Lec 15: Interfaces

Sam McCauley

March 16, 2026

Admin

- Last week before break!
- Lab on Wednesday
  - In-person
  - (I'll take attendance this week; required to do these labs in person)
- Taco Tuesday tomorrow at noon in the CS department!

Asymptotics Examples

indexOf

```
1  public int indexOf(E element) {
2      for (int index = 0; index < numElems; index++) {
3          if (element.equals(arr[index])) {
4              return index;
5          }
6      }
7      return -1;
8  }
```

How long does this code take on a list of length n? $O(n)$

Find Duplicate in an array?

- In pairs: Given an array of `Students`, how can we find if two have the same name?
- Plan:
  - Go through each `Student`
  - See if the *i*th `Student` has the same name as the *j*th `Student` for $j > i$

hasDuplicate

```java
public static boolean hasDuplicate(Student[] course) {
  for (int i = 0; i < course.length; i++) {
    for (int j = i + 1; j < course.length; j++) {
      if (course[i].getName().equals(course[j].getName())) {
        return true;
      }

    }
  }
  return false;
}
```

How long does this code take on an array of length n? $O(n^2)$

isPalindrome Linked List

```
1  public boolean isPalindrome() {
2      Node<E> forward = head;
3      Node<E> backward = tail;
4      for(int i = 0; i < size(); i++) {
5          if(!forward.getData().equals(backward.getData())) {
6              return false;
7          }
8          forward = forward.getNext();
9          backward = backward.getPrevious();
10     }
11     return true;
12 }
```

How long does this code take on a linked list of length n? $O(n)$

isPalindrome Linked List With Get

```
1  public boolean isPalindrome() {
2      for(int i = 0; i < size(); i++) {
3          if(!get(i).equals(get(size() - i - 1))) {
4              return false;
5          }
6      }
7      return true;
8  }
```

How long does this code take on a linked list of length n? Each `get()` is $O(n)$ time and there are n of them. So: $O(n^2)$

Binary Search

```
1  private static int binarySearch(ArrayList<Integer> list, int
       target, int left, int right) {
2          if (left > right) {
3              return -1;
4          }
5          int mid = (left + right) / 2;
6          int midValue = list.get(mid);
7          if (midValue == target) {
8              return mid; // exact match
9          }
10         else if (target < midValue) {
11             return binarySearch(list, target, left, mid - 1);
12         }
13         else {
14             return binarySearch(list, target, mid+1, right);
15         }
16 }
```

How long does this code take on a list of length *n*?

Analyzing Binary Search

- We start out with an array of length n
- Each time the method is called, we do $O(1)$ work and call a method half the size
- So we call the method once for $n, n/2, n/4, n/8, \ldots, 1$
- How many method calls is that?
- Want $n/2^k = 1$, so $2^k = n$. This means that $k = \log_2 n$
- $O(\log_2 n)$ time

Interfaces

A Way to Group Classes

- We wrote code to find two `Students` with duplicate names
- How would this code change if rather than two `Students` with duplicate names, we wanted to find two `Employees` with duplicate names?
    - (Assuming both classes have a `public String getName()` method)

hasDuplicate

```java
public static boolean hasDuplicate(Student[] arr) {
    for (int i = 0; i < arr.length; i++) {
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[i].getName().equals(arr[j].getName())) {
                return true;
            }

        }
    }
    return false;
}
```

hasDuplicate

```java
public static boolean hasDuplicate(Employee[] arr) {
    for (int i = 0; i < arr.length; i++) {
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[i].getName().equals(arr[j].getName())) {
                return true;
            }

        }
    }
    return false;
}
```

Changes to hasDuplicate

- Literally nothing changed except the name of the type of the parameter
- Do we really need create a new method just to re-specify the class? (And then copy-paste it for *every* new class with a `getName()` method where we want to find a duplicate?)
- What we want: we don't really care what the objects in the array are, so long as they are of a class that has a `getName()` method
- In other words: want a guarantee on the *methods implemented* by a class

Interface

- Way to specify a *required set of methods* for a class
    - Can think of it like a contract: we're required to implement these methods.
- The `Interface` is specified in its own file, similarly to a class
- When you write a class, if it has the appropriate methods, you may say that it `implements` the `Interface`.
- Then, you can refer to elements of the class using the interface name, rather than the name of the specific class
- Let's look at an example

Interface Details

- While we can make a variable of `Interface` type, must instantiate a *specific* object (using a class)

```
1       Named newNamed = new Student(); //works
2       Named newNamed = new Named();   //does not work!
```

