15-750 Algorithms (Spring 2002) **Prof. Manuel Blum**

Homework #1 **Solutions**

Problem 1: `make_heap` using `insert`'s (10 points)

Prove that the construction of a heap (`make_heap`$[a_1, \ldots, a_n]$) by repeated insertions takes $\Omega(n \log n)$ steps in the worst case, i.e., show that there exists a constant c such that for every n there exists an arrangement of the (totally ordered) elements $a_1, \ldots, a_n$ s.t. `make_heap`$[a_1, \ldots, a_n]$ takes at least $c\, n \log n$ steps to construct a heap. (This contradicts (!) Manuel's assertion at the end of lecture #1.)

Consider an arrangement in which the elements are inserted into the (min-)heap in decreasing order, i.e. $a_1 > a_2 > \ldots > a_n$. In this case, each inserted element is bubbled up from the bottom of the heap to the top, so it takes at least $\lceil \log k \rceil$ steps to insert element a_k. Therefore, the total cost C is

$$C = \sum_{k=1}^{n} \lceil \log k \rceil \geq \sum_{k=1}^{n} \log k = \log n! = \Theta(n \log n),$$

where the last identity (obtained by Stirling's formula) implies $C = \Omega(n \log n)$. □

Problem 2: min-max-med heap (40 points)

Design a data structure *min-max-med heap* (mmm-heap), that supports the following operations

Operation	Running time
`make_mmm-heap`$[a_1, \ldots, a_n]$	$\mathcal{O}(n)$
`find_min`$[h]$	$\mathcal{O}(1)$
`find_max`$[h]$	$\mathcal{O}(1)$
`find_med`$[h]$	$\mathcal{O}(1)$
`insert`$[h, a]$	$\mathcal{O}(\log n)$
`delete_min`$[h]$	$\mathcal{O}(\log n)$
`delete_max`$[h]$	$\mathcal{O}(\log n)$
`delete_med`$[h]$	$\mathcal{O}(\log n)$

where n denotes the number of elements in the mmm-heap h.

Hint: You may assume the existence of an algorithm that finds median of n elements in $\mathcal{O}(n)$ steps.

(solution by Charlie Garrod)

First, note that we can easily implement a min-max heap by maintaining two separate heap structures – one min heap and one max heap. Each heap contains the value of the element as well as a pointer to the associated entry in the other heap. Since constructing a min-max heap is exactly the same as constructing two normal heaps, *make_heap* is clearly $O(n)$. Also, *find_min* and *find_max* are clearly still $O(1)$ since we maintain a pointer to those elements at all times. *insert* is still $O(\lg n)$ since we simply insert the element into one heap, insert the element and corresponding pointer into the other heap, and finally fix the pointer in the first insertion. *delete_min* and *delete_max* are both $O(\lg n)$ since we easily delete the minimum or maximum element from min or max heap (respectively), and then use the pointer the entry in the other heap to delete it, bubbling up entries in the heap to maintain the heap structure as with a normal delete. Thus, the min-max heap supports the desired heap properties.

To create a min-max-med heap, we note that if the elements are broken into two groups such that all the elements in the first group are smaller than any element in the second group and the size of the first group is $\lfloor \frac{n}{2} \rfloor$, the minimum element is the minimum of the first group, the maximum element is the maximum of the second group, and the median element is the maximum of the first group. Thus, we can implement a min-max-med heap by maintaining two min-max heaps with these properties,

We implement *make_mmm_heap* by finding the median ($O(n)$ time), partitioning the elements into two groups using the median as the pivot ($O(n)$ time), and make one min-max heap from each group as described above, for a total of $O(n)$ time.

find_min is simply *find_min* on the "small" min-max heap, *find_max* is simple *find_max* on the "large" min-max heap, and *find_med* is simply *find_max* on the "small" min-max heap. Each of these is clearly $O(1)$.

To insert an element we first determine if it is larger than or smaller than the current median, and insert it in the appropriate min-max heap ($O(\lg n)$ time). If the size of the "small" heap is no longer $\lfloor \frac{n}{2} \rfloor$ we adjust the heap sizes by removing the extreme element (the max of the min-max heap of small elements or the min of the min-max heap of large elements) and inserting it in the other heap. This could take another $O(\lg n)$ time, making the total insert $O(\lg n)$.

delete_min, *delete_max*, and *delete_med* are each similar and trivial, simply deleting the correct extremal element from the correct min-max heap, and adjusting the heap sizes as with insertion if necessary. These operations are clearly all $O(\lg n)$.

Problem 3: `delete_min` in binomial heaps (extra credit)

In lecture #2 you learned how to delete the min-element from a binomial heap with n elements in $\mathcal{O}(\log n)$ steps (`delete_min`[h]), and how to insert an element in *amortized* $\mathcal{O}(1)$ steps (`insert`[h, a]). After class, Richard Dore gave us a beautiful 1-line proof, that it is impossible to have both, `delete_min`[h] and `insert`[h, a], in $\mathcal{O}(1)$ (amortized or not).

Is it possible in binomial heaps to have `insert`[h, a] in $\mathcal{O}(\log n)$ steps and `delete_min`[h] in *amortized* $\mathcal{O}(1)$ steps? Justify your answer.

Yes, it is possible. To justify this answer, we use the potential method of amortized analysis.

Let D_i denote the data structure after i-th operation, and $|D_i|$ be the number of elements in D_i. Moreover, let γ be a constant, such that the (actual) cost of `delete_min`[h] is bounded by $\gamma \log n$, where n is the number of elements in h. Define a potential function $\Phi(D_i)$ as

$$\Phi(D_i) = \gamma \sum_{k=1}^{|D_i|} \log k \ .$$

Clearly, $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0$, for all $i \geq 1$.

The amortized cost $\hat{c}_i$ of the i-th operation is

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \ .$$

From the above definitions we get the following bounds on the amortized costs:

`insert`[h, a]: $\hat{c}_i = c_i + \gamma \log |D_i| = \mathcal{O}(\log n)$

`delete_min`[h]: $\hat{c}_i = c_i - \gamma \log |D_i| = \mathcal{O}(1)$

□

Note: The above analysis does not include `meld`[h, h'] operations. Does the result hold also when `meld`s are allowed? How would you prove it?