NJIT New Jersey's Science & Technology University

A. V. Gerbessiotis

Mar 13, 2008

Exam 2 (333 points) in 2 hours

CS 610-104

Spring 2008

Name: .....SOLUTION OUTLINE.....

ID Number: ..... **Exam Type:** ...A... **Exam Number:** .....

Grade: 1: ... 2: ... 3: ... 4: ... 5: ... 6: ... 7: ... Total: .....

Solve all **7 (SEVEN)** problems in the space provided

Read the Problems CAREFULLY!

There are 6 (SIX) pages this page included; the last page is left blank

STATEMENT

On my honor, I pledge that I have not violated the provision of the NJIT Student Honor Code.

Sign below at the end of the exam

Signature .....

Any algorithm you present must be given in concise and complete form. Argue about its correctness. Analyze its (worst-case) running time and express it in asymptotic notation. Random ramblings or sketches will not be given any points. You may use algorithms presented in class as black-boxes without further description. For example, instead of repeating the code of MergeSort you can just write MergeSort(B,m) to indicate that you sort an array B of m keys.

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 : $0 \leq f(n) \leq c_2 g(n)$ $\forall\, n \geq n_0$.

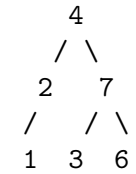
Master Method. $T(n) = aT(n/b) + f(n)$, such that $a > 0$, $b > 1$, $f(n) > 0$.

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. (50 POINTS)
Inorder : $1, 2, 4, 3, 7, 6$. **Preorder** : $4, 2, 1, 7, 3, 6$.



Problem 2. (50 POINTS)

$$\begin{aligned} T(n) &= T(n/2) + n. \\ T(n) &= 2T(n/2) + \sqrt{n}. \\ T(n) &= T(n-5) + n \end{aligned}$$

a. By the master method case 3, we have $T(n) = \Theta(n)$. To satisfy $af(n/b) \leq cf(n)$ just choose $c = 1/2$.
b. By the master method case 1, we have $T(n) = \Theta(n)$.
c. $T(n) = T(n-5) + n = T(n-10) + (n-5) + n = T(n-5i) + (n-5(i-1)) + \ldots + (n-5\cdot 1) + (n-5\cdot 0)$. If $i = n/5$ we have assuming $T(0) = 0$ (any other constant $T(.) = .$ works ok as well), we have $T(n) = n \cdot i + 5 \cdot ((i-1) + (i-2) + \ldots + 1 + 0)$. Thus $T(n) = n^2/5 + 5i(i-1)/2 = n^2/5 + 5(n/5)(n/5-1)/2 = \Theta(n^2)$. In other words $T(n) = \Theta(n^2)$.

Problem 3. (50 POINTS)

$$\begin{aligned} T(n) &= 2T(n/2) + 3n \\ &= 2^2 T(n/2^2) + 2 \cdot 3n \\ &= 2^i T(n/2^i) + i \cdot 3n \\ &= 2^{\lg n - 3} T(n/2^{\lg n - 3}) + (\lg n - 3) \cdot 3n \\ &= (n/8)T(8) + 3n \lg n - 9n \\ &= (n/8)8 + 3n \lg n - 9n \\ &= n + 3n \lg n - 8n \\ &= 3n \lg n - 8n \end{aligned}$$

We used in the fourth equality the facts that $T(8) = 8$ and $2^{\lg n - 3} = n/8$.

Problem 4. (50 POINTS)

(1) Between MergeSort and RadixSort, the former is more time-efficient in sorting n keys in the range $0, \ldots, n^3 - 1$. **FALSE**, RadixSort (i.e. 3 rounds of CountSort) gives $\Theta(n)$ running time.

(2) Between MergeSort and RadixSort, the former is more time-efficient in sorting n keys in the range $0, \ldots, n^n - 1$. **TRUE**, the former is $O(n \lg n)$ but the latter is $\Omega(n^2)$.

(3) Between HeapSort and RadixSort, the former is more time-efficient in sorting n keys in the range $0, \ldots, 2^n - 1$. **TRUE** the former is $O(n \lg n)$ but the latter requires $\Omega(n^2/\lg n)$ time.

