CS 381, Fall 1999

Solution Sketches to Sample Midterm Problems

1.) (i) Order the following functions according to their asymptotic growth rate. Indicate which functions belong to the same complexity class.
$4n\log n$, $2^n \log n$, $n^2 + 8n\log n$, 2^n, $(n+4)(n-6)$, $\sqrt{n}$, 2^{n-4}, $2^{n/2}$

$\sqrt{n} < 4n\log n < n^2 + 8n\log n,\ (n+4)(n-6) < 2^{n/2} < 2^n,\ 2^{n-4} < 2^n\log n$

(ii) $5n = O(n\log n)$: true; $5 \le 5\log n$ is true for all $n \le 2$
$12n^2 = O(n\log n)$: false; there exists no constant c such that $12n \le c\log n$ for all $n > N$, where N is fixed
$\frac{n}{\log n} = \Theta(n)$: false; $\frac{n}{\log n} = O(n)$ holds, but there exists no constant c such that $c \le \frac{1}{\log n}$ for all $n > N$, where N is fixed.

2.) Assume A is an array of size n containing integers in arbitrary order and A_s is an array of size n containing integers in sorted order. Give the running times (in big-O notation) for the specified operations. Give a brief explanation of each entry below the table (not given in this solution sketch).

	Given x, determine whether x is **not** in the array	Given x, determine whether x occurs at least $n/2$ times	Given x, determine the smallest element y in the array with $y > x$
A (not sorted)	$O(n)$	$O(n)$	$O(n)$
A_s (sorted)	$O(\log n)$	$O(\log n)$	$O(\log n)$

3.) (i) Use the master theorem to determine the tight asymptotic bounds of the following recurrence relations:

$$T(n) \le \begin{cases} T(n/2) + c\log n & \text{if } n > 2 \\ 1 & \text{for } n = 2 \end{cases}$$

We have $E = \frac{\log 1}{\log 2} = 0$, $n^E = 1$ and $f(n) = \log n$.
It is obvious that cases 1 and 2 of the master theorem do not apply. It can be easily shown that there is no positive ϵ for which $\log n = \Omega(n^\epsilon)$ Thus the master theorem does not apply.

We prove that $T(n) = \Theta(\log^2 n)$.

We will assume that $c \geq 1$. (For $c < 1$, minor modification have to be done since the base case may not hold.) Assume that $T(n) \leq dc \log^2 n$ for some constant d. For the basis case, we get $T(1) = 1 \leq dc$. If $d \geq 1/c$, the basis is satisfied. Assume the induction hypothesis holds for $n/2$. Then,
$T(n) \leq T(n/2) + c \log n$
$\leq dc \log^2(n/2) + c \log n = dc(\log n - 1)^2 + c \log n$
$\leq dc \log^2 n - 2dc \log n + dc + c \log n = dc \log^2 n - c \log n + dc.$
Set $d = \frac{1}{c}$. (Since $c \geq 1$, we have $d \leq 1$.) Then, $-c \log n + dc = -c \log n + 1 \leq 0$ holds since $\log n \geq 1$. Hence, $T(n) \leq dc \log^2 n$ and $T(n) = O(\log^2 n)$ follows.

$$T(n) = \begin{cases} T(n/2) + \sqrt{n} & \text{if } n > 2 \\ 1 & \text{for } n = 2 \end{cases}$$

We have $E = \frac{\log 1}{\log 2} = 0$, $n^E = 1$ and $f(n) = \sqrt{n}$.
Choosing ϵ equal to any fraction less than $1/2$ and δ equal to any fraction greater than $1/2$ shows that case 3 of the master theorem applies. It follows that $T(n) = \Theta(\sqrt{n})$.
(ii) Consider the recurrence relation $T(n) = 3T(n-1) + 2$ with $T(1) = 1$.
Show by induction that $T(n) = O(3^n)$.

Prove that $T(n) \leq c3^n - 1$ for some constants c. For the basis case, we get $T(1) = 1 \leq 3c - 1$. If $c \geq 2/3$ the basis holds. Assume the induction hypothesis holds for $n-1$. Then,
$T(n) = 3T(n-1) + 2 \leq 3(c3^{n-1} - 1) + 2 = c3^n - 1$. Then $T(n) < c3^n + 2$ and the claim holds.

4.) Describe a data structure to implement the following version of a priority queue Q. Each operation should take $O(\log n)$ time, where n is the current number of elements in Q.

Insert(Q, x) - insert element x into Q
DeleteMax(Q) - delete the largest element from Q
DeleteMin(Q) - delete the smallest element from Q.

Sketch how each operation is implemented and analyze the achieved time bounds.

We will use two heaps: a max-heap stored in array MAX having the maximum element at the root and a min-heap stored in array MIN with the minimum element at the root. Every element appears twice, once in MAX and once in MIN. In addition, for any element x in the min-heap, we create a pointer to element x in the max-heap and vice-versa.

In operation Insert(Q, x), element x is inserted into the max-heap as well as the min-heap. The insertion follows the procedure described in class.

Consider now DeleteMax(Q). In the max-heap, we delete element $MAX[1]$. This is done by placing the element at $MAX[n]$ at location $MAX[1]$ and then invoking the procedure to restore the heap property in MAX. This procedure proceeds from the root towards a leaf (it may terminate before a leaf is reached). The maximum element needs to be deleted from the min-heap MIN. The maximum element is stored at a leaf node in MIN and its location can be determined in constant time from $MAX[1]$ (before it is deleted). Let $MIN[l]$ be this location. Then, we place $MIN[n]$ at position $MIN[l]$ and invoke a procedure to fix up the heap property (the fix-up now proceeds from a leaf towards the root). After this, both heaps contain $n-1$ elements and the next query can be processed.

DeleteMin(Q) operates analogous to DeleteMax(Q) (with the roles of array MAX and MIN reversed). Each operation uses existing procedures to fix up necessary heap properties. In all cases, a constant number of path of length at most $\log n$ are traversed. Hence, each operation uses $O(\log n)$ time.

An interesting question is whether in a heap which supports fast min- and max finding we need to make two copies of every element. This is not desirable in many applications. There exist extensions (non-trivial) of the basic heap which have $O(\log n)$ time for MIN and MAX and which do not duplicate elements. This might be something for a future homework (in a future 381 class).

5.) Assume you are given two sets S_1 and S_2, which contain a total of n integers, and an integer x. Determine whether there exists an element in S_1 and an element in S_2 such that the sum of the two elements is equal to x. The running time should be $O(n \log n)$.

First, use an $O(n \log n)$ time sorting algorithm to sort S_1 as well as S_2. Then, for every element $S_1[i]$ less than x, search in S_2 for element $x - S_1[i]$ using binary search. The algorithm consists of two nested loops, one is executed at most n times (if all the elements of S_1 are less than x) and the other takes $O(\log n)$ time. So, the running time of the entire algorithm is $O(n \log n)$.

6.) Let A be an array of even size, say n, containing integers. The problem is to partition the elements in A into $n/2$ pairs with the following property: for every pair formed, determine the sum. Let $s_1, s_2, \ldots, s_{n/2}$ be these sums. Pairs should be formed so that the maximum of the s_i's is a minimum. Describe an efficient algorithm to determine a partitioning minimizing the maximum sum.

Sort the array and then pair up $A[i]$ with $A[n-i+1]$, $1 \leq i \leq n/2$. Correctness of this pairing would show that any other pairing has larger or equal maximum. Running time is $O(n \log n)$.