarySearch (for Lists).

```java
String[] ar = {"ape", "bird", "cat", "dog", "elephant", "ferret", "gecko", "hippo"};

int index = Arrays.binarySearch(ar, "cat");

Returns 2
```

Careful, does not check for you if list is sorted!

```java
String[] ar = {"cat", "ape", "bird", ...}

int index = Arrays.binarySearch(ar, "cat");

Returns -4
```
Java API Binary Search with Comparator

Can pass a comparator comp, in which case:
1. Array/List should be sorted by that comp, and
2. Want an index $i$ such the $i$'th element $e_i$ has $\text{comp.compare}(e_i, \text{target})==0$.

```java
[ape, cat, dog, bird, gecko, hippo, ferret, elephant]

Comparator<String> comp =
    Comparator.comparing(String::length);

index = Arrays.binarySearch(ar, "dog", comp);
```

Sorted by length

Returns 1. $\text{comp.compare}(\text{"cat"}, \text{"dog"})==0$
How is Binary Search $O(\log(N))$?

• How to find something in a list of $N$ elements without looping over the list?

• Let $low$ (initially 0) and $high$ (initially $N-1$) mark the limits of the active search space.

• Want to cut down the search space by half at each step:

<table>
<thead>
<tr>
<th>value</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘d’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘j’</th>
<th>‘k’</th>
<th>‘m’</th>
<th>‘p’</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

$N$ $N/2$ $N/4$ $N/8$ ...

$1$

$log_2(N)$ steps!
Binary Search in Pictures

- Searching for ‘d’ in

<table>
<thead>
<tr>
<th>value</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘d’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘j’</th>
<th>‘k’</th>
<th>‘m’</th>
<th>‘p’</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

- ‘h’ > ‘d’, so need to keep searching in the lower half.
- Set high = mid - 1;
Binary Search in Pictures

• Searching for ‘d’ in

<table>
<thead>
<tr>
<th>value</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘d’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘j’</th>
<th>‘k’</th>
<th>‘m’</th>
<th>‘p’</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

- ‘b’ < ‘d’, so need to keep searching in the upper half.
- Set low = mid+1;
Binary Search in Pictures

• Searching for ‘d’ in

<table>
<thead>
<tr>
<th>value</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘d’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘j’</th>
<th>‘k’</th>
<th>‘m’</th>
<th>‘p’</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Mid = (low + high) / 2

• ‘d’ equals ‘d’, return mid (2)