600.363/463 Midterm Exam 1
10/14/2003

1 (15%)

Although merge sort runs in $\Theta(n \lg n)$ worst-case time, and insertion sort runs in $\Theta(n^2)$ worst-case time, the constant factors in insertion sort make it faster for small n. Thus, it makes sense to use insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which n/k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

a. What is the worst-case time to sort the n/k sublists (each of length k)?

b. Show that the sublists can be merged in $\Theta(n \lg(n/k))$ worst-case time.

c. What is the largest asymptotic (Θ-notation) value of k as a function of n for which the modified algorithm has the same asymptotic running time as standard merge sort?

Solution:
a. Omsertopm sprt takes $\Theta(k^2)$ time per k-element list in the worst case. Therefore, sorting n/k such that k-element lists takes $\Theta(k^2 n/k) = \Theta(nk)$ worst-case time.

b. (Just extending the 2-list merge to merge all the lists at once would take $\Theta(n \cdot (n/k)$ time.
The key idea is to merge the lists pairwise, then merge the resulting lists pairwise until there is only one list. the pairwise merging requires $\Theta(n)$ time at each level. The number of levels is $\lfloor \log(n/k) \rfloor$. So the totalrunning time is $\Theta(n \log(n/k))$.

c. The modified algorithm has the same asymptotic running time as standard merge sort when: $\Theta(nk + n \log(n/k)) = \Theta(n \log n)$.
So the largest asymptotic value of k as a function of n is $k = \Theta(\log n)$.

2 (25%)

Let $X[1..n]$ and $Y[1..n]$ be two arrays, each containing n numbers already in sorted order. Give an $O(\lg n)$-time algorithn to find the

median of all $2n$ elements in arrays X and Y.

Solution:
The median can be obtained recursively as follows. Pick the median of the sorted array A. This is just $O(1)$ time as median is the $n/2$th element in the sorted array. Now compare the median of A, call is a^* with median of B, b^*. We have two cases.

- $a^* < b^*$: In this case, the elements in $B[\frac{n}{2} \cdots n]$ are also greater than a^*. So the median cannot lie in either $A[1 \cdots \frac{n}{2}]$ or $B[\frac{n}{2} \cdots n]$. So we can just throw these away and recursively solve a subproblem with $A[\frac{n}{2} \cdots n]$ and $B[1 \cdots \frac{n}{2}]$.
- $a^* > b^*$: In this case, we can still throw away $B[1 \cdots \frac{n}{2}]$ and also $A[\frac{n}{2} \cdots n]$ and solve a smaller subproblem recursively.

In either case, our subproblem size reduces by a factor of half and we spend only constant time to compare the medians of A and B. So the recurrence relation would be $T(n) = T(n/2) + O(1)$ which has a solution $T(n) = O(\log n)$.

3 (15%)

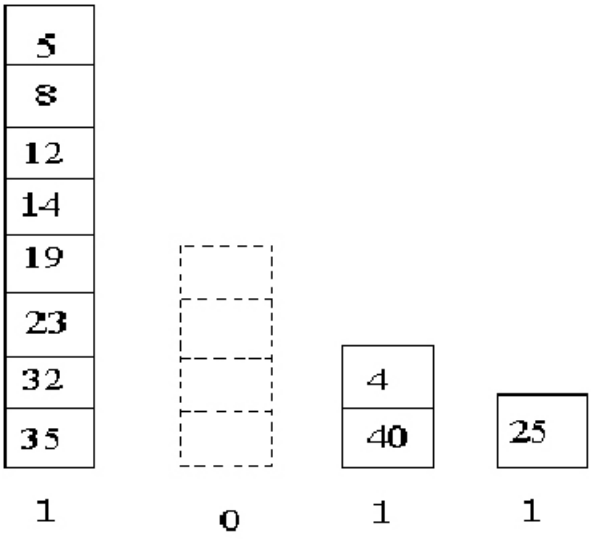
Suppose we use RANDOMIZED-SELECT to select the minimum element of the array $A = \langle 3, 2, 9, 0, 7, 5, 4, 8, 6, 1 \rangle$. Describe a sequence of partitions that results in a worst-case performance of RANDOMIZED-SELECT.

