Department of Computer Science Fall 2007 New Mexico State University

Ph.D. Qualifying Exam: Analysis of Algorithms

This is a closed book exam. The total score is 100 points. Please answer all questions.

(20 points) 1. This problem tries to establish that polynomial running time is asymptotically bounded from above by exponential running time.

Prove $n^{2.1} = o(15^n)$. Hint: You may take advantage of the fact that $\ln x \leq x - 1$ for any $x > 0$. Also note that $\ln 15 \approx 2.7$.

Solution:

$$n^{2.1} < c15^n \tag{1}$$
$$\Leftrightarrow 2.1 \ln n < \ln c + n \ln 15 \tag{2}$$
$$\Leftarrow 2.1(n-1) < \ln c + n \ln 15 \tag{3}$$
$$\Leftrightarrow n > \frac{-2.1 - \ln c}{\ln 15 - 2.1} \tag{4}$$

Thus one can choose $n_0 = \lceil \frac{-2.1-\ln c}{\ln 15 - 2.1} \rceil + 1$.

(35 points) 2. An array $A[1 \ldots n]$ is said to have a *majority element* if more than half of its entries are the same. The elements of the array are not comparable, i.e., the truth of $A[i] > A[j]$ is not defined. But the truth of $A[i] = A[j]$ or $A[i] \neq A[j]$ are uniquely defined. For example, the array can be

$$A = (\spadesuit, \clubsuit, \heartsuit, \spadesuit, \clubsuit, \spadesuit, \diamondsuit, \spadesuit, \spadesuit)$$

In this case, "$\spadesuit$" would be the majority element. We also define the majority element of a singleton array is just that singleton element. The problem is to detect a majority element from the array A of size n if it exists or simply report the array does not have a majority element.

You will not be able to apply any hash function or indexing on the elements in A.

You may find the Master Theorem useful in deriving the running time.

Theorem 1 (Master Theorem)**.** *Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence*

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows.

1. *If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.*
2. *If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \lg n)$.*
3. *If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.*

Show how to solve this problem in $O(n \log n)$ time. (Hint: Split the array A into two arrays A_1 and A_2 of half the size. Observe the relations of the majority elements among A, A_1 and A_2.) You must justify your algorithm and analyze the running time.

Solution:

The algorithm:

```
FIND-MAJORITY-ELEMENT-1(A, p, r)
 1  if (p = r) return A[p]
 2  q ← ⌈(p + r)/2⌉
 3
 4  m ← FIND-MAJORITY-ELEMENT-1(A, p, q)                ▷ First half
 5  if (m ≠ NIL)
 6     then count ← 0
 7          for i ← p to r
 8              do if (A[i] = m) count ← count + 1
 9                 if (count > n/2) return m
10
11  m ← FIND-MAJORITY-ELEMENT-1(A, q + 1, r)            ▷ Second half
12  if (m ≠ NIL)
13     then count ← 0
14          for i ← p to r
15              do if (A[i] = m) count ← count + 1
16                 if (count > n/2) return m
17  return NIL
```

Justification:

(1) If A has a majority element, it must be the majority element of one partition.

Let the partitions have size n_1, n_2 and $n_1 + n_2 = n$. Let the number of m in partition one be k_1 and k_2 for partition 2.

By contradiction. If $k_1 < \frac{n_1}{2}$ and $k_2 < \frac{n_2}{2}$, then

$$k_1 + k_2 < \frac{n_1}{2} + \frac{n_2}{2} = \frac{n}{2}$$

which contradicts that m is the majority element of A.

(2) If A has no majority element, the partitions may or may not have their own majority elements.

For example, $A = 1,2,3,5,3,4,7$, we get two partitions

$$1,2,3,5 \text{ and } 3,4,7$$

and none of them have majority elements

For another example, $A = 1,2,3,5,7,4,7$, we get two partitions

$$1,2,3,5 \text{ and } 7,4,7$$

we actually have a majority element for the second partition, although there is no majority element in A.

