Computer Science 1 — CSci 1100
Lab 11 — Closest Point Algorithm
Fall Semester 2014

Lab Overview

Algorithms that compute geometric properties of data are common in applications ranging from games, to computer vision, to robotics, to bioinformatics. This labs explores a simple geometric problem — finding the two closest points in a list. These will just be values rather than x, y or x, y, z coordinates. As an example, in the list

```
L1 = [ 15.1, -12.1, 5.4, 11.8, 17.4, 4.3, 6.9 ]
```

the values 5.4 and 4.3 are the closest together.

In the first two checkpoints, you will write two different solutions, one that finds the two closest values without sorting, and another that finds the values by first sorting the list. You will need to test these carefully to convince yourself (and us) that they are correct. In the last checkpoint, you will analyze the solutions both "by hand" and using timing experiments to decide which is better.

We are not directly providing you with any of the code for this lab. However, code from earlier lectures may be helpful here, and you can use as much of it as you wish.

Checkpoint 1: Solution Based on Double For Loops

Write a function called `closest1` that takes as its only argument a list of floats (or ints) and returns a tuple containing the two closest values. You may assume there are at least two values in the list. This function should not change the list at all, but instead it should use two `for` loops over the range of indices in the list to find the closest values. If you can, try to do this without testing the same pair of values more than once.

To illustrate its usage, assuming we've assigned `L1` from above, the code

```
(x,y) = closest1(L1)
print x, y
```

should output

```
4.3 5.4
```

Think hard about the cases you should use to test that your code is correct. Then generate examples of these cases using Nose and run your function on them. Fix any mistakes.

To complete Checkpoint 1 Show your TA or mentor (a) your code and (b) your Nose test program. Be prepared to explain why your tests are reasonably complete. You might be asked to generate new test cases.

Checkpoint 2: Solution Based on Sorting

Write a function called `closest2` that takes as its only argument a list of floats (or ints) and returns a tuple containing the two closest values. You may assume there are at least two values in the list. This function should not change the list. It should, however, make a copy of the list, sort the copy, and then, in a single pass through the sorted list, decide which are the two closest values. You might have to think a bit about why this idea should work.

Once again, think through the test cases you will need. Then, generate test cases using Nose and use them to ensure your function is correct.

To complete Checkpoint 2 Show your TA or mentor (a) your code and (b) your Nose test program. Be prepared to explain why your tests are somewhat complete. You might be asked to generate new test cases.

Checkpoint 3: Comparative Evaluation

In this checkpoint you will evaluate the result in two ways. First, following through what we did in Lectures 19, if the length of the list is N, roughly how many comparisons does the first version make as a function of N? Assuming (correctly), that sorting makes $O(N \log N)$ comparisons, how many additional comparisons does the code you wrote for the second version make? Based on this, which function do you think is faster?

The second evaluation is to run timing experiments on lists of random values. Unlike the code in `lec19_search_smallesttwo.py` from the searching lectures which uses `random.shuffle` to randomly shuffle a sequence of integers, you will need to use the function `random.uniform` to generate your experimental sequence. For example,

```
random.uniform(0.0, 1000.0)
```

generates a single random float between 0 and 1000. Other than this, feel free to adapt and modify as much of `lec19_search_smallesttwo.py` or `lec19_search.py` as you wish.

For simplicity, you can include all code from Checkpoints 1-3 in a single py file. For this checkpoint, you will not use Nose tests anymore. We are now assuming your code is correct and we are comparing its performance.

All code developed in class is available under class modules:

`http://www.cs.rpi.edu/~sibel/csci1100/fall2014/course_notes/class_modules/doc/class_modules.html`

Run timing experiments to the compare the performation of your two solutions on random lists of length 100, 1,000, 10,000. What do you conclude about the two solutions?

To complete Checkpoint 3 Explain your analysis from the first part of this checkpoint, and then show your code and your experimental results from the second part. Explain your final conclusion.