CS524

MIDTERM EXAM

Name_____

Date: October 24, 2005
All documentation permitted

1. (25 points) Give an exact closed-form solution to the recurrence

$$T(n) = \begin{cases} 2T(n-1)+3, \text{ if } n>1 \\ 1, \text{ if } n=1 \end{cases}.$$

2. (25 points) Suppose you want to find the $42^{\underline{nd}}$ largest element, x, of an array $A[1..n]$ of $n \geq 42$ distinct elements. A contains exactly 41 elements larger than x.Your benchmark operations are pairwise comparisons.

a Describe an algorithm to solve this problem which uses n-1 comparisons in the best case.

b Describe a $O(n)$ upper bound on the worst case complexity of this problem.

c Show that at least $n + \lceil \lg n \rceil - 2$ comparisons are necessary in the worst case.

3. (25 points) Assume you are given two sorted arrays $A[1..n]$ and $B[1..n]$ of $2n$ distinct elements. Describe an algorithm to find the median of these $2n$ elements. That is, find the n^{th} smallest of these $2n$ elements. The time complexity of your algorithm should be in $O(\lg n)$. You may assume that n is a power of 2. For example, if A=(18, 95) and B=(10, 99), then your algorithm should return 18.

4. (25 points) Assume that you have to implement a counter which initially contains an integer *m*, and you want to perform *n* INCREMENTs on the counter. Ultimately the counter should contain *m*+*n*. For example, if *m* =00011010 (the initial state of the counter), then after each of three INCREMENTs the counter will contain

Operation	counter
INCREMENT	00011011
INCREMENT	00011100
INCREMENT	00011101

Assume the binary representation of *m* contains $O(n)$ 1's. Show that for any value of *m* (for any intial state of the counter) the cost of performing *n* INCREMENTs is in $O(n)$, where the benchmark operation is performing a bit change in implementing the counter.

CS524

Solutions to Midterm Exam

1. Unfolding the recurrence yields (for $n \geq 3$)

$T(n) = 2T(n-1) + 3$

$= 2\left(2T(n-2)+3\right)+3 = 4T(n-2)+6+3$

$= 4\left(2T(n-3)+3\right)+6+3 = 8T(n-3)+12+6+3$.

After *k* levels of unfolding we get $T(n) = 2^k T(n-k) + 3\sum_{0 \leq i \leq k-2} 2^i$. After *n*-1 levels of unfolding we get $T(n) = 2^{n-1}T(1) + 3\sum_{0 \leq i \leq n-2} 2^i = 2^{n-1} + 3\left(2^{n-1}-1\right) = 2^{n+1} - 3$.

2. ***a*** We first check if *A* is already sorted (using *n*-1 comparisons). If not, we sort the list. Finally, we **return** *A*[*n*-41].

```
sorted? ← true
for i ← 1 to n-1
    if A[i] < A[i+1] then sorted? ← false
if not sorted? then HEAPSORT(A)
return A[n-41]
```

b

BUILDHEAP(*A*)	$\Theta(n)$
for $i \leftarrow 1$ **to** 41 **do** HEAP-EXTRACT-MAX(*A*)	$\Theta(\lg n)$
return HEAP-EXTRACT-MAX(*A*)	$\Theta(\lg n)$

c In order to "know" the 42nd largest, the algorithm must "know" the 41 elements of *A* which are larger than the 42nd largest. We showed in class that $n + \lceil \lg n \rceil - 2$ comparisons are necessary in the worst case to find the 2 largest elements of *A*.

3. This problem is like the nuts-&-bolts problem.

```
MEDIAN (A,B,n)
  if n=1
  then return min(A[1], B[1])
  else
    if A[n/2] < B[n/2]
    then
        remove the n/2 smallest elements of A
        remove the n/2 largest elements of B
        n ← n/2
        MEDIAN (A,B,n)
    else
        remove the n/2 smallest elements of B
        remove the n/2 largest elements of A
        n ← n/2
```

MEDIAN (A,B,n)

4. We use amortized analysis, using the accounting model. We assume that flipping a bit costs \$1. We invest \$1 $O(n)$ times to place \$1 on each 1-bit in the counter. We then charge \$2 for each of n INCREMENTs. The invariant that we maintain is that \$2 enters to execute each bit flip, and there is always \$1 on each 1-bit. Every time a 0-bit is flipped, we use \$1 to cover the flip and leave \$1 on the bit. When \$2 enters to flip a 1-bit, it combines with the \$1 on the 1-bit to cover \$1 to flip the 1-bit and then send \$2 up to cover the cost of flipping higher order bits.