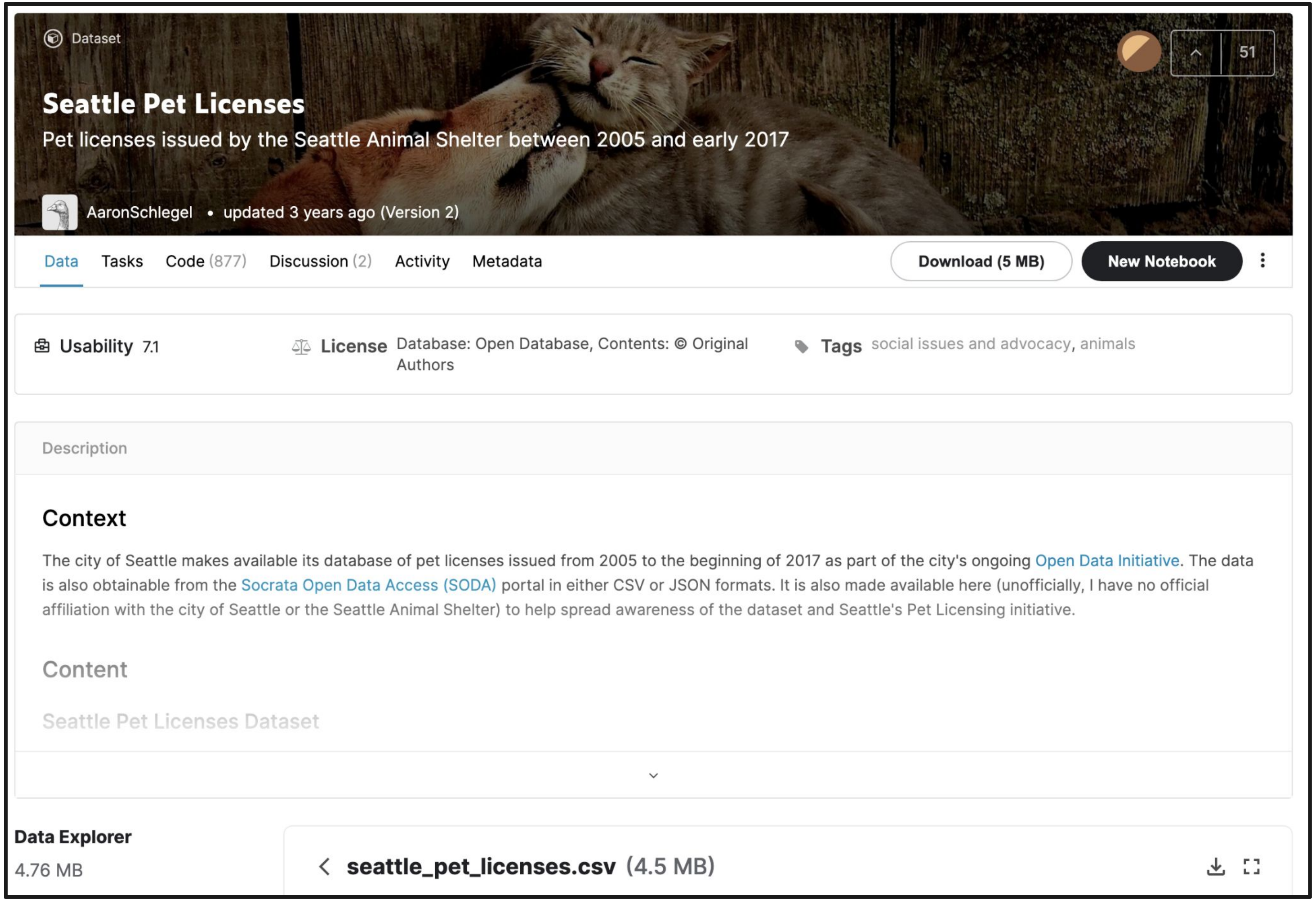
Lecture 13

Sorting III

- Lab 4 – Preview
- Generalized Sorting
  - Generics
  - Comparators
- Midterm Review
  - Problem 3
  - Problem 4

Lab 4 – Preview



Seattle Pet Licenses in .csv (comma separated values) format.