(4) The solution of $T(n) = T(n/5) + T(7n/10) + n$, is $T(n) = \Theta(n \lg n)$. **FALSE** by the discussion on Select we know that $T(n) = \Theta(n)$.

(5) CountSort does not sort in-place. **TRUE**.

Problem 5. (50 POINTS)

Perform binary search on the integers in the range $1, \ldots, n$. This takes $O(\lg n)$. If you want to optimize further, you can perform binary search in the interval $1, \ldots, \sqrt{n}$. However this will still give an $O(\lg n)$ algorithm.

Let in the beginning of the BinSearch $l = 1$ and $r = n$. Find the middle point $m = (l + r)/2$. If $m * m == n$ return m, which is the answer. Otherwise if $m * m < n$ recur on the $m + 1, \ldots, r$ interval; else recur on the $l, \ldots, m$ interval. The base case is when $l == r$. If $l * l == n$ return l, which is the answer else return not found.

Problem 6. (50 POINTS)

(a) Sort X and Y using MergeSort/HeapSort. This takes $O(n \lg n)$ time.

Find the m largest keys of X and Y in $O(m)$ time respectively. This is easy since X, Y are sorted. This $O(m)$.

Then find all possible m^2 pairs of the largest values and sort the sums of the pairs using HeapSort/MergeSort in $O(m^2 \lg m)$ time.

The m rightmost (i.e. largest) of the m^2 values are printed out and provide the answer in sorted order. This step takes $O(m)$ time.

Overall time is $O(n \lg n + m^2 + m^2 \lg m + m)$. Since $m \leq \sqrt{n})$ we have $O(n \lg n + m^2 + m^2 \lg m) = O(n \lg n + n + n \lg n) = O(n \lg n) = o(n^2)$.

The pseudocode looks as follows.

```
   Find(X,Y,n,m)
 1. HeapSort(X,n);  // Sort X
 2. HeapSort(Y,n);  // Sort Y
 3. allocate A[0..m-1] and B[0..m-1]
 4. for(i=0;i<m;i++) {
 5.   A[i] = X[n-m+i];  B[i] = Y[n-m+i];  // m  largest X values in A, Y values in B
 6. }
 7. allocate C[0...m*m-1]
 8. for(i=0;i<m;i++)
 9.   for(j=0;j<m;j++)
10.    C[i*m+j ] = A[i]+B[j] ;    // Form all m*m sums x_i + y_j
12.HeapSort(C,m*m);               // Sort all m*m values
13.for(i=0;i<m;i++)               // Print largest m ones!
14. print(C[m*m-m+i]);
```

(b) The sorting operations could have been eliminated if we had used Selection instead in steps 1,2, and 12. The use of selection returns only one element, the requested statistic (m in the first two invocations, and m^2 in the latter); after each selection we need to go through the keys of the input and select those greater than the statistic, a linear operation. The use of selection makes our algorithm run in time $O(n)$ instead.

Problem 7. (33 POINTS)

Since A, B, C have sizes $\lg n$, n and n^2, it means that D can have size at most $\lg n$, the size of the smallest of the three sets. In other words D must be a subset of A. Perform a binary search of A into B and C. This takes $\lg n$ (the size of A) times log the size of B (n) plus log the size of C (n^2). The overall running time is thus $\lg n \cdot (\lg (n + 1) + \lg (n^2 + 1)) = O(\lg^2 n)$.

```
  Intersect(A,B,C,n)
0. k=0;
1. for(i=0;i<lgn;i++)
2.  D[i]= empty;
3. for(i=0;i<lgn;i++)
4.   b = BinSearch(A[i],B,n);     // lg(n+1) time this Binary Search
5.   c = BinSearrc(A[i],C,n**2)   // lg(n**2+1) time this one
6.   if (b != NOTFOUND) && (c != NOTFOUND) {  // i.e. A[i] is in B and in C
7.      D[k] = A[i]; k++;
8.   }
9. return(D[0]...D[k-1]);
```