COMPARATORS + ITERATORS

David Kauchak
CS 62 – Spring 2021

1

Admin

Compression assignment

2

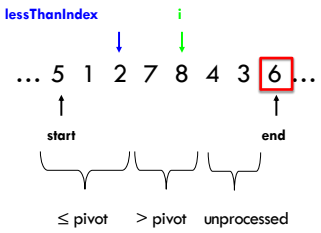
Partition

```
public static <E extends Comparable<E>> int partition(E[] vals, int start, int end){
    int lessThanIndex = start-1;

    for( int i = start; i < end; i++ ){
        if( vals[i].compareTo(vals[end]) <= 0 ) {
            lessThanIndex++;
            swap(vals, lessThanIndex, i);
        }
    }

    swap(vals, lessThanIndex+1, end);
    return lessThanIndex+1;
}
```

3



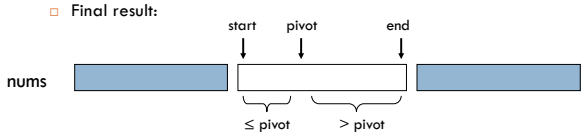
4

```
public static <E extends Comparable<E>> int partition(E[] vals, int start, int end){
    int lessThanIndex = start-1;

    for( int i = start; i < end; i++ ){
        if( vals[i].compareTo(vals[end]) <= 0 ) {
            lessThanIndex++;
            swap(vals, lessThanIndex, i);
        }
    }

    swap(vals, lessThanIndex+1, end);
    return lessThanIndex+1;
}
```

- vals[end] is called the ***pivot***
- Partitions the elements nums[start…end-1] in to two sets, those ≤ pivot and those > pivot
- Operates in place
- Final result:



5

Partition running time?

O(n)

```
public static <E extends Comparable<E>> int partition(E[] vals, int start, int end){
    int lessThanIndex = start-1;

    for( int i = start; i < end; i++ ){
        if( vals[i].compareTo(vals[end]) <= 0 ) {
            lessThanIndex++;
            swap(vals, lessThanIndex, i);
        }
    }

    swap(vals, lessThanIndex+1, end);
    return lessThanIndex+1;
}
```

6

Quicksort

```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

```
public static <E extends Comparable<E>> int partition(E[] vals, int start, int end){
    int lessThanIndex = start-1;

    for( int i = start; i < end; i++ ){
        if( vals[i].compareTo(vals[end]) <= 0 ) {
            lessThanIndex++;
            swap(vals, lessThanIndex, i);
        }
    }

    swap(vals, lessThanIndex+1, end);
    return lessThanIndex+1;
}
```

7

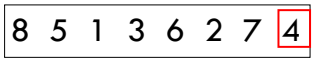
Quicksort

8	5	1	3	6	2	7	4

```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

8

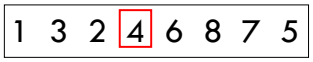
Quicksort




```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

9

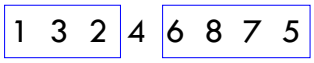
Quicksort




```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

10

Quicksort




```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

11

Quicksort


```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

12

Quicksort


```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

13

Quicksort


```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

14

Quicksort


```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

15

Quicksort


```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

16

Quicksort

1 2 3 4 6 8 7 5

```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

17

Quicksort

1 2 3 4 5 8 7 6

What happens here?

```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

18

Quicksort

1 2 3 4 5 8 7 6

```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

19

Quicksort

1 2 3 4 5 8 7 6

