Question 1 of 3

Recall that pseudocode is an informal, English-like language that describes an algorithm. In Lecture 0, David showed the following pseudocode for finding Mike Smith in a phone book.

```
1  Pick up phone book
2  Open to middle of phone book
3  Look at page
4  If Smith is on page
5      Call Mike
6  Else if Smith is earlier in book
7      Open to middle of left half of book
8      Go back to line 3
9  Else if Smith is later in book
10     Open to middle of right half of book
11     Go back to line 3
12 Else
13     Quit
```

a) Write pseudocode for an algorithm that would identify the tallest person in a room.

b) What is the running time of your algorithm in Big O notation?

   - *Hint: if your algorithm has one or more loops, how many times do the loops execute if there are N people in the room?*

Answers

a) Answers may vary. One possible algorithm is below.

```
Point at first person in the room
For each remaining person p in the room
   If p is taller than the person you are pointing at
       Point at p
Return person you are pointing at
```

b) This algorithm runs in O(n) time, since we need to make one pass through all n people in the room.

Question 2 of 3

Recall from lecture how we implemented a phone book in C using the below `struct`.

```
typedef struct
{
    string name;
    string number;
}
person;
```

a) Why was it arguably better design to use one array of `struct`s than to use two arrays, one to store names and one to store phone numbers?
b) Imagine you were to use the above `struct` to implement an app for your contacts, like the one on your mobile phone. What are two additional fields might you want to add to the `struct`, and what should their types be?

Answers

a) Using `struct`s allows us to encapsulate related pieces of data together. This avoids the assumption that the array of names and the array of phone numbers will always line up with one another, and avoids the potential danger of rearranging names but forgetting to rearrange the corresponding numbers the same way.
b) Answers may vary. Possibilities include:
   i) Email address, a string.
   ii) Favorite, a boolean.
   iii) Address, a string.

Question 3 of 3

Imagine that you have an unsorted collection of items (maybe they're notes for class, or a collection of old receipts) that you expect you'll need to search. When might it make more sense to sort the collection of items first before searching, and when might it make more sense to leave the collection unsorted?

Hint: Consider algorithmic efficiency. What's the cost (i.e., running time) of linear search? Of binary search? Of sorting?

Answer

If you're only ever going to search the collection once (or very few times), then it makes more sense to search the data unsorted, using linear search. Linear search takes O(n) time, whereas sorting the data first and then searching would take O(n log n) time.

If you're going to search the collection many times, then it might make sense to sort the data first, taking O(n log n) time, but speeding up all subsequent searches, so that all future searches could be done in O(log n) time each, rather than O(n) each.