Purdue University | Hambrusch
CS 381 | Fall 2008

Sample Midterm Problems

Not to be handed in, solutions will be posted before the exam

1.) (i) Order the following functions according to their asymptotic growth rate. Indicate which functions belong to the same complexity class. Explain your answers.

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

(ii) Which are true? Explain your answers.
$5n = O(n \log n)$
$12n^2 = O(n \log n)$
$\frac{n}{\log n} = \Theta(n)$
$4n = \Omega(\sqrt{n} \log n)$

2.) Assume A is an array of size n containing integers in arbitrary order, A_s is an array of size n containing integers in sorted order, and A_h is an array of size n containing integers arranged in a min-heap. Give the running times (in big-O notation) for the specified operations on a given element x. Give a brief explanation of each entry below the table.

	determine whether x is in the array	determine whether x occurs at least $3n/4$ times	determine whether x is smaller than the smallest element in the array
A (not sorted)			
A_s (sorted)			
A_h (min-heap)			

3.) (i) Determine whether the Master Theorem can be applied. If yes, use it to find a tight asymptotic bound.

$T(n) = 8T(n/4) + 6n$ for $n > 4$ and $T(i) = 4$ for $i \leq 4$
$T(n) = 8T(n/4) + n^2$ for $n > 4$ and $T(i) = 14$ for $i \leq 4$
$T(n) = 4T(n/3) + n^2 \log n$ and $T(i) = 1$ for $i \leq 3$
$T(n) = 2T(n/4) + \sqrt{n}$ and $T(i) = 1$ for $i \leq 4$
$T(n) = 2T(n/2) + n \log n$ and $T(i) = 1$ for $i \leq 2$

(ii) Consider the recurrence relation $T(n) = 3T(n-1) + 2$ with $T(1) = 1$.
Show by induction that $T(n) = O(3^n)$.

4.) (i) Let S be a set of n elements which contains only 8 distinct elements. You do not know these 8 elements. Describe and analyze an efficient algorithm to sort set S under this assumption.

(ii) Assume you are given two sets S_1 and S_2 (not sorted), 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 $O(n \log n)$.

(iii) Let S be a set of size n containing integers. Design an $O(n)$ time algorithm to determine an integer *not* in S. Give an argument why $\Omega(n)$ is a lower bound for this problem.

(iv) Show that in an unsorted array of size n, the sum of the $2 \log n$ smallest elements can be computed in $O(n)$ time.

(v) Assume that N persons checked their coats in a restaurant. The coats get mixed and each person gets back a random coat. Compute the expected number of people getting back their own coat.

(vi) Is it true that counting sort will always sort an array of n integers from $\{1, 2, ..., m\}$ in $O(n)$ time?

5.) For an algorithm solving a problem of input size n, let $B(n)$ be the best-case, $A(n)$ be the average, and $W(n)$ be the worst case time performance. For each statement below give an algorithm seen in class which satisfies the claim. Give the name of the algorithm (or a brief description) and state its time bounds (if not already given).

(i) $B(n) = A(n) = W(n)$ (i.e., all three bounds are asymptotically the same)
(ii) $B(n) = O(1)$ and $W(n) = O(\log n)$
(iii) $A(n) = O(W(n))$, but $W(n) = O(A(n))$ does not hold

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.
(i) For $A = [4, -6, 14, 8, 1, 5, -2, 23, 7, 15]$, give the partition into pairs minimizing the maximum sum.
(ii) Describe an efficient algorithm to determine a partitioning minimizing the maximum sum.

7.) Given a sequence A of n distinct integers, $a_1, a_2, \ldots a_n$, the goal is to find a longest in-

creasing subsequence of A. For example, if A is $10, 4, 5, 20, 7, 30, 9$, then a longest increasing subsequence of A is 4,5,7,9.
(i) Give another longest increasing subsequence in the above example.
(ii) Consider the greedy algorithm described below. Give the running time and explain your answer.

for $i = 1$ to n do
    Start a (sub)sequence S_i that has a_i as the first element
    for $j = i + 1$ to n do
        Add a_j to S_i if it is greater than the current last element in S_i
    endfor
endfor
Output the subsequence S_i that has the maximum length.

(iii) Does the above greedy algorithm always output a longest increasing subsequence?