CS 473U: Undergraduate Algorithms, Fall 2006
Homework 4

Due Tuesday, October 10, 2006 in 3229 Siebel Center

Remember to submit **separate, individually stapled** solutions to each of the problems.

---

1. Chicago has many tall buildings, but only some of them have a clear view of Lake Michigan. Suppose we are given an array $A[1..n]$ that stores the height of n buildings on a city block, indexed from west to east. Building i has a good view of Lake Michigan if every building to the east of i is shorter than i. We present an algorithm that computes which buildings have a good view of Lake Michigan. Use the taxation method of amortized analysis to bound the amortized time spent in each iteration of the for loop. What is the total runtime?

```
GOODVIEW(A[1..n]):
  Initialize a stack S
  for i = 1 to n
      while (S not empty and A[i] ≥ A[S.top])
          POP(S)
      PUSH(S, i)
  return S
```

2. Design and analyze a simple data structure that maintains a list of integers and supports the following operations.

   (a) CREATE(): creates and returns a new list L

   (b) PUSH(L, x): appends x to the end of L

   (c) POP(L): deletes the last entry of L and returns it

   (d) LOOKUP(L, k): returns the kth entry of L

   Your solution may use these primitive data structures: arrays, balanced binary search trees, heaps, queues, single or doubly linked lists, and stacks. If your algorithm uses anything fancier, you must give an explicit implementation. Your data structure should support all operations in amortized constant time. In addition, your data structure should support LOOKUP() in worst-case $O(1)$ time. At all times, your data structure should use space which is linear in the number of objects it stores.

3. Consider a computer game in which players must navigate through a field of landmines, which are represented as points in the plane. The computer creates new landmines which the players must avoid. A player may ask the computer how many landmines are contained in any simple polygonal region; it is your job to design an algorithm which answers these questions efficiently.

   You have access to an efficient static data structure which supports the following operations.

- CREATES($\{p_1, p_2, \ldots, p_n\}$): creates a new data structure S containing the points $\{p_1, \ldots, p_n\}$. It has a worst-case running time of $T(n)$. Assume that $T(n)/n \geq T(n-1)/(n-1)$, so that the average processing time of elements does not decrease as n grows.
- DUMPS(S): destroys S and returns the set of points that S stored. It has a worst-case running time of $O(n)$, where n is the number of points in S.
- QUERYS(S, R): returns the number of points in S that are contained in the region R. It has a worst-case running time of $Q(n)$, where n is the number of points stored in S.

Unfortunately, the data structure does not support point insertion, which is required in your application. Using the given static data structure, design and analyze a dynamic data structure that supports the following operations.

(a) CREATED(): creates a new data structure D containing no points. It should have a worst-case constant running time.

(b) INSERTD(D, p): inserts p into D. It should run in amortized $O(\log n) \cdot T(n)/n$ time.

(c) QUERYD(D, R): returns the number of points in D that are contained in the region R. It should have a worst-case running time of $O(\log n) \cdot Q(n)$.