- What exactly are we saying here?
  - On the right: we're making a new `Student`. This specifies what constructor to call; what data to fill in
  - On the left: this variable points to an object with a `getName()` method, but I'm making no other guarantees
    - Can call `newNamed.getName()`, but cannot call `newNamed.getID()` even though it exists

More on Interfaces

- Do require javadoc comments
- Can use generics
- Can (and often do) have more than one method
- An `ArrayList<E>`, `LinkedList<E>`, both implement the `List<E>` interface
  - Methods include: `add()`, `remove()`, `get()`, `indexOf()`, and so on
  - Let's look at the code
- A single class can implement multiple `Interfaces`

Sorting

How can we sort a set of cards?

- Goal: sequence of steps
- Guarantee that the cards are sorted at the end
- We want to be able to:
  - Code it up in Java
  - Analyze the running time

Specifics

- Have an array of numbers
- Want to sort them *in place* (without copying to a new array)
    - In other words: sort them using $O(1)$ extra space.

Where to Start?

10	21	-3	40	17	13	11	-4

- Is there any number we can put directly in the correct place?
- We can put the largest number in the last slot
- Scan through the array to find the maximum number
  - Time?
  - $O(n)$
- *Swap* that number with the last number

10	21	-3	40	17	13	11	-4

Maximum so far: 10 at pos 0 21 at pos 1 40 at pos 3

Where to Start?

10	21	-3	40	17	13	11	-4

- Is there any number we can put directly in the correct place?
- We can put the largest number in the last slot
- Scan through the array to find the maximum number
  - Time?
  - $O(n)$
- *Swap* that number with the last number

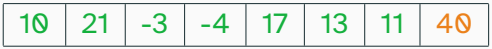
10	21	-3	-4	17	13	11	40

Now what?

- Do it again! But now on all but the last element of the array

- (This is essentially a recursive algorithm)

Selection Sort



Maximum so far: 10 at pos 0 21 at pos 1

Selection Sort

Maximum so far: 10 at pos 0 11 at pos 1 17 at pos 4

Selection Sort

Let’s look at the code

- We’ll do loops, not recursion
- Let’s assume we have a `swap(int[], int, int)` method that swaps two indices of an array

```
1  public void swap(int[] arr, int index1, int index2) {
2      int temp = arr[index1];
3      arr[index1] = arr[index2];
4      arr[index2] = temp;
5  }
```

Selection Sort Code

```
1  public static void selectionSort(int data[]) {
2      int numUnsorted = data.length;
3      int index; // general index
4      int max; // index of largest value
5      while (numUnsorted > 0) {
6          // determine maximum value in array
7          max = 0;
8          for (index = 1; index < numUnsorted; index++) {
9              if (data[max] < data[index]) {
10                 max = index;
11             }
12         }
13         swap(data, max, numUnsorted-1);
14         numUnsorted--;
15     }
16 }
```

Selection Sort Code: Method Version

```
1  public static int findMax(int data[], int maxIndex) {
2      int max = 0;
3      for (int index = 1; index < maxIndex; index++) {
4          if (data[max] < data[index]) {
5              max = index;
6          }
7      }
8      return max;
9  }
10
11 public static void selectionSort(int data[]) {
12     int numUnsorted = data.length;
13     while (numUnsorted > 0) {
14         int max = findMax(data, numUnsorted);
15         swap(data, max, numUnsorted-1);
16         numUnsorted--;
17     }
18 }
```

Why Does This Work?

- Idea: after the loop iterates i times,
  - The last i slots of the array contain the i largest elements of the array in sorted order
- When $i = n$ we are done

Using Generics for Selection Sort

Generalizing our Sort

- We wrote selection sort code that only works on an array of `ints`
- Can we use the power of interfaces to write code that will work for a wide variety of objects?
- What method(s) do we need the objects to have to use selection sort?

