Lecture 10: Iterators

CS 62
Spring 2018
Alexandra Papoutsaki & William Devanny
Friday Quiz

• Sorting, big-O,
• Iterators
• Set up induction
Sorting (in case you forgot)

Selectionsort
   Find largest (smallest) element, put at end (beginning), sort rest
   Complexity: $O(n^2)$
   In-place

Mergesort:
   Divide in half, sort each half, then merge them in order
   Complexity: $O(n \log n)$
   $O(n)$ extra space to merge into
Quicksort (a quick intro)

Divide & conquer
1. Pick a pivot (different techniques: first, last, random, median, etc)
2. Move smaller elements to left of pivot, larger to right
3. Recursively sort each of the smaller lists
4. Make one big list

Complexity: $O(n \log n)$ on average, $O(n^2)$ in worst case
Which one should I use?

You already know the answer... It depends

• If small list then selection/insertion (less overhead)
• If large list and need to always run quickly then merge sort (but needs extra space)
• If large list, need to run fast on average, but being occasional slow is OK, then quick sort.

• [https://www.toptal.com/developers/sorting-algorithms/](https://www.toptal.com/developers/sorting-algorithms/)
Collections and Iterators

A Collection represents a group of objects known as its elements.

Iterator: Object to traverse through a collection one element at a time. Implemented as interface in Java.
public interface Iterator<E> {
    // returns true if the iteration has more elements
    boolean hasNext();

    // returns the next element in the iteration
    E next();

    // removes the last element that was returned by next
    void remove(); // optional
}
Iterator rules

• remove is optional (we won’t use it)
• Only allowed to call remove once and then must terminate iteration.
• Never change a collection in middle of an iteration
  • Behavior is officially undefined if you do
  • Iterator often copies data structure before iterating, so changes may not appear to original!
Iterator example

Iterator listIterator = myList.iterator();
while(listIterator.hasNext()){  
    System.out.println(listIterator.next());
}

while(listIterator.hasNext()){  
    String elt = listIterator.next();  
    System.out.println(elt);  
}
Iterable

Interface which when implemented allows for-each loop

Includes `Iterator<T> iterator()`

Example: `ArrayList<E>`

See definition and use of `iterator` in `ArrayIndexList`. Often implemented by inner class.
For-each loop

for(String elt: myList){
    System.out.println(elt);
}

• Abbreviates previous code
• Fine as long as myList has an iterator method
• Called an active or external iterator.

• Cannot modify the collection
Iterable

- Notice can have two iterators going through list independently!

- Never modify a data structure when iterating through elements as may get unpredictable results

- Most classes in Java collection classes have iterators which are designed to “fail fast”.
  - Throw an exception if simultaneous access and modification
Java 8

• Read Iterating over collections in Java 8
• forEach() method now in collection classes
  • public void forEach(Consumer action)
  • Internal iterator
• Description copied from interface Iterable

Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception. Unless otherwise specified by the implementing class, actions are performed in the order of iteration (if an iteration order is specified). Exceptions thrown by the action are relayed to the caller.
Using `forEach()`

```java
myList.forEach(elt -> {System.out.println(elt);});
```

- No explicit control over iterator
- Similar to for-each loop, but it is a method of data structure
- `Consumer` is an interface with method `void accept(T t)`
- `accept` method has code to be executed
- Most valuable when more than one way to traverse
- May only access effectively final variables from scope
• Method definition:

```java
public void forEach(Consumer<? super E> action) {
    for (E elt: this)
        action.accept(elt);
}
```

• `forEach()` is “default method” of `Iterable` interface.
• Automatically inherited in all classes implementing it.
• See article for restrictions on default methods – can’t access instance variables!