CS 466/666
Assignment 2
Fall 2005
Due date: Friday October 21st, 6:00pm

Part 1 Amortized Analysis

1. Consider a UNION-FIND tree with no path compression. The unions are performed according a variant of weighted union in which *all* the nodes of the tree with the fewer nodes are made to point to the root of the tree with more nodes (ties are broken arbitrarily). In this case all nodes are at depth one and hence the cost of a find is O(1).
    i. Give an example of a UNION operation that takes Θ(n) time
    ii. Show that on a sequence of *m* operations containing *n* unions the amortized cost of a union is O(log n).
2. Assume that the entire sequence of *n-1* UNION's and *m* FIND's is given beforehand.
    i. give a definition for an expression tree which describes the sequence of unions.
    ii. consider a parameter *k* such that so long as the number of elements in a set is less than *k* the answer to a UNION or FIND operation can be computed in O(1) time. (*Hints: (a) create a special data structure to represent such small sets (b) since the sequence of union/finds are known in advance it is possible to use a small precomputed table to UNION two such sets (b) the precomputation must take place in O(n) time or less, to keep the amortized time per operation as O(1) (c) a post order traversal of the expression tree is helpful in establishing the known order of unions).*
    iii. describe a method to represent sets larger than *k* elements by using several of the small set structures described in part (ii) and collecting those smaller sets into a larger set using a standard union-find data structure such as weighted-union without path compression. (*Hint: every leaf node of a tree representing a single set in this part is a data structure of the type described in (ii) containing at least k/2 elements and at most k elements).*
    iv. observe that the number of elements involved in the larger sets of part (iii) is O(*n/k)*
    v. write your time analysis for this classical union-find data structure in terms of *n* and *k.*
    vi. show that there exists a value of *k* such that the amortized cost per operation for a find or union using the combined data structure, i.e. (ii)+(iii), is O(1).
3. Consider dynamic arrays as, say, in Java. In this case the size of the array is determined by the largest index accessed. Alison wishes to implement these as efficiently as possible.
    i. Give a method for implementing dynamic arrays such that *n* access operations have access cost of O(*m+n*) in the amortized sense and space usage is at most *2m,* where *m* is the largest index accessed.
    ii. Give a method that implements dynamic arrays such that *n* access operations have access cost of O(*m+n*) in the amortized sense but space usage is at most *m+O(sqrt(m)).*

Part 2 Adaptive Analysis

4. Given a list of unsorted numbers $a_1, a_2, \ldots, a_n$, an inversion is defined as a pair *(i,j)* such that $i < j$ but $a_i > a_j$.

i. What is the maximum number of inversions that a list may have?
ii. Show that the following sort runs in time $\Theta(n+I)$ comparisons where I is the number of inversions in the given input.

```
for i=1 to n do
    j = i;
    temp = list[i]
    while ((j > 1) && (list[j-1] > temp))
        list[j] = list[j-1];
        j = j - 1;
    list[j]=temp
```

5. Consider a balanced binary search tree such as an AVL tree or a B-tree, which stores *(key, value)* pairs, where the value *key* is unique. The data structure answers queries of the form *search(key)* with the associated *value* of the *key* if the *key* is present or "not there" otherwise. Give a modification of the search algorithm such that the time taken to do the *i-th* find is a function of the distance (in the sorted order of the keys) between the key retrieved in *i*-th find and the key retrieved in the *(i-1)th* find.