CSE 101

Sample Midterm: Spring, 2003

Answer all questions. Give informal (at least) proofs for all answers. Grading will be on completeness and logical correctness, as well as correctness. For the data structure question, the efficiency of your solution will also be taken into account.

Analyzing loops-10pts Consider the following iterative algorithm, that uses an $\Theta(I)$ time procedure $proc(I)$.

Algorithm(n: positive integer);

1. begin;
2. $T \leftarrow 1$;
3. While $T^2 \leq n$ do:
4. begin;{while}
5. $proc(T^2)$;
6. $T{+}{+}$;
7. end;{while}
8. end;

Give a time analysis, up to Θ, for this algorithm.

Proof of correctness - 10 pts Below is an algorithm, that, given a *sorted* array $A[1..n]$ of integers, and an integer T, decides whether there is a pair $1 \leq K < L \leq n$ with $T = A[K] + A[L]$. After the algorithm, there is a proof of correctness, with some parts missing. Fill in the missing sections in the proof to get a complete proof of correctness.

IsSum[A[1,,n]: sorted array of integers, T: integer];

1. $Found \leftarrow False$;
2. $I \leftarrow 1$;
3. $J \leftarrow n$;
4. While $I < J$ and NOT $Found$ do:
5. begin;{while}
6.   IF $A[I] + A[J] = T$ THEN $Found \leftarrow True$;
7.   IF $A[I] + A[J] > T$ THEN J– ELSE I++;
8. end; {while}
9. Return Found.

Proof of correctness: Let A[1..n] be a sorted array, i.e., . Let T be any integer. We need to show that if for $1 \leq K < L \leq n$ then $IsSum$ returns $True$; and coversely, that if $IsSum$ returns $True$, then for $1 \leq K < L \neq n$,

To see the first direction, we use the loop invariant method. Assume there are $1 \leq K \leq L \leq n$ with . Let I_t and J_t be the values of I, J after t iterations of the while loop. The intuition is that K, L never leave the range I_t, J_t to be searched by the algorithm, i.e, that $\leq K < L \leq$.

We prove this by . First we show that it is true at $t = 0$. At initialization, $I_0 =$ and $J_0 =$. Thus, $\leq K < L \leq$. So the is true.

For the induction step, assume the statement is true after some number of loops t, i.e. that . We want to show that after the next iteration, .

There are three cases: if $A[I_t] + A[J_t] = T$, the algorithm , and there is nothing to prove.

If $A[I_t] + B[J_t] > T$, then in the next iteration $I_{t+1} =$ and $J_{t+1} =$. Thus, the only way we could not have what we want to show, $\leq K < L \leq$, is if $=$. Now if $=$, then

$< A[I_t] + A[J_t] = A[I_t] +$ $\le A[K] +$ $=$. This is a , so we have shown by that in this case , which is what we wanted to prove.

The final case $<$, is very similar. In the next iteration $I_{t+1} =$ and $J_{t+1} =$. Thus, the only way we could not have what we want to show, $\le K < L \le$, is if $=$. Now if $=$, then $> A[I_t] + A[J_t] =$ $+ A[J_t] \ge$ $+$ $=$ This is a , so we have shown by that in this case , which is what we wanted to prove. Thus, the loop invariant still holds in all three cases.

The first direction follows from the loop invariant, since at the end of the loop, either we return $True$ or $=$. If $=$, by the loop invariant, we must have $I_t \le$ $<$ $\le J_t = I_t$, which is a contradiction. Thus, if $A[K] + A[L] = T$, the algorithm returns $True$.

To see the converse direction, if $IsSum$ returns $True$, then $Found$ must be set to $True$, which can only happen in the THEN clause of line . Because of the IF clause of that line, we must have , which is what we were proving, since we can pick $K =$ and $L =$.

Using data structures and pre-processing: Consider the following problem. The input is a $m \times n$ matrix $A[1..m][1..n]$ where each row is sorted, $A[I][1] \le A[I][2] \le ...A[I][n]$. The problem is to combine all of these sorted rows into a single sorted array $B[1...nm]$, with $B[1] \le B[2] \le .... \le B[nm]$.

A strategy to solve this problem is as follows: Here S represents the set of smallest elements from each list that are not yet output.