~/GitEvolene — -bash SeattlePets.csv

```
December 18 2015,S107948,Zen,Cat,Domestic Longhair,Mix,98117
June 14 2016,S116503,Misty,Cat,Siberian,,98117
June 16 2016,S116742,Frankie Lee Jones,Cat,Domestic Shorthair,,
August 04 2016,S119301,Lyra,Cat,Mix,,98121
November 08 2016,73859,Minnie,Cat,Domestic Shorthair,,98106
April 11 2017,133966,Charlie,Cat,Domestic Shorthair,,98102
June 06 2017,208553,Puck,Cat,Russian Blue,,98107
June 10 2017,142312,Frankie,Cat,Domestic Shorthair,,98107
September 22 2017,730786,Pepper,Cat,Domestic Shorthair,,98146
September 26 2017,273798,Cathol,Cat,Domestic Shorthair,,98108
January 19 2018,895915,Emily,Cat,Domestic Medium Hair,,98103
February 13 2018,964371,George,Cat,Domestic Shorthair,,98118
February 13 2018,964372,Jet,Cat,Domestic Shorthair,,98118
February 17 2018,S140000,Wilbur,Cat,Domestic Shorthair,,98103
March 20 2018,272327,Slightly,Cat,Manx,,98102
April 12 2018,S141564,Kevin Boots Ewing,Cat,Domestic Shorthair,Mix,98125
April 17 2018,282510,Violet,Cat,Domestic Medium Hair,,98144
April 20 2018,214982,Faith,Cat,Domestic Shorthair,,98116
April 20 2018,214983,Tanner,Cat,Domestic Medium Hair,,98116
April 26 2018,S109975,Simon,Cat,American Shorthair,Siamese,98126
May 02 2018,83448,North Star,Cat,Domestic Shorthair,Mix,98133
May 16 2018,S126432,Chico,Cat,Domestic Shorthair,Mix,98122
June 22 2018,964224,Hudson,Cat,Domestic Shorthair,,98118
July 10 2018,85322,Zen,Cat,Domestic Shorthair,Siamese,98105
July 26 2018,S131486,Estrela,Cat,LaPerm,,98103
August 13 2018,578368,Lucky,Cat,Oriental Shorthair,Mix,98106
August 30 2018,129654,Daisy,Cat,Domestic Longhair,,98144
August 30 2018,S121902,Buffy,Cat,Siamese,European Shorthair,98107
August 31 2018,S120217,Leeloo,Cat,Domestic Shorthair,,98102
August 31 2018,S120218,Zorg,Cat,Domestic Shorthair,,98102
September 08 2018,8000753,Digger,Cat,Domestic Longhair,,98133
September 12 2018,S103858,Piper,Cat,Domestic Shorthair,,98144
September 16 2018,8000965,Gamzee,Cat,Maine Coon,Mix,98103
September 16 2018,8000966,Nepeta,Cat,American Shorthair,Mix,98103
October 08 2018,273248,Cloudy,Cat,Domestic Shorthair,,98122
October 21 2018,715838,Little Bear,Cat,Domestic Shorthair,,98122
October 31 2018,S126958,Bailey,Cat,Domestic Shorthair,,98119
November 18 2018,118752,Princess,Cat,Domestic Shorthair,,98103
```

`SeattlePets.csv` is part of Lab 4

```
GNU nano 5.8                                                   Pet.java
// A simple class holding information about a pet.

import structure5.*;
import java.util.Scanner;
import java.io.*;

/**
 * A class for keeping track of information about a pet:
 * name, species, breed, license, license date, and zipcode.
 */
public class Pet
{
    /**
     * Construct a pet from a Vector generated from the CSV reader on the
     * Seattle pet registration database.
     *    Field   Meaning
     *      0     Licensing date
     *      1     License ID (alphanumeric)
     *      2     Name
     *      3     Species (e.g. "Cat")
     *      4     Breed (primary; e.g. "retriever"
     *      5     Breed (secondary; ignored; e.g. "mix")
     *      6     Zipcode (a possibly empty integer string representation)
     * Secondary breed information is ignored, and integer zipcode defaults
     * to 0.
     * <p>
     * @param description A Vector of strings in Seattle Pet format
     */
    public Pet(Vector<String> description) {
    }

    /**
     * Returns the license date associated with the pet.  E.g. "June 08 2018"
     * @return A string representing the licensing date.
     */
    public String licenseDate()
    {
        return "";
    }
```

