Engi 6892 Algorithms: Complexity and Correctness

T. S. Norvell (c) 2011

2011 Dec 10

Instructions: Answer all questions. If possible, write your answers in the space provided. Request a yellow booklet if more space is required. *Answers in yellow booklets will **not** be marked unless there is a clear indication in the space provided on the exam paper that the answer is in the yellow booklet and your name is on the yellow booklet. The same goes for answers on the back-side of any pages.* This is an **closed book** exam. Textbooks, notes, and electronic devices are not permitted; no, you can't use your cell phone as a clock. However, paper inter-language dictionaries are permitted. *Set your cell phone to **silent** and put it away.*
Formula Sheet: You may use a single 8.5" by 11" sheet of paper.
Total points: 90 + bonus of 15

Name:

Student #:

Folders/Documents/courses/alg/2011/Exams-and-quizzes/Ada.jpg

Library of Congress

Student #: _____

Q0 [15] Suppose we have two arrays of strings a and b, each of length n; array a is sorted in ascending order array b is in descending order. We want to determine, in $O(\log n)$ time, whether there is a point p where a and b have the same value. If there is such a point, then there is a last one, and that point will be such that $a[p] \leq b[p]$ but $a[p+1] > b[p+1]$. To deal smoothly with boundary cases, we will assume that $a[0] \leq b[0]$ and $a[n-1] > b[n-1]$.

precondition: $a[0] \leq b[0] \wedge a[n-1] > b[n-1]$
postcondition: $0 \leq p < n-1 \wedge a[p] \leq b[p] \wedge\ a[p+1] > b[p+1]$

(Of course changing a and b is not allowed.)
(a)[5] What is the invariant?

(b)[3] What is the loop guard

(c)[2] What is the variant?

(d)[5] Write a proof outline for this problem so that it takes $O(\log n)$ time.

Student #: _____

Q1 [15]

Design the syntactic and semantic interface for an abstract data type representing "bags". A bag is similar to a set except that each element may be in the "bag" any number of times, as long as that number is natural (i.e. a nonnegative integer). Methods should include "add(x)", which increases (by 1) the number of times item x is in the bag, "remove(x)", which reduces (by 1) the number of times x is in the bag, provided the item is in the bag a positive number of times, and "contains(x)", which returns the number of times x is in the bag.

Student #: _____

Q2 [15]

(a)[5] Last week I figured out that, if I could find an algorithm for the Frobnitz problem, I could sort n things as follows

```
procedure frobnitzSort( var a : Array[T])
    var b : Array[T] := new Array[T](a.length)
    transform(a, b)
    frobnitz(b)
    untransform(b, a)
end frobnitzSort
```

The transform and untransform procedures are both $\Theta(n)$ time (where n is the length of the array) in the worst case.

What do these facts imply about of the Frobnitz problem? (Include any caveats.)

(b)[5] Yesterday, I discovered a $O(n \log n)$ time (in the worst case) algorithm to solve instances of the Frobnitz problem of size n, using only comparisons and moves. What can we say about this algorithm? What more can we say about the problem? (Include any caveats.)

(c)[5] Usually we are concerned more with worst-case time complexity than with average-case time complexity. Why is that?

Student #: _____

Q3 [15] We need to fly a small plane from St. John's to Acapulco. The plane can safely fly up to 500km on one tank of gasoline. An optimal route will have the fewest stops for refuelling. Suppose as input we have a list of airports in North America and the Caribbean along with their latitude and longitude. Assume that flights can follow great circle routes (geodesics).

(a)[10] Describe an efficient approach to solving this problem. (You do not need to write out the algorithm in detail.)

(b)[5] What is the time complexity of your algorithm? (In stating the complexity, be sure to define any variables you use; e.g., don't use n unless you tell me what n means.)