AbstractMergeLists(A[1..m][1..n]: matrix of integers)

1. Initialize counters $C[I], 1 \le I \le m$ to 1, and a set S of possible next elements in the sorted list to the smallest element from each list, $A[1][1], A[2][1], ...A[m][1]$.
2. FOR $J = 1$ TO mn do:
3. begin; {for}
4. Let X be the smallest element in S.
5. Let I be the row X is from.
6. Delete X from S.
7. $B[J] \leftarrow X$
8. $C[I]++$
9. IF $C[I] \le n$ THEN Add $A[I][C[I]]$ to S.
10. end; {for}

Describe what data structures you would use in an algorithm to implement the above strategy. Describe how you initialize the data structures. Then rewrite each line of the strategy with the corresponding operations in your data structures. Give a time analysis of the resulting algorithm, in terms of both n and m.

Example: Say $m = 3, n = 5$ and the array is: A[1][1..5]=3,8, 17, 38, 43
A[2][1..5]=18, 19, 35, 41, 52
A[3][1..5]=1,2,7,10,11

We start $S = \{3, 1, 18\}$. Since the smallest element of S is 1, from the third row, we define $B[1] = 1$ and replace 1 with the next element of the third row, 2. Now $S = \{2, 3, 18\}$. So we define $B[2] = 2$, and swap 2 with the next element of the third row, 7. Now the smallest element in S is 3 from row 1, and we define $B[3] = 3$ and swap 3 for 8. So now $S = \{7, 8, 18\}$. We then set $B[4] = 7$, adding 10 to S, set $B[5] = 8$, adding 17 to S. Now $S = \{10, 17, 18\}$. We set $B[6] = 10$, $B[7] = 11$, and now the counter for the third list is past 5, so we don't replace 11 by anything. So at this point $S = \{17, 18\}$. We define $B[8] = 17$, and $S = \{18, 38\}$; $B[9] = 18$ and $S = \{19, 38\}$, $B[10] = 19$, $S = \{35, 38\}$, $B[11] = 35$, $B[12] = 38, B[13] = 41, B[14] = 43$. At this point S just contains 52 from list 2. So $B[15] = 52$ and we're done.

Divide-and-Conquer Recurrence: 25 points Consider the following recursive algorithm. Its inputs are two *sorted* n element arrays A and B (so the algorithm assumes $A[1] < A[2] < ... < A[n]$ and $B[1] < B[2] < ... < B[n]$) and a target T. The following algorithm computes whether $T = A[I] + B[J]$ for some $1 \leq I, J \leq n$. Give a recurrence relation for the time the algorithm takes, and give a brief explanation of your answer. Use the recurrence to give a time analysis.

1. Program: TargetSum(A[1..n], B[1..n]: Arrays of Integers, T: integer):
2. IF $n = 0$ return False;
3. IF $n = 1$
   THEN IF $T = A[1] + B[1]$
   THEN return True;
   ELSE return False;
4. $k \leftarrow \lceil n/2 \rceil$
5. $k' \leftarrow \lceil (n+1)/2 \rceil$
6. IF $A[k] + B[k] = T$ THEN return True;
7. IF $A[k] + B[k] > T$ THEN return
   $(TargetSum(A[1..k], B[1..k], T)$
   OR $TargetSum(A[1..k], B[k'..n], T)$
   OR $TargetSum(A[k'..n], B[1..k], T))$;
8. ELSE return
   $(TargetSum(A[k'..n], B[1..k], T)$ OR
   $TargetSum(A[1..k], B[k'..n], T)$ OR
   $TargetSum(A[k'..n], B[k'..n], T))$;

Here, OR is the Boolean OR operation on the values of the returned calls.

Example: say $n = 4, A[1..4] = 3, 8, 13, 21; B[1..4] = 1, 2, 3, 5; T = 13$. Then the algorithm would first compare $A[2] + B[2] = 10$ to T. Since it is smaller, it would recursively check TargetSum[A[1..2],B[3..4],13], TargetSum[A[3..4],B[1..2],13] and TargetSum[A[3..4],B[3..4],13]. (It could ignore the case $I, J \leq 2$, since all such sums are less than T). After computing recursively, which I won't show here, the sub-procedures would return $True$, $False$ and $False$. Thus, the main procedure would return $True$.