```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

20

Quicksort

1 2 3 4 5 6 7 8

```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

21

Quicksort

1 2 3 4 5 6 7 8

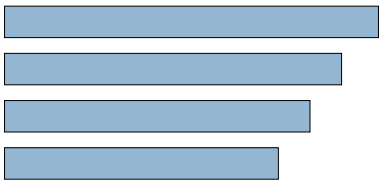
```
void quicksortHelper(E[] vals, int start, int end){
    if( start < end ){
        int partition = partition(vals, start, end);
        quicksortHelper(vals, start, partition-1);
        quicksortHelper(vals, partition+1, end);
    }
}
```

22

Running time of Quicksort?

Worst case?

Each call to Partition splits the array into an empty array and n-1 array



23

Quicksort: Worst case running time

$n-1 + n-2 + n-3 + \ldots + 1 = O(n^2)$

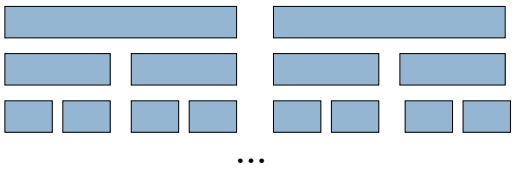
When does this happen?

- sorted
- reverse sorted
- near sorted/reverse sorted

24

Quicksort best case?

Each call to Partition splits the array into two equal parts



…

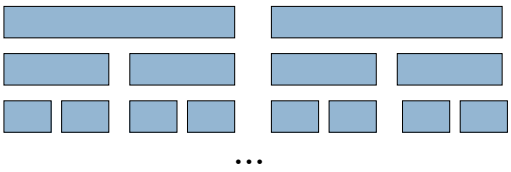
How much work is done at each "level", i.e. running time of a level?

O(n)

25

Quicksort best case?

Each call to Partition splits the array into two equal parts



…

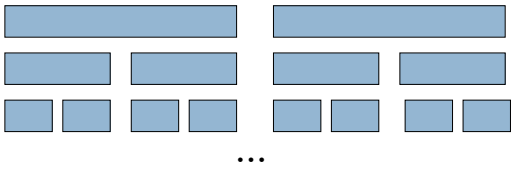
How many levels are there?

Similar to mergesort, each call to Partition will throw away half the data until we're down to one element: $\log_2 n$ levels

26

Quicksort best case?

Each call to Partition splits the array into two equal parts



…

Overall runtime?

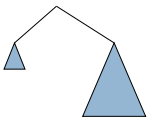
O(n log n)

27

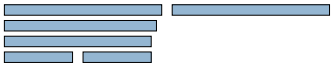
Quicksort Average case?

Two intuitions

- As long as the Partition procedure always splits the array into some constant ratio between the left and the right, say L-to-R, e.g. 9-to-1, then we maintain O(n log n)



- As long as we only have a constant number of "bad" partitions intermixed with a "good partition" then we maintain O(n log n)



28

How can we avoid the worst case?

Inject randomness into the data

```
void randomizedPartition(E [] nums, int start, int end){
  int i = randomInt(start, end);
  swap(nums, i, end);
  return partition = partition(nums, start, end);
}
```

Randomized quicksort is average case O(n log n)

29

What is the worst case running time of randomized Quicksort?

$O(n^2)$

We could still get very unlucky and pick "bad" partitions at every step

30

Quicksort properties

Stable?

In-place?

31

Quicksort properties

Stable: possible, but not the way we've written it (and requires more storage!)

In-place: yes!

32

Sorting summarized

	in-place?	stable?	Best	Average	Worst	Notes
Selection	X		$O(n^2)$	$O(n^2)$	$O(n^2)$	n swaps
Insertion	X	X	O(n)	$O(n^2)$	$O(n^2)$	use for partially ordered
Merge		X	O(n log n)	O(n log n)	O(n log n)	guaranteed
Quick	X		O(n log n)	O(n log n)	$O(n^2)$	fastest in practice

33

Comparable interface

https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

Interface Comparable<T>

int compareTo(T other)

- -1: this object is less than other (technically, any negative number)
- 0: this object is equal to other
- 1: this object is greater than other (technically, any positive number)

34

Built-in sorting

Arrays: (https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html)

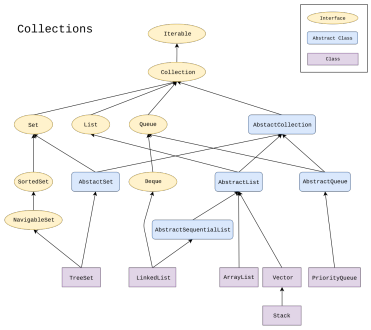
static void	sort(Object[] a) Sorts the specified array of objects into ascending order, according to the **natural ordering** of its elements.
static void	sort(Object[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of objects into ascending order, according to the **natural ordering** of its elements.

Collections:(https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html)

static <T extends Comparable<? super T>> void	sort(List<T> list) Sorts the specified list into ascending order, according to the **natural ordering** of its elements.

35

Collections



36

Naturally sorting cards

https://github.com/pomonacs622021sp/LectureCode/blob/master/SortingCards/SortableCard.java

SortableCard:

- implements Comparable<SortableCard>
- Utilizes String.compareTo and Integer.compare
- Foreach loop!

naturalSort()

37

Comparator: unnatural sorting

https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html

Create a different ordering **without having to modify the class!**

Interface Comparator<T>

int compare(T o1, T o2)
-1: o1 is less than o2 (technically, any negative number)
0: o1 is equal to o2
1: o1 is greater than o2 (technically, any positive number)

38

Unnatural sorting

Arrays: (https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html)

static <T> void	sort(T[] a, Comparator<? super T> c) Sorts the specified array of objects according to the order induced by the specified comparator.
static <T> void	sort(T[] a, int fromIndex, int toIndex, Comparator<? super T> c) Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.

Collections:(https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html)

static <T> void	sort(List<T> list, Comparator<? super T> c) Sorts the specified list according to the order induced by the specified comparator.

39

Unnaturally sorting cards

https://github.com/pomonacs622021sp/LectureCode/blob/master/SortingCards/BridgeCardSort.java

- Add 20 to aces
- Reverse the suit ordering
- Reverse the number ordering

bridgeOrderingSort

40

Iterator

https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

A way to move through all of the data in a collection

Interface Iterator<E>:

- boolean hasNext()
- E next()

Have we seen this before? How can we iterate through the data?

41

Iterator

