CS 466/666, Spring 2011
Timothy Chan

Assignment 2 (due June 8 Wednesday 5pm)

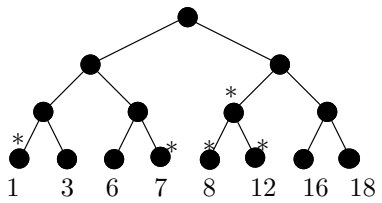
Please read `http://www.student.cs.uwaterloo.ca/~cs466/policies.html` first for general instructions.

1. [*7 marks*] In the Fibonacci heap method, suppose we change Fredman and Tarjan's invariant to allow at most 100 (instead of 1) children to be cut from a non-root node. Prove that any tree of size n that can be formed by the algorithm still has degree bounded by $O(\log n)$. [Note: no pseudocode is required. Hint: a recurrence satisfying $f_i \geq 2f_{i-c}$ has solution $f_i = \Omega(2^{i/c})\ldots$]

2. [*20 marks*] Consider the problem of maintaining a set S of elements (real numbers) to support deletions and searches, but without insertions. We know that standard balanced search trees can achieve $O(\log n)$ time. In this question, we consider an approach which is simpler (no tree rotations necessary!) and which also has faster amortized deletion time (assuming that during a deletion, we are given a pointer to the element to be deleted). Specifically, consider the following operations:

   - preprocess($x_1, \ldots, x_n$): initialize S to contain the n elements $x_1, \ldots, x_n$, which are assumed to be given in sorted order.
   - delete(x): delete x from S.
   - search(z): return either the predecessor or the successor of z, i.e., the largest element in S smaller than z or the smallest element in S greater than z. Note that z is not necessarily in S.

   Your data structure should be based on a complete (perfectly balanced) binary tree with n leaves storing the initial elements $x_1, \ldots, x_n$ from left to right. To delete an element, we do not change the shape of the tree but just mark the corresponding leaf node. Furthermore, we mark an internal node z iff all leaves underneath z have been deleted. The example below represents the set $\{3, 6, 16, 18\}$ after the deletions of $1, 7, 8, 12$ (marked nodes are shown with $*$):



   Give details of the data structure, and in particular, give pseudocode for both delete() and search(). For delete(), remember to describe how marks are updated. Show that search() takes $O(\log n)$ worst-case time, preprocess() takes $O(n)$ amortized time (assuming input has already been sorted), and delete() takes $O(1)$ amortized time. For the amortized analysis, use a potential method or a charging argument.

3. [*21 marks*] In this question, we consider a refinement of the "list" approach for solving the union-find problem; its advantage (over Tarjan's tree approach) is that it can guarantee $O(1)$ *worst-case* time for find.

   Instead of representing each set as a linked list of elements, we represent each set as a linked list of *blocks*, where each block is itself a linked list of elements. Fix a parameter k and classify blocks into 2 types: *mature* blocks, each of which have at least k elements, and *immature* blocks, each of which have fewer than k elements. We maintain that for each set A, there is at most 1 immature block in A's block list. Here is the rough pseudocode:

```
makeset(x):
 1. create a block β containing just one element x
 2. create a list B containing just one block β
 3. block-label[x] = β, set-label[β] = B

find(x):
 1. return set-label[block-label[x]]

union(A, B):
 1. if number of blocks in A ≥ number of blocks in B then {
 2.     for each block β of B do set-label[β] = A
 3.     A = concatenation of A's and B's block lists
 4.     if A now contains 2 immature blocks α and β,
            say, with number of elements in α ≥ number of elements in β, then {
 5.         for each element x of β do block-label[x] = α
 6.         α = concatenation of α and β
 7.         remove β from A's block list
        }
    }
 8. else do the same but with A and B swapped
```

   (a) [*4 marks*] For a fixed element x, prove that block-label[x] can change at most $\log k$ times.

   (b) [*4 marks*] For a fixed block β, prove that set-label[β] can change at most $\log n$ times.

   (c) [*4 marks*] Prove that at most $O(n/k)$ mature blocks can be formed during the lifetime of the data structure.

   (d) [*6 marks*] Using parts (a)–(c), prove that union takes $O(\log k + (1/k)\log n)$ amortized time.

   (e) [*3 marks*] What is the best choice of the parameter k (asymptotically) and the resulting amortized time bound for union and worst-case time bound for find?

4. [*12 marks*] Suppose we have a Las Vegas algorithm A with expected running time μ (measured in number of steps). As noted in class, we can convert A into a Monte Carlo algorithm with error probability at most 0.01 and worst-case running time at most 100μ, by Markov's inequality. We show that better strategies are possible.

(a) [*5 marks*] Consider the following strategy: run algorithm A for up to 10μ steps; if the algorithm terminates, stop and return the answer; otherwise, re-run algorithm A from scratch for up to 10μ steps (with an independent sequence of random bits). Show that this leads to a Monte Carlo algorithm with error probability at most 0.01 and requiring only at most 20μ steps.

(b) [*7 marks*] Improve the strategy further to get another Monte Carlo algorithm with error probability at most 0.01 and requiring at most 13μ steps. [Hint: repeat k times, each for $c\mu$ steps, for a suitable choice of constants k and c.]