NJIT New Jersey's Science & Technology University

A. V. Gerbessiotis

Feb 21, 2008

Exam 1 (167 points)

CS 610-104

Spring 2008

Name: .....

ID Number: ..... **Exam Number:** .....

Grade: 1: ... 2: ... 3: ... 4: ... 5: ... 6: ... Total: .....

Solve ALL 6 PROBLEMS in the space provided.

Read the Problems CAREFULLY!

There are 4 (FOUR) pages this page included

List of useful formulae

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4} \qquad n! \approx \left(\frac{n}{e}\right)^n, \qquad a^{\log_b n} = n^{\log_b a},$$

For $x \neq 1$, we have that

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}, \qquad \sum_{i=0}^{n-1} ix^i = \frac{(n-1)x^{n+1} - nx^n + x}{(1-x)^2},$$

B1. $f(n) = \Theta(g(n))$ iff $\exists$ positive constants $c_1, c_2, n_0 :\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\ \ \forall\ n \geq n_0$.
B2. $f(n) = \Omega(g(n))$ iff $\exists$ positive constants $c_1, n_0 :\ 0 \leq c_1 g(n) \leq f(n)\ \ \forall\ n \geq n_0$.
B3. $f(n) = O(g(n))$ iff $\exists$ positive constants $c_2, n_0 :\ 0 \leq f(n) \leq c_2 g(n)\ \ \forall\ n \geq n_0$.

Master Method. $T(n) = aT(n/b) + f(n)$, such that $a > 0$, $b > 1$, $f(n) > 0$.

M1 If $f(n) = O(n^{\lg_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\lg_b a})$.

M2 If $f(n) = \Theta(n^{\lg_b a} \lg^k n)$, then $T(n) = \Theta(n^{\lg_b a} \lg^{k+1} n)$, where $k \geq 0$ is a non-negative constant.

M3 If $f(n) = \Omega(n^{\lg_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and for large n, then $T(n) = \Theta(f(n))$.

Problem 1. (40 POINTS)

Give a **TRUE** or **FALSE** for each one of the statements below. Any answer other than a full TRUE or FALSE will be considered wrong. All algorithms are the book/notes implementations.

(1) Insertion-Sort sorts in-place.

(2) MergeSort sorts in-place.

(3) On the input sequence $\langle 1, 2, \ldots, n\rangle$, Insertion-Sort is asymptotically faster than MergeSort.

(4) On the input sequence $\langle n, n-1, \ldots, 1\rangle$, InsertionSort is asymptotically faster than MergeSort.

(5) On the input sequence $\langle n, n-1, \ldots, 1\rangle$, InsertionSort has running time that is $\Omega(n)$.

(6) On the input sequence $\langle 1, 2, \ldots, n\rangle$, InsertionSort has running time that is $O(n^2)$.

(7) The asymptotic solution of $T(n) = T(n/2) + \lg n$ is $T(n) = \Theta(n)$.

(8) The asymptotic solution of $T(n) = 3T(n/2) + T(2n/4) + n^2$ is $T(n) = \Theta(n^2)$

Problem 2. (30 POINTS)

Rank the following functions by order of growth; that is, find an arrangement $g_1, g_2, g_3, \ldots, g_6$ of the functions satisfying $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3)$, $g_3 = \Omega(g_4)$, $g_4 = \Omega(g_5)$, $g_5 = \Omega(g_6)$. Partition your list in equivalence classes such that $f(n)$ and $h(n)$ are in the same class if and only if $f(n) = \Theta(h(n))$. For example for functions $\lg n, n, n^2$, and $2^{\lg n}$ you could write: n^2, $\left\{n, 2^{\lg n}\right\}$, $\lg n$.

$$2^n, \quad 3^{\lg n}, \quad 2^{3\lg n}, \quad n^2 \lg n, \quad \lg(n!), \quad n!.$$

Problem 3. (17 POINTS)

Show that $n^3 - 2n^2 + 1 = \Theta(n^3)$ by providing the c_1, c_2, n_0 of the definition.

Problem 4. (30 POINTS)

What does the following program do if called on an array of n keys (very last line of the code below)? Explain. Analyze its worst-case running time and provide a tight asymptotic bound for it.

```
   FindMe(keys A[0..n-1], int p, int r)
0.  int q;
1.  if (p==r)
2.    return(TRUE);
3.  if (A[p] != A[r])
4.    return(FALSE);
5.  q= (p+r)/2;
6.  return ( FindMe(A,p,q) && FindMe(A,q+1,r))

1.   FindMe(A[0..n-1],0,n-1);
```

Problem 5. (30 POINTS)

Solve exactly the following recurrence. You may assume that n is a power of two.

$$T(n) \quad = \quad 2T(n/2) + 6n, \qquad T(4) = 12.$$

Problem 6. (20 POINTS)

We have an array A[0..n-1] of n keys that have at most k distinct values, where $k \leq \sqrt{n}$. (The fact that k is bounded is known in advance.) We intend to sort this array in time faster than the generic $\Omega(n \lg n)$ depending on the value of k, by taking into consideration that there are not many distinct values among the n keys. We will do so in two rounds of computation. In the first round, we will compute a sorted array X[0..k-1] that contains the k distinct keys of A. Then, in the second round we can sort A using X as a guidance. You are asked to implement the first step of this algorithm.

Design an algorithm that given A, k and n as input determines the $k \leq \sqrt{n}$ distinct keys of the input, and stores them in array X in sorted order without using more than $\Theta(1)$ additional extra space. Your algorithm should run in worst-case running time of $O(n \lg k)$. Note once again that $k \leq \sqrt{n}$. Briefly explain the algorithm, comment on its correctness, and analyze its worst-case running time.

Observation. $O(n \lg k)$ is $O(n)$ if k is constant; if $k = O(\lg n)$ then $O(n \lg k) = O(n \lg \lg n)$, which is $o(n \lg n)$.

This is the end of the exam.