CpSc 840: Design and Analysis of Algorithms

Instructor: Dr. Brian Dean — Spring 2013
Webpage: `http://www.cs.clemson.edu/~bcdean/` — TTh 11:00-12:15
Handout 7: Quiz #1. 1,000,000 possible points. — Lehostsky 246

Some of the following questions ask you to design an algorithm. Please do so *clearly* and also *concisely*, in English (pseudocode is generally not necessary). For each algorithm, please also briefly justify its running time. Standard results from class or the textbook can be used as "black boxes" without extra elaboration. Partial credit will be awarded for valuable insight or slightly slow but correct solutions, as long as they are not overly complicated. Unless otherwise stated, use the RAM model of computation. Be sure to keep an eye on the clock and pace yourself well. Good luck!

1. High Frequency (200,000 points). Given an array $A[1 \ldots n]$, a *high-frequency* element is a value x that occurs a maximum number of times in A. For example, if $A = [4, 6, 1, 2, 6, 1, 1, 3, 6]$, then both 1 and 6 are high-frequency elements, since no other elements have more occurrences. Give a fast comparison-based algorithm for computing a high-frequency element in A.

2. High Frequency, Revisited (200,000 points). Argue that any comparison-based algorithm for solving the previous problem must take $\Omega(n \log n)$ time in the worst case.

3. Piles (200,000 points). Bored in CpSc840 lecture one day, Shiree draws a picture of a set of rectangles in her notebook:

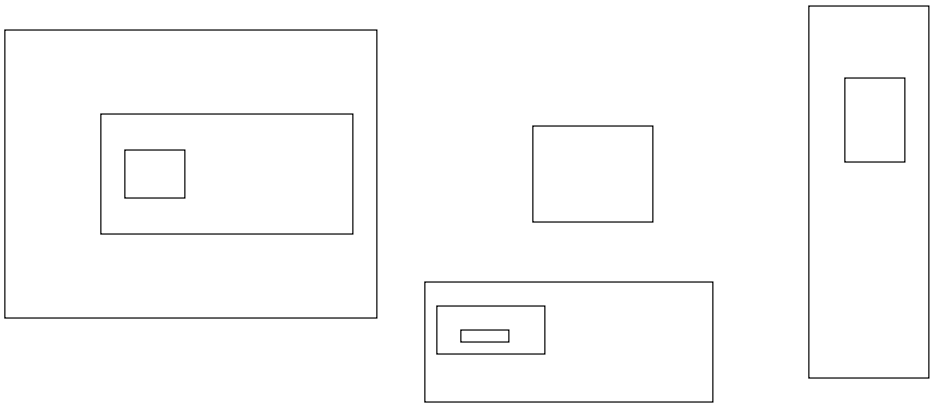

Figure 1: A collection of rectangles.

Curiously, Shiree notes that if any two rectangles touch, then one is strictly contained within the other. Moreover, if two rectangles r and r' are contained within the same larger rectangle, then either r must be contained within r' or vice-versa. As a result, the rectangles form what Shiree calls "piles", where a pile is defined as a maximal chain of rectangles where each contains the next. The figure above consists of 4 piles of rectangles. Shiree would like to know the number of piles in her drawing. Help her by devising a fast algorithm that counts the number of piles in a set of n rectangles satisfying the conditions above. Each rectangle is specified by the coordinates of its corner points.

4. No Duplicates (200,000 points). Sakti is writing code to implement an AVL tree. Unfortunately, she has stayed up too late studying for a CpSc 840 quiz and in her tired state, accidentally modifies the *insert* routine as follows:

```
Insert(x):
  Call find(x) to check if x is already present in the structure
  If so, call delete(x) and then insert(x+1)
  Otherwise, insert the value x as in a normal AVL tree.
```

Sakti is concerned that a single call to *insert* might take a very long time, since it might cause a propagating chain of calls to *insert* in the worst case. For example, if 3, 4, and 5 are already present in the tree and *insert*(3) is called, then 3, 4, and 5 will be deleted and the value 6 will ultimately be inserted. However, please argue that the amortized running time of *insert* is only $O(\log n)$, where n is an upper bound on the number of elements that are ever present in the structure during its existence. For full credit, use potential functions; partial credit can still be obtained by using other methods of amortized analysis.

5. Stratified Heaps (200,000 points). A common misconception about heaps is that if x is an element on the level below element y, then $x \geq y$. This is certainly true if x is a child of y due to the heap property, but it is not true in general, as shown in the valid heap on the left, where the element 8 appears on the level below the element 12.

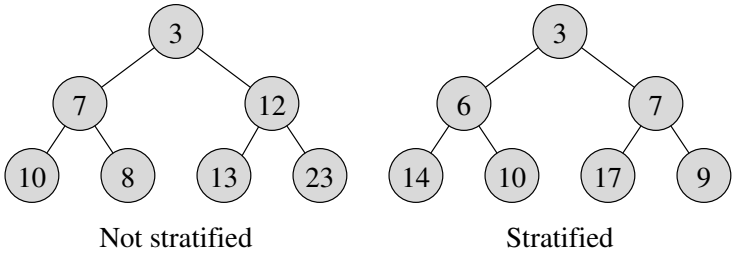

Figure 2: Examples of non-stratified vs. stratified heaps.

Let us call a heap *stratified* if it does satisfy the more stringent "heap property" above. In a stratified heap, all the elements on one level of the heap are no larger than all the elements in the next level of the heap. Since stratified heaps look harder to build than normal heaps due to this more demanding requirement, please either (i) show that you can still build a stratified heap in $O(n)$ time, *or* (ii) argue that in the comparison model, any algorithm for building a stratified heap must take $\Omega(n \log n)$ worst-case time.

This page can be used for scratch work.

This page can be used for scratch work.