Homework 4

22C:44 Algorithms, Fall semester 2000

Four problems, ten points each. Due in class on Thursday, Sept. 28.

1 Design algorithm **Delete** for deleting an arbitrary element of a heap. 
**Delete**($A[1\ldots n]$, $i$) should remove element $A[i]$ from heap $A$ and re-
arrange the remaining $n - 1$ elements to maintain the heap property. 
Your algorithm should run in $O(\log n)$ worst-case time. You may use 
any algorithms presented in the class as subroutines.

2 Exercies 8.1-1 (page 155) and 8.3-2 (page 163) of the text-book.

3 Design the modified **Partition** we used in the class during the analysis 
of the randomized quicksort. **Partition**($A,p,r$) should (a) use the 
first element $A[p]$ as the pivot, (b) compare the pivot and all other 
elements $A[p+1\ldots r]$ exactly once, (c) perform no other comparisons 
between the elements, (d) re-order the elements in such a way that the 
pivot moves to position $q$, for some $p \leq q \leq r$, all elements smaller 
than the pivot move before position $q$ and all elements larger than the 
pivot move after position $q$, (e) return number $q$, the new position of 
the pivot, and (f) run in the linear $\Theta(r-p)$ time.

4 Let us investigate the problem of sorting arrays $A[1\ldots n]$ that are 
known to be almost ordered initially in the sense that only some ele-
ments close to each other may be in the wrong order. More precisely, 
there exists a constant $c$ such that whenever two element $A[i]$ and $A[j]$ 
are in the wrong order then $|j-i| \leq c$.

(a) In this case, what is the worst-case time complexity of the 
**BetterBubbleSort** algorithm of Homework assignment # 1. Justify 
(=prove) your answer.

(b) Design a linear time algorithm based on quicksort. Analyze the 
complexity of your algorithm to prove that it indeed runs in the 
$\Theta(n)$ worst-case time. (Hint: modify the partitioning program to 
run in constant time.)