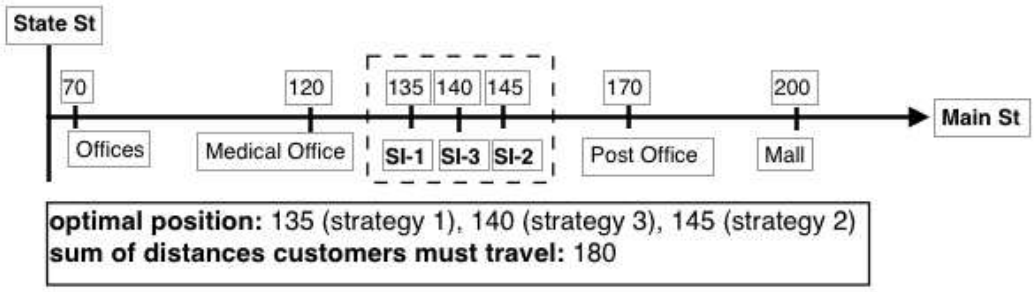
Problem-Based Intro. to Computer Science
Store Locator Lab (2011-1)

10/11/2011

1 Problem

You have been hired to help determine the placement for a new Donut Delite store. There are several store fronts available along Main Street. Developers are trying to determine the best placement of a store based on the distance to several large office buildings. They want to locate the store so that it minimizes the distance anyone in the office buildings would have to travel to reach the store, and they would like to use your application to determine the location of any new store. (Ignore the fact that each building could have a different number of people.) The diagram below shows locations chosen by several strategies. Numbers indicate distances to the intersection of Main and State.



1.1 Problem-Solving Session Part One: Analyze Strategies

You were asked to consider the following strategies to determine the best location of the new store (sometimes there might be more than one best location). The problem is that only one of these strategies *always* provides the *best* possible sum of distances.

1. **Midpoint Distance Strategy**
   Let x be the distance of the business closest to State Street. Let y be the distance of the business farthest from State Street. Locate the new store at the mid-point between x and y; that is, at location $(x+y)/2$. For the above example, this strategy produces the location 135 and the sum of the distances to this location is $65 + 65 + 15 + 35 = 180$.
2. **Median Distance Strategy**
   Sort the distances and choose the location of the middle element (if the number of elements is even, choose the midpoint between the two middle elements). For the above example, this strategy produces 145, and the sum of the distances to this location is $75 + 55 + 25 + 25 = 180$.
3. **Average Distance Strategy**
   Locate the store at the average of the distances. For this example, this strategy produces 140, and the sum of the distances to this location is $70+60+20+30 = 180$.

Problem-Solving Tasks - (20%)

In a team of four to five students, do the following:

1. Create test cases that demonstrate how two of the strategies do not work correctly. For each test case, indicate the calculated store location and the sum of the distances to each business, versus the optimal choice. You may need more than one test case to show that two of the three strategies are sub-optimal. The remaining strategy is, of course, the correct, optimal one.

   After 20 minutes' work on this task, hand in your team's work with all team members' names clearly legible, and we will discuss the teams' findings before continuing.

2. Develop pseudocode that implements the correct strategy.

At the end of problem-solving or when you have finished your pseudocode task, hand in your team's pseudocode with all team members' names clearly legible.

[You will receive a handout on the lab tasks after problem-solving before you go to the labs.]

1.2 Problem-Solving Session Part Two: QuickSelect

There is an algorithm, known as quickselect, that runs faster than a *simplistic implementation* of the optimal, sorted median strategy.

The quickselect approach is more efficient than sorting techniques we have seen so far. Quickselect finds the **kth smallest number** in an *unsorted* list of numbers. Study the algorithm below and consider how you could use quickselect to compute the median and solve the store location problem.

```
Function quickSelect( aList, k ):
    If aList is not empty:
        pivot <- Choose the element at position ( len( aList ) // 2 )
        smallerList <- All elements of aList smaller than pivot
        largerList <- All elements of aList larger than pivot
        count <- the number of occurrences of pivot value in aList
        m <- the size of smallerList
        If k >= m and k < m + count then:
            return pivot
        If m > k:
            return quickSelect( smallerList, k )
        Otherwise:
            return quickSelect( largerList, k - m - count )
```

Questions

Answer the following questions on quickselect and program performance as you design, implement and test your programs. This will show that you understand how quickselect works and how to measure execution time in a program. (hint: to analyze the algorithm, apply it by hand to a small, unsorted data sequence.)

1. What is the purpose of the `pivot`? Do we really need to choose the middle item as the pivot? Would the algorithm have behaved differently if we had selected the first item as the pivot each time?
2. Quickselect is versatile because it can find the **kth smallest number**. What test cases are needed to fully exercise quickselect? List several cases that exercise all aspects of the quickselect function. Remember: *a test case identifies specific input values and expected outputs*.
3. What is the elapsed time performance of your quickselect program when it processes a large data set? (See link below). What is the elapsed time performance of your median program when it processes the same large data set? Enter the elapsed times of *several runs* of each program in your report.

Lab Implementation - (80%)

For this lab's work, you must implement *two programs* that implement the same strategy and *instrument* each program to measure and report its time performance.

Your *individual implementation* must complete and deliver the following:

1. `storeLocation.py`: a program implementing the *simplistic* median approach;
2. `selectMedian.py`: a program implementing the *quickselect* algorithm; and
3. `report.txt`: a file containing answers to the questions posed earlier.

Program Operation

Each program has similar structure, processes the same input files, and produces the same kind of results. Both implementations must prompt for the input file name. The input file content will be formatted as shown in this short example:

```
Offices 70
MedicalOffice  120
PostOffice 170
Mall 200
```

Each program must calculate and print the optimal location of the store and the sum of the distances that people would have to travel to reach the store from each given location.

Each implementation must have instrumentation that reports the elapsed time used to sort and select the location. The Python `time` module has a `time.clock()` function that returns the current number of fractions of a second (system dependent). Use the time module to capture a start time and a finish time; the difference is the elapsed time.

To test performance, Download and analyze a large test case linked here:

`http://www.cs.rit.edu/~vcss241/Pub/Labs/05/testDataSet10K.txt`

Run each implementation several times with this data file. Measure how much time it takes to find the median *after* reading the file and *before* printing results. List your time comparisons for each implementation. Be patient; these runs may take some time.

You may not use the built-in `list.sort()` function. You may want to copy the **lecture's sort program**, modify the code to make it work with files, and run the program with this lab's input files.

1.3 Submission

Submit all the required, components in a zip file. Your grade will be based on the following *individually-developed work*:

1. 25%: The simplistic median program, `storeLocation.py`;
2. 25%: The quickselect median program, `selectMedian.py`; and
3. 30%: The `report.txt` file containing the answers to the questions and a report of each program's performance.

Zip the files with their appropriate documentation into a file called `lab.zip` and submit the zip file to the MyCourses dropbox for this assignment.