Selection Sort Code

```
1  public static void selectionSort(int data[]) {
2      int numUnsorted = data.length;
3      int index; // general index
4      int max; // index of largest value
5      while (numUnsorted > 0) {
6          // determine maximum value in array
7          max = 0;
8          for (index = 1; index < numUnsorted; index++) {
9              if (data[max] < data[index]) {
10                 max = index;
11             }
12         }
13         swap(data, max, numUnsorted-1);
14         numUnsorted--;
15     }
16 }
```

The only thing we ever do with an element of `data` is see if `data[max] < data[index]`

`Comparable<T>` Interface

- This is a Java interface (Built-in; don't need to write a file, or even import anything.)
- `Comparable<T>` has only one method:
  `public int compareTo(T other)`
  - Returns a negative integer if this object is smaller than the argument; `0` if equal; a positive integer if larger
- Let's add a `compareTo()` method, and then tell Java that our `Student` class implements this interface
- `Integer` already implements `Comparable<Integer>` so we can already sort `Integers`; same with `Strings`

Creating a generic sorting method

- We could make the `SortTest` class generic.
- Really: want to make one method generic. Can we do this in Java?
- Yes! Looks something like this:
  - `public static <E> void SelectionSort(ArrayList<E> list)`
- Problem: can't use *any* E. Needs to be comparable with other objects of type E

Generic Upper Bounds

- Way to tell Java that a generic type needs to meet certain requirements
- That way, at compile time, Java can make sure our types match up
- These are called *upper bounds*
- Let's say we only want to accept objects that meet the requirements of the `List` interface. Rather than `<E>`, we write something like `<E extends List>`
  - (Yes, it's `extends` and not `implements`. There are some good back-end reasons for this.)
- What do we want for our `insertionSort` method?
  - Want `<E extends Comparable<E>>`
  - That is to say: we want a type `E` that implements `Comparable<E>`. That is to say: need that objects of type `E` have a `compareTo` method that takes objects of type `E` as argument

Generic Selection Sort Code

```
1  public static <E extends Comparable<E> > void selectionSort(
       ArrayList<E> data) {
2      int numUnsorted = data.size();
3      int index; // general index
4      int max; // index of largest value
5      while (numUnsorted > 0) {
6          // determine maximum value in array
7          max = 0;
8          for (index = 1; index < numUnsorted; index++) {
9              if (data.get(max).compareTo(data.get(index)) < 0)
                   {
10                 max = index;
11             }
12         }
13         swap(data, max, numUnsorted-1); //assume swap method
               for ArrayList<E>
14         numUnsorted--;
15     }
16 }
```

Where we are

- Can sort any object so long as it implements `Comparable<E>`
- What are the downsides of this?
    - What if we want to sort objects that aren't already comparable and we don't want to modify the class?
    - Can only sort objects one way. (What if we want to sort `Students` by grade? Would need to rewrite the `Student` class!)
- There are upsides as well; we'll come back to this after we talk about `Comparators`

Insertion Sort

Insertion Sort

- Similar to Selection Sort
- Significantly more efficient in practice (we'll come back to this)
- This time we'll start with why it works, and derive the algorithm

Insertion Sort

- A different approach to sorting
- After the kth loop, the first k items in the array are sorted
  - The first k items may not be the smallest—but they are in sorted order
- How can we guarantee this for $k = 1$?
  - Don't need to do anything
- Let's say it works for k. What does the $k + 1$st loop need to accomplish to maintain the invariant?

  - Needs to insert the $k + 1$st item among the first k items in sorted order.

A Beautiful Way to Accomplish This

- Want to take the new item and move it into sorted position
- Idea: need to move it down until the previous element is smaller
- Inner loop: store element we are trying to insert. Shift elements down while it is smaller.

Insertion Sort Code

```
1  public static void insertionSort(int data[]) {
2      int numSorted = 1; // number of values in place
3      int index; // general index
4      while (numSorted < data.length) {
5          int temp = data[numSorted]; // first unsorted value
6          fo                                              {
7
8
9              } else {
10                 break;
11             }
12         }
13         data[index] = temp; // reinsert value
14         numSorted++;
15     }
16 }
```

Can we get rid of the `break` command in this code?

Insertion Sort Code # 2

```
1  public static void insertionSort(int data[]) {
2      int numSorted = 1; // number of values in place
3      while (numSorted < data.length) {
4          int temp = data[numSorted]; // first unsorted value
5          int index = numSorted;
6          while(index > 0 && temp < data[index - 1]) {
7              data[index] = data[index-1];
8              index--;
9          }
10         data[index] = temp; // reinsert value
11         numSorted++;
12     }
13 }
```

Tradeoff with Selection Sort

- No swap method needed
- Code is a little shorter
- Efficiency?
  - Both take n iterations of the outer loop. What about the inner loop?
  - Selection sort *always* iterates through $n - i$ elements on the ith iteration
  - Insertion sort may stop early! Can lead to better performance in practice (and is never worse)
- To be clear: both are still $O(n^2)$ in terms of worst-case performance. Insertion sort just has better constants, and better best-case performance