SI 335 Spring 2012: Project 5

This is a written project. Solutions to all parts must be typed or very neatly written, and handed in before the beginning of class on **Wednesday, April 4**. Late submissions will not be accepted, in accordance with the class policy. Students with excused absences on that date may email a copy to the instructor by the deadline *in addition to handing in a paper copy at the earliest opportunity.*

Guidelines:

- All work must be typed or neatly written
- Multiple-page submissions must be stapled or otherwise bound securely into a single packet.
- **Be concise**. Most problems have at least one simple and relatively short correct answer. Solutions will be graded not only on their correctness but also on how clearly that solution is presented.
- In general, there is a preference for **correct** algorithms over efficient ones. If the problem asks for a $O(n)$ algorithm, a *correct* $O(n \log n)$ algorithm is usually preferable over an incorrect one that claims $O(n)$ running time.
- **Do not try to pass off a solution you know is incorrect**. A statement to the effect of "I know this doesn't work because of blah" will be effective in garnering partial credit points.
- Follow the course honor policy posted here. Remember that **no written collaboration with other humans is permitted**, and that you must document all your sources.

1 Another Sorting Algorithm

Important: For this problem only, you are NOT allowed search for resources online. You might stumble across someone else's solution, and what would be the fun in that? To be clear: I am instructing you on your **honor** not to consult any websites or do any web searches related to this particular problem.

Now here is a sorting algorithm (from an exercise in our CLRS textbook, and named after the three stooges):

```
StoogeSort(A)
Input: Array A of size n
Output: A is sorted in increasing order

if n <= 3 then
  InsertionSort(A)
else
  k := floor(n/3)
  StoogeSort(A[0..2k])
  StoogeSort(A[k..n-1])
  StoogeSort(A[0..2k])
end if
```

a) Show why this algorithm is correct.

   Hint: There are multiple ways to do this, but what I would do is split into cases based on the position of any given element. If you know every element ends up in the correct position, then the whole array must be sorted!

b) Give a big-Theta bound on the worst-case running time of this algorithm. Show your work.

c) Improve the algorithm by replacing one of the three recursive calls with a linear-time subroutine. (Two of the recursive calls must stay the same!)

d) Give a big-Theta bound on the worst-case running time of the improved algorithm from part (c), and again show your work. You should be able to do this even if you didn't get part (c).

e) Very briefly say how the original algorithm and the improvement in part (c) each compare to the other sorting algorithms we have covered in class.

2 Hacking

(Note: this is **not** an ethics question. It's just an algorithms question with what I hope is an intriguing set-up. If you must, assume that you are working for the NSA, or that you are testing internal security — that accomplishing this task is ethical and moral.)

You want to break into the computer system of organization X. The first step in accomplishing this is to compromise a single user's account by guessing their password. Someone else on your hacking team has already written a program called `guess_password(user)` that tries to guess the password for the named user on X's system. You're not sure exactly how `guess_password` works, but it probably relies on trying the most commonly-used passwords from a list.

Now this `guess_password` program is pretty good: you have reason to believe that it will work to hack into about 20% (that is, $\frac{1}{5}$) of the accounts on X's system. Of course you only need to hack into *a single* account, and you don't care which one.

Your task is to write a program that takes a massive list L of usernames, and tries to find one whose password you can guess. Here is a simple idea for this:

`basicHack(L)`
Input: List L containing n usernames
Output: The username of an account whose password has been guessed.

```
for i from 0 to n-1 do
  if guess_password(L[i]) = "success" then
    return L[i]
  end if
end for
return "failure"
```

a) What is the best-case running time of this algorithm, in terns of n, the size of the list of usernames L?

b) What is the worst-case running time of this algorithm, in terns of n, the size of the list of usernames L?

c) Describe (in a short description and then give pseudocode) an algorithm that uses **randomization** to quickly find a username whose password is easily guessed.

(Hint: I have in mind an extremely simple recursive algorithm that never returns "failure" but might run forever.)

d) What is the worst-case expected running time of your randomized algorithm? For full points, this should be the same asymptotically as the best-case of the original algorithm from part (a).

(Hint: To do the randomized analysis, you split into two cases, say "lucky" and "unlucky". Determine the cost (recurrence) in each case. The expected cost is the sum of the cost in each case, times the probability of that case. Look back at the notes for the average-case analysis of QuickSelect for a refresher.)

3 Data Mining

A certain grocery store has a "customer loyalty card" and uses these to record every time an individual with the card makes a purchase at the store. The day of the week (0 for Sunday, 1 for Monday, ..., 6 for Saturday) and the hour of the day (0 up to 23) are recorded at every transaction for each customer.

For each customer, for a given week, there is a list of times $[d_1, h_1, d_2, h_2, d_3, h_3, \ldots]$, where d_i is the day of the week of the ith visit, and h_i is the hour of the ith visit. For example, if Bob visits the store on Monday at 10am, Tuesday at noon, Friday at 9am and Friday again at 11pm, his times list would be

$$[1, 10, 2, 12, 5, 9, 5, 23]$$

Now the store wants to use this data to group customers according to shopping habits, so that it can target advertising to them. Specifically, the store wants an algorithm that takes a list of customers and the time each customer visited in a given week, and returns a list of groups of customers, such that every customer in each group made visits to the store at the same times in that week.

Here is a more formal specification of the problem:

> `groupCustomers(L)`
> **Input**: List L containing n lists of customer times, where each list in L stores at most m visit times (and therefore has length at most $2m$).
> **Output**: A list of lists of integers, where each list represents a group of customers that visited at all the same times, and each customer is represented by an index in the original array `L`.

Of course each customer will only appear in exactly one group.

a) Describe an algorithm for this problem. Try to make your algorithm as efficient as possible.

b) Analyze the worst-case running time of your algorithm, in terms of n (the number of customers) and m (the maximum number of visits per customer).

c) This probably isn't a very useful data-mining algorithm because it is too restrictive. Even if two customers' visit times aren't *exactly* the same, they still might be very similar and so should end up in the same group.

   Describe another algorithm to group customers according to the similarity of their visit times. (You will first have to define a similarity measure.)

   This part is intentionally a bit vague and open-ended, like most real-life problems. I mostly want to see that you come up with a sensible idea and describe your algorithm clearly. You do not have to analyze your algorithm for this part.

4 Selection (up to 5% BONUS)

Consider the following variant on the selection problem: Given a list A of n elements, and an integer k, find the kth largest DISTINCT element in A, counting from 0. This is like the original selection problem, but ignoring duplicated elements.

Show that, at least in the comparison model, there can't possibly be a $O(n)$-time algorithm for this modified selection problem that ignores duplicate elements.