Solution:
In the worst case, each time the RANDOMIZED-SELECT will find the biggest element, and partition the whole set into a set of $k-1$ elements and a set of 1 element.
The worst-case partition is:
1. select 9, partition into: 3,2,0,7,5,4,8,6,1 and 9
2. select 8, partition the first set in the previous set into: 3,2,0,7,5,4,6,1 and 8.
... ...
9 select 1, partition the first set in the previous set into: 0 and 1

The worst-case time is: O(n^2).

4 (30%)

Binary search of a sorted array takes logarithmic search time, but the time to insert a new element is linear in the size of the array. We



can improve the time for insertion by keeping several sorted arrays. Specifically, suppose that we wish to support SEARCH and INSERT on a set of n elements. Let $k = \lceil \lg(n+1) \rceil$, and let the binary representation of n be $(n_{k-1}, n_{k-2}, ..., n_0)$. We have k sorted arrays $A_0, A_1, ..., A_{k-1}$, where for $i = 0, 1, ..., k-1$, the length of array A_i is 2^i. Each array is either full or empty, depending on whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all k arrays is therefore $\sum_{i=0}^{k-1} n_i 2^i = n$. Although each individual array is sorted, there is no particular relationship between elements in different arrays.

a. Describe how to perform the SEARCH operation for this data structure. Analyze the worst-case running time.

b. Describe how to insert a new element into this data structure. Analyze its worst-case and amortized running times.

The above is an example. There are 11 elements, and $11 = 1011_{two}$.

Solution:

a. Since the number of arrays is $\mathrm{O}(\log n)$, and search is done by performing a binary search in each, of size 1, 2, $2^2 ... 2^{\lfloor log_2^n \rfloor}$, and it takes $\Theta(\log_2 2^i) = \Theta(i)$ time to perform a binary search in each, the query time in the worst case is:

$$\Theta(\textstyle\sum_1^{i=\lfloor \log_2 n \rfloor} i) = \Theta(\log_2^2 n)$$

b. To perform insertion of a new element x: first create an array of size 1 for x. Then repeat: as long as there are two arrays of the same size, merge them into an array ofdouble size.
For the amortized cost, suppose the cost of inserting an element into a set of i elements is c_i. The amortized cost is: $\frac{\sum_{i=1}^{n} c_i}{n}$.
For simplicity we assume n is a power of 2. During the n operations, we need to merge an array of size $m = 2^k$ only after k insertions, and the merge process takes cm time (for a constant k). Hence the time needed for n insertions is:

$$cn + 2c\frac{n}{2} + 4c\frac{n}{4} + ... + c2^i\frac{n}{2^i} + ... + nc = cn\log_2 n$$

Thus the amortized time for an insertion is $\mathrm{O}(\log n)$.

5 (15%)

Show the data structure that results and the answers returned by the FIND-SET operations in the following program. Use the linked-list representation with the weighted-union heuristic.

```
1   for i ← 1 to 16
2       do MAKE-SET(x_i)
3   for i ← 1 to 15 by 2
4       do UNION(x_i, x_{i+1})
5   for i ← 1 to 13 by 4
6       do UNION(x_i, x_{i+2})
7   UNION(x_1, x_5)
8   UNION(x_11, x_13)
9   UNION(x_1, x_10)
10  FIND-SET(x_2)
11  FIND-SET(x_9)
```

Assume that if the set containing x_i and x_j have the same size, then the operation $UNION(x_i, x_j)$ appends x_j's list onto x_i's list.

Solution:
At the end of the execution, there is only one set which contains all elements $\{x_1, x_2, ...x_{16}\}$. In this set, for each i, x_i is linked to x_{i+1}. The set's representative is x_1.
So both of the two FIND-SET operations return the pointer to x_1.