Running time: Line 4 and 11 accounts for the time of solving two problems of $n/2$. All other lines accounts for a linear time of input size n. Thus the recurrence equation for the time $T(n)$ can be written as

$$T(n) \leq 2T(n/2) + an$$

Using the Master Theorem, we can obtain $T(n) = O(n \log n)$.

3. For two strings $X = \langle x_1, \ldots, x_n \rangle$ and $S = \langle s_1, \ldots, s_m \rangle$, if for some $1 \leq i_1 < i_2 < \ldots < i_n \leq m$,

$$s_{i_1} s_{i_2} \ldots s_{i_n} = x_1 \ldots x_n$$

then S is a *super-sequence* of X, and X is a *subsequence* of S. For example, *abecbdaed* is a super-sequence of *abcbae*, but *baecbdaed* is not.

(10 points) (a) Please determine a shortest common super-sequence of

acbd and *dccab*

Solution:

```
a     c   b d
      |   |
  d c c a b
-------------
a d c c a b d
```

(35 points) (b) If you are given two strings X and Y, of length m and n respectively, can you devise a dynamic programming algorithm to find the shortest common super-sequence for X and Y. Please

(1) define the recurrences

(2) provide the dynamic programming algorithm

(3) provide the backtrack algorithm to print out the shortest common super-sequence in the right order.

Solution: The recurrence is defined as follows

$$L[i,j] = \begin{cases} 0 & i = 0, j = 0 \\ L[0, j-1] + 1 & i = 0, j > 0 \\ L[i-1, 0] + 1 & i > 0, j = 0 \\ L[i-1, j-1] + 1 & x_i = y_j;\ i, j > 0 \\ \min\{L[i-1, j], L[i, j-1]\} + 1 & x_i \neq y_j;\ i, j > 0 \end{cases}$$

The auxiliary data structure to keep track of the optimal solutions:

$$P[i,j] = \begin{cases} 00 & i = 0, j = 0 \\ 01 & i = 0, j > 0 \\ 10 & i > 0, j = 0 \\ 11 & x_i = y_j;\ i, j > 0 \\ 10 & L[i-1, j] < L[i, j-1];\ x_i \neq y_j;\ i, j > 0 \\ 01 & L[i-1, j] \geq L[i, j-1];\ x_i \neq y_j;\ i, j > 0 \end{cases}$$

```
SHORTEST-COMMON-SUPERSEQUENCE(X)
 1  for i ← 0 to m
 2       do for j ← 0 to n
 3                do
 4                   if i = 0, j = 0
 5                     then L[i, j] ← 0
 6                          P[i, j] ← 00
 7                   elseif i = 0, j > 0
 8                     then L[i, j] ← L[i, j − 1] + 1
 9                          P[i, j] ← 01
10                   elseif i > 0, j = 0
11                     then L[i, j] ← L[i − 1, j] + 1
12                          P[i, j] ← 10
13                   elseif i > 0, j > 0, x_i = y_j
14                     then L[i, j] ← L[i − 1, j − 1] + 1
15                          P[i, j] ← 11
16                   elseif i > 0, j > 0, x_i ≠ y_j, L[i − 1, j] < L[i, j − 1]
17                     then L[i, j] ← L[i − 1, j] + 1
18                          P[i, j] ← 10
19                   elseif i > 0, j > 0, x_i ≠ y_j, L[i − 1, j] ≥ L[i, j − 1]
20                     then L[i, j] ← L[i, j − 1] + 1
21                          P[i, j] ← 01
22  return L, P

BACKTRACK(P, i, j)
 1  if P[i, j] = 00
 2     then return
 3  elseif P[i, j] = 01
 4     then BACKTRACK(P, i, j − 1)
 5          Print y_j
 6  elseif P[i, j] = 10
 7     then BACKTRACK(P, i − 1, j)
 8          Print x_i
 9  elseif P[i, j] = 11
10     then BACKTRACK(P, i − 1, j − 1)
11          Print x_i or y_j
```