`Pet.java` is part of Lab 4

```
GNU nano 5.8                                                    InsertionSort.java
// A stable sort - insertion sort - from the Java Structures text.

public static void insertionSort(int data[], int n)
// pre: 0 <= n <= data.length
// post: values in data[0..n-1] are in ascending order
{
    int numSorted = 1;  // number of values in place
    int index;          // general index
    while (numSorted < n) {
        // take the first unsorted value
        int temp = data[numSorted];
        // ...and insert it among the sorted:
        for (index = numSorted; index > 0; index--) {
            if (temp < data[index-1]) {
                data[index] = data[index-1];
            } else {
                break;
            }
        }
        // reinsert value
        data[index] = temp;
        numSorted++;
    }
}
```

`InsertionSort.java` is part of Lab 4

Generalized Sorting

Discussion: Generalized Sorting

When discussing sorting algorithms we have been focused on `int` arrays.

- How would we sort other common objects like `String`?
- How would we sort new objects like `Pet`?

How is this handled in Java? (Or how should it be handled in Java?)

Think about this for 2 minutes.
Then discuss it with your neighbor for 3 minutes.

Is there only one way of sorting a given type of object?

- Provide five different ways of sorting `String` objects. Be creative!
- Provide five different ways of sorting `Pet` objects.

How is this handled in Java? (Or how should it be handled in Java?)

Generics, Comparables, Comparators, Lambdas

Activity: Generalized Sorting

Determine answers to the following questions to the best of your abilities:

- What is a *generic*?
- What is a *comparable*? What is a *comparator*?
- What is a *lambda expression*? (Also know as a *lambda function*.)

Work with your neighbor for 5 minutes.

Relate what you have found to the previously discussed problems.

- How could you sort `String` objects in five different ways?
- How could you sort `Pet` objects in five different ways?

You will learn more about the precise mechanisms for doing these things in Java in Lab 4.

Midterm Review

Problem 3 [15 points]
Big-O and Counting.

a. A *subsequence* of charactacters found in a string, `str`, is any string that can be constructed by possibly deleting characters from `str`. Two subsequences are *distinct* if they are not equal. For example, there are 7 distinct subsequences of `"eve"`:

`"", "e", "v", "ev", "ee", "ve", "eve"`

If `str` has length `n`, then what is the maximum number of distinct subsequences that it can contain? What is the minimum number? Provide exact answers to these two questions (e.g., *the maximum is* n^2+1), as opposed to a big-O.

b. Our formal definition of big-O is below. Fill in the blanks. *Also* provide an intuitive definition of big-O in your own words, making sure to address the role of c and n_0.

Formal definition:

Definition 5.1 *A function* $f(n)$ *is* $O(g(n))$ *(read "order g" or "big-O of g"), if and only if there exist two positive constants,* c *and* n_0*, such that*

____

for all ____________ .

Intuitive definition:

c. Rank the following from smallest to largest. Provide your answer in the table below. For example, O(1) is the smallest, so it is given rank 1, as indicated in the table. You do not need to justify your answer to this question.

$O(n)$	$O(n^2)$	$O(2^n)$	$O(n^3)$	$O(1)$	$O(n!)$	$O(\log_2(n))$	$O(n^n)$
				1			

d. Suppose that `A` is an `int` array of length `n` that is sorted from smallest to largest. How quickly could you determine the number of distinct values in the array? For example, if `A` is `{1,1,3,4,4}` then it has three distinct values. Provide your answer in big-O along with a brief justification.

e. The following function `binary(n)` takes a non-negative integer, `n`, and returns a string that gives the base-2 (binary) representation of `n`. What is its run-time in big-O?

```
/**
 * Build a String with the binary representation of a number n.
 *
 * @param number the number to represent in binary
 * @result a String whose characters are the binary digits of n
 * @pre
 */
public static String binary(int n) {
    if (n < 2) return ""+n;
    else
        return binary(n/2)+(n%2);
}
```

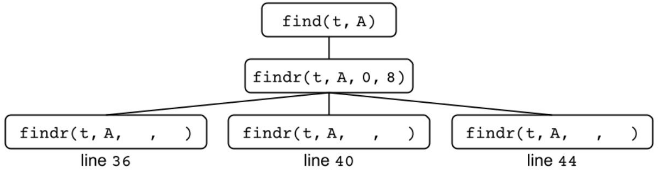
Problem 4 [20 points]
Debugging. A *ternary search* is similar to a binary search, except that the search space is repeatedly divided into thirds instead of halves. You are trying to implement ternary search in a function `find` which calls a recursive function `findr` (see next page). The input to `find` is an `int` array `A` that is sorted from smallest to largest, and a target value `t`. If the target is in `A`, then `find` should return an `index` with `A[index] = t`; otherwise, it should return `null`.