```
while( dealer.hasNext() ) {
    Card c = dealer.next();
    System.out.println(c);
    cards.add(new SortableCard(c.getNumber(), c.getSuit()));
}
```

42

Iterator example

```
public static void temp(){
    List<String> list = new ArrayList<String>();

    list.add("bananas");
    list.add("taste");
    list.add("good");

    Iterator<String> iterator = list.iterator();

    while( iterator.hasNext() ) {
        System.out.println(iterator.next());
    }
}
```

What would we see printed?

43

Iterator example

```
public static void temp(){
    List<String> list = new ArrayList<String>();

    list.add("bananas");
    list.add("taste");
    list.add("good");

    Iterator<String> iterator = list.iterator();

    while( iterator.hasNext() ) {
        System.out.println(iterator.next());
    }
}
```

bananas
taste
good

44

Iterator example

```
public static void temp2(){
    List<String> list = new ArrayList<String>();

    list.add("bananas");
    list.add("taste");
    list.add("good");

    Iterator<String> iterator = list.iterator();
    Iterator<String> iterator2 = list.iterator();

    System.out.println(iterator2.next());

    while( iterator.hasNext() ) {
        System.out.println(iterator.next());
    }

    System.out.println(iterator2.next());
}
```

What would we see printed?

45

Iterator example

```
public static void temp2(){
    List<String> list = new ArrayList<String>();

    list.add("bananas");
    list.add("taste");
    list.add("good");

    Iterator<String> iterator = list.iterator();
    Iterator<String> iterator2 = list.iterator();

    System.out.println(iterator2.next());

    while( iterator.hasNext() ) {
        System.out.println(iterator.next());
    }

    System.out.println(iterator2.next());
}
```

bananas
bananas
taste
good
taste

Each iterator has its own state!

46

Iterable

https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html

interface Iterable<E>:

- Iterator<E> iterator()

Just a single method that returns an Iterator.

47

Why Iterable??

```
for( SortableCard c: cards ) {
    System.out.println(c);
}
```

Any class that implements the Iterable class can be used in a foreach loop!

48

How to make a class Iterable

- Implement Iterable interface
- Make a private inner class that implements the Iterator interface
- Have the iterator method return an instance of the private inner class

49

An example

https://github.com/pomonacs622021sp/LectureCode/blob/master/Iterable/IterableArrayList.java

Each instance of the inner class will have its own *count* instance variable

50

Iterator vs. Iterable

Iterators are a useful mechanism for iterating over almost any type of data

Iterators are the thing that do most of the work (and require most of the coding!)

Iterable allows us to use it in a foreach loop and is often just creating an instance of an Iterator

51