Chapter 11: Amortized Analysis

11.1 When the number of trees after the insertions is more than the number before.

11.2 Although each insertion takes roughly $\log N$, and each *DeleteMin* takes $2\log N$ actual time, our accounting system is charging these particular operations as 2 for the insertion and $3\log N - 2$ for the *DeleteMin.* The total time is still the same; this is an accounting gimmick. If the number of insertions and *DeleteMins* are roughly equivalent, then it really is just a gimmick and not very meaningful; the bound has more significance if, for instance, there are N insertions and $O(N/\log N)$ *DeleteMins* (in which case, the total time is linear).

11.3 Insert the sequence N, $N+1$, $N-1$, $N+2$, $N-2$, $N+3$, ..., 1, $2N$ into an initially empty skew heap. The right path has N nodes, so any operation could take $\Omega(N)$ time.

11.5 We implement *DecreaseKey(X, H)* as follows: If lowering the value of X creates a heap order violation, then cut X from its parent, which creates a new skew heap H_1 with the new value of X as a root, and also makes the old skew heap H smaller. This operation might also increase the potential of H, but only by at most $\log N$. We now merge H and H_1. The total amortized time of the *Merge* is $O(\log N)$, so the total time of the *DecreaseKey* operation is $O(\log N)$.

11.8 For the $zig-zig$ case, the actual cost is 2, and the potential change is $R_f(X) + R_f(P) + R_f(G) - R_i(X) - R_i(P) - R_i(G)$. This gives an amortized time bound of

$$AT_{zig-zig} = 2 + R_f(X) + R_f(P) + R_f(G) - R_i(X) - R_i(P) - R_i(G)$$

Since $R_f(X) = R_i(G)$, this reduces to

$$= 2 + R_f(P) + R_f(G) - R_i(X) - R_i(P)$$

Also, $R_f(X) > R_f(P)$ and $R_i(X) < R_i(P)$, so

$$AT_{zig-zig} < 2 + R_f(X) + R_f(G) - 2R_i(X)$$

Since $S_i(X) + S_f(G) < S_f(X)$, it follows that $R_i(X) + R_f(G) < 2R_f(X) - 2$. Thus

$$AT_{zig-zig} < 3R_f(X) - 3R_i(X)$$

11.9 (a) Choose $W(i) = 1/N$ for each item. Then for any access of node X, $R_f(X) = 0$, and $R_i(X) \ge -\log N$, so the amortized access for each item is at most $3\log N + 1$, and the net potential drop over the sequence is at most $N\log N$, giving a bound of $O(M\log N + M + N\log N)$, as claimed.

(b) Assign a weight of q_i/M to items i. Then $R_f(X) = 0$, $R_i(X) \ge \log(q_i/M)$, so the amortized cost of accessing item i is at most $3\log(M/q_i) + 1$, and the theorem follows immediately.

11.10 (a) To merge two splay trees T_1 and T_2, we access each node in the smaller tree and insert it into the larger tree. Each time a node is accessed, it joins a tree that is at least twice as large; thus a node can be inserted $\log N$ times. This tells us that in any sequence of $N-1$ merges, there are at most $N \log N$ inserts, giving a time bound of $O(N \log^2 N)$. This presumes that we keep track of the tree sizes. Philosophically, this is ugly since it defeats the purpose of self-adjustment.

(b) Port and Moffet [6] suggest the following algorithm: If T_2 is the smaller tree, insert its root into T_1. Then recursively merge the left subtrees of T_1 and T_2, and recursively merge their right subtrees. This algorithm is not analyzed; a variant in which the median of T_2 is splayed to the root first is with a claim of $O(N \log N)$ for the sequence of merges.

11.11 The potential function is c times the number of insertions since the last rehashing step, where c is a constant. For an insertion that doesn't require rehashing, the actual time is 1, and the potential increases by c, for a cost of $1 + c$.

If an insertion causes a table to be rehashed from size S to $2S$, then the actual cost is $1 + dS$, where dS represents the cost of initializing the new table and copying the old table back. A table that is rehashed when it reaches size S was last rehashed at size $S/2$, so $S/2$ insertions had taken place prior to the rehash, and the initial potential was $cS/2$. The new potential is 0, so the potential change is $-cS/2$, giving an amortized bound of $(d - c/2)S + 1$. We choose $c = 2d$, and obtain an $O(1)$ amortized bound in both cases.

11.12 We show that the amortized number of node splits is 1 per insertion. The potential function is the number of three-child nodes in T. If the actual number of nodes splits for an insertion is s, then the change in the potential function is at most $1 - s$, because each split converts a three-child node to two two-child nodes, but the parent of the last node split gains a third child (unless it is the root). Thus an insertion costs 1 node split, amortized. An N node tree has N units of potential that might be converted to actual time, so the total cost is $O(M + N)$. (If we start from an initially empty tree, then the bound is $O(M)$.)

11.13 (a) This problem is similar to Exercise 3.22. The first four operations are easy to implement by placing two stacks, S_L and S_R, next to each other (with bottoms touching). We can implement the fifth operation by using two more stacks, M_L and M_R (which hold minimums).

If both S_L and S_R never empty, then the operations can be implemented as follows:

Push(X,D): push X onto S_L; if X is smaller than or equal to the top of M_L, push X onto M_L as well.

Inject(X,D): same operation as *Push*, except use S_R and M_R.

Pop(D): pop S_L; if the popped item is equal to the top of M_L, then pop M_L as well.

Eject(D): same operation as *Pop*, except use S_R and M_R.

FindMin(D): return the minimum of the top of M_L and M_R.

These operations don't work if either S_L or S_R is empty. If a *Pop* or *Eject* is attempted on an empty stack, then we clear M_L and M_R. We then redistribute the elements so that half are in S_L and the rest in S_R, and adjust M_L and M_R to reflect what the state would be. We can then perform the *Pop* or *Eject* in the normal fashion. Fig. 11.1 shows a transformation.

Define the potential function to be the absolute value of the number of elements in S_L minus the number of elements in S_R. Any operation that doesn't empty S_L or S_R can

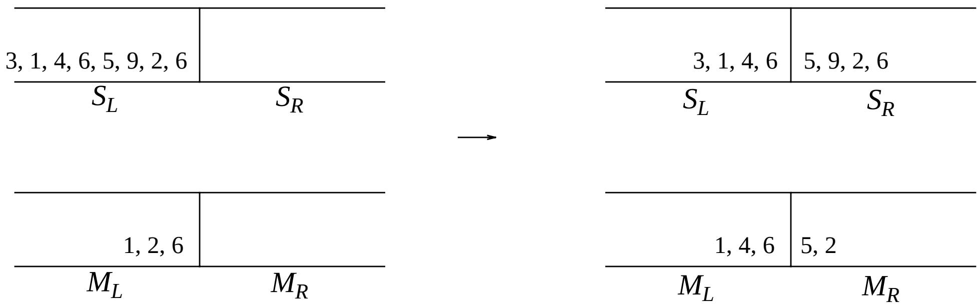

Fig. 11.1.

increase the potential by only 1; since the actual time for these operations is constant, so is the amortized time.

To complete the proof, we show that the cost of a reorganization is $O(1)$ amortized time. Without loss of generality, if S_R is empty, then the actual cost of the reorganization is $|S_L|$ units. The potential before the reorganization is $|S_L|$; afterward, it is at most 1. Thus the potential change is $1-|S_L|$, and the amortized bound follows.