Student #: _____

Q4 [15] We have a list of n jobs $[a_0, a_1, ..a_n]$ to do. Each job a_i is associated with a profit $a_i.p$ and a deadline of $a_i.d$. For example we might have

Job:	a_0	a_1	a_2	a_3	a_4
$.p$	50	25	15	30	20
$.d$	3	1	2	1	3

Each job takes 1 unit of time. The clock starts at time 0. A set of jobs is a solution if the set can be ordered so that each job ends on or at its deadline. For example $\{a_0, a_1, a_2\}$ is a solution as we can execute a_1 during the time period $[0, 1)$, a_2 during period $[1, 2)$,and a_0 during period $[2, 3)$. However any set of size 4 is not feasible and neither is any superset of $\{a_1, a_3\}$. The optimal solution has the greatest profit; in the example it is $\{a_0, a_3, a_4\}$ with a profit of 100.

(a) [10] Design a greedy algorithm to find an optimal solution. [Hint: consider jobs in order of decreasing profit.] No proof is required.

(b) [5] What is the time complexity of your algorithm?

Student #: _____

Q5 [15] A Stirling number of the first kind $\begin{bmatrix} n \\ k \end{bmatrix}$ is the number of permutations of $\{0, ..n\}$ that have exactly k cycles. (For example $[0, 2, 1, 4, 5, 3]$ has 3 cycles and so contributes 1 to $\begin{bmatrix} 6 \\ 3 \end{bmatrix}$.) This can be calculated by the following set of formulas

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1$$

$$\begin{bmatrix} n \\ 0 \end{bmatrix} = 0, \text{ for } n > 0$$

$$\begin{bmatrix} 0 \\ k \end{bmatrix} = 0, \text{ for } k > 0$$

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}, \text{ for } n > 0 \text{ and } k > 0$$

For example $\begin{bmatrix} 4 \\ 2 \end{bmatrix} = 11$ because $\begin{bmatrix} 3 \\ 2 \end{bmatrix} = 3$ and $\begin{bmatrix} 3 \\ 1 \end{bmatrix} = 2$

Design a bottom-up dynamic-programming algorithm to calculate $\begin{bmatrix} n \\ k \end{bmatrix}$, where $n, k \geq 0$.

Student #: _____

Bonus [15] I recently saw a "jigsaw puzzle" with 1024 interlocking, roughly rectangular pieces (all the same shape), where each piece is coloured a different shade of gray on the front and has a different number on the back. The puzzler can upload a photograph to a website and will receive instructions for putting together the puzzle to resemble the photograph. If the photo is broken into 1024 pixels, then it is fairly obvious that the best solution (in terms of total error) is to pair the lightest pixel with the lightest puzzle piece and so on. A more interesting problem is what happens when the photo is broken into m pixels, with $m < 1024$. Now a subset of the grey pieces must be selected. The problem is this:

- Each instance consists of two sorted (nondecreasing) lists of integers, a and b, with lengths of m and n, respectively, so that $m \leq n$. (a lists the grey levels of the pixels, while b lists the grey levels of the pieces.)
- A solution is a sublist c of b of length m. (A sublist of b is a subset of the items of b in their original order; see examples below.)
- An optimal solution minimizes this cost function:

$$\sum_{k \in \{0,..m\}} (a_k - c_k)^2$$

For example, if the pixels are $a = [4, 5, 6]$ and the pieces are $[1, 3, 5, 7, 9]$, the solutions are the 10 sublists of b that have length 3, namely

$$[1,3,5], [1,3,7], [1,5,7], [3,5,7], [1,3,9], [1,5,9], [3,5,9], [1,7,9], [3,7,9], [5,7,9].$$

The cost of the first solution is $(4-1)^2 + (5-3)^2 + (6-5)^2$. All the costs are shown below:

$[1,3,5]$	$[1,3,7]$	$[1,5,7]$	$[3,5,7]$	$[1,3,9]$	$[1,5,9]$	$[3,5,9]$	$[1,7,9]$	$[3,7,9]$	$[5,7,9]$
14	14	10	2	22	18	10	22	14	14

(a)[5] Design a recursive procedure to find the cost of a minimum cost solution for subinstances that use only the first i items of list a and only the first j items from list b. (You may assume, as a precondition, $0 \leq i \leq j \leq n$ and $i \leq m \leq n$).

(b)[5] Design an efficient bottom-up dynamic-programming algorithm to compute a cost table.

(c)[5] Design a procedure to print an optimal solution from the table.

(Out of room? Ask for a yellow book.)

Have a happy holiday and an enjoyable work term.