Good news: `find` is partially working! Bad news: It has some bugs. To debug your code, you created an array `A = {1,1,3,3,5,7,9,11,11}` and tested `find` with all targets `t` between `0` and `12`, with results given in the table below, where ∞ denotes infinite recursion.

target t	0	1	2	3	4	5	6	7	8	9	10	11	12
expected	null	0 or 1	null	2 or 3	null	4	null	5	null	6	null	7 or 8	null
received	null	1	null	3	null	4	null	null	null	∞	∞	null	null
	✔	✔	✔	✔	✔	✔	✔	✘	✔	✘	✘	✘	✔

Running `find(t, A)` on array `A = {1,1,3,3,5,7,9,11,11}` with targets `t = 0,1,…,12`.
For example, when the target is `t = 5`, the `find` function is correct in returning `4` since `A[4] = 5`.
Similarly, when the target is `t = 8`, the `find` function is correct in returning `null` since `A` has no `8`.

a. Your tests have identified some false negatives, including `find(7,A)` returning `null`. You believe that these errors are being caused by the recursive calls in lines `34-46`. To test this hypothesis, consider the first level of recursion resulting from `find(t,A)`, where `A` is the test array given above. Depending on the target `t`, a recursive call will be made on line `36`, `40`, or `44`. Provide the missing arguments for these three calls in the figure below. For example, line `36` calls `findr(t, A, left, middle1 - 1)`, so you should provide the values of `left` and `middle1 - 1` in the space provided. You may find it helpful to record intermediate values directly in the code. For example, you may wish to write `size = 9` on line `17`, since `right-left+1 = 8-0+1 = 9`.



Function calls resulting from testing `find` on array `A = {1,1,3,3,5,7,9,11,11}`.
Depending on the target `t`, one of the three recursive calls on the bottom row will be called.
Fill in the missing `left` and `right` arguments for each of these calls with specific values.

b. Explain how the three recursive calls from part a. lead to false negatives in your tests. Are there any other issues in lines `34-46` that would lead to false negatives?

c. Your tests have also identified infinite recursion, including `find(9,A)` running forever. In general, what are some potential causes of infinite recursion in implementations of binary search and ternary search?

d. You believe that your infinite recursion is caused by the base cases in lines `7-13` and/or the calculations in lines `23-25`. Identify the issue. Note that your assertion on line `26` is never triggered. Hint: Think about `size1` and the recursive call in line `36`.

```
public static Integer find(int t, int A[]) {
      return findr(t, A, 0, A.length - 1);
}

protected static Integer findr(int t, int A[], int left, int right) {
      // Base Case: There are no elements to search.
      if (left > right) return null;

      // Base Case: There is one element to search.
      if (left == right) {
            if (t == A[left]) return left;
            else return null;
      }

      // Compute the size of the range A[left], ... ,A[right], and
      // divide it into thirds, and account for any extra remaining.
      int size = right - left + 1;
      int third = size / 3;  // Java rounds towards zero.
      int extra = size % 3;  // The remainder modulo 3.

      // Split size into three parts, with the extra in the first part.
      // Assert that the three smaller sizes sum to the overall size.
      int size1 = third + extra;
      int size2 = third;
      int size3 = third;
      Assert.condition(size1+size2+size3 == size, "Wrong size sum");

      // We will split A[left], ..., A[right] into three sub-ranges.
      // Compute the two middle points.
      int middle1 = left + size1;
      int middle2 = left + size1 + size2;

      // Search for the target value in one of the three sub-ranges.
      if (t < A[middle1]) {

            return findr(t, A, left, middle1 - 1);

      } else if (t < A[middle2]) {

            return findr(t, A, middle1, middle2 - 1);

      } else {

            return findr(t, A, middle2, right - 1);

      }
}
```