COMP 3711H – Honors Design and Analysis of Algorithms
2014 Fall Semester – Written Assignment # 2
Distributed: October 13, 2014 – Due: October 23, 2014
Revised October 19, 2014

Your solutions should contain (i) your name, (ii) your student ID #, and (iii) your email address

Information:

- Please write clearly and briefly.
- Please follow the guidelines on doing your own work and avoiding plagiarism given on the class home page. Don't forget to *acknowledge* individuals who assisted you, or sources where you found solutions.
- Please make a *copy* of your assignment before submitting it. If we can't find your answers, we will ask you to resubmit the copy.
- In what follows the phrase *Give an* $O(f(n))$ *algorithm* means to describe the algorithm, prove its correctness and correctly analyze its running time to show that it's $O(f(n))$.
- Each problem is worth 20 points.
- Your solution should be submitted as a PDF. This can be generated by Latex, from Word or a scan of a (legible) handwritten solution, etc..
- Your solution should be submitted via the CASS system by 11:59PM on October 23, 2014. The class web page has reminders on how to use CASS.
- Revised October 19, 2014: Extra boundary conditions (for 1×2 and 2×1 matrices) added to code for problem 5.

Problem 1: A Greedy Selection Algorithm

Modified from a CLRS problem

You are driving from Boston to Miami along Highway 1. When you start, your gas (petrol) tank is full. When full, the tank holds enough gas to travel M miles. There are n gas stations on your route and your map gives the distances between them. Find an $O(n)$ algorithm for finding the route that will require you to stop for gas the fewest times.

Hint:
(a) Let s be the furthest station on your route that can be reached without having to ever stop for gas. Prove that there is some optimal solution to the problem that has station s as the first stop. Bootstrap off of this to design a greedy algorithm and prove that it yields the correct solution.
(b) Most correctness proofs for this problem require induction. If you do not remember induction, come and talk with the instructor.

Problem 2: Selection on Two Lists

(a) Suppose that X and Y are two sorted sequences, containing m and n elements respectively. Design an efficient algorithm to find the kth smallest element in the set of $m + n$ combined elements. Note that the best algorithm runs in time $O(\log(\max(m, n)))$. (Hint: Use binary search).

(b) Now suppose that you are given two unsorted arrays A and B, again containing m and n elements respectively. You are also given a black box $O(n)$ selection algorithm SEL that can be called on A and B separately and, in addition, a small number (say, ten) of extra memory words that can be used to store some extra data. Using SEL as a subroutine, design an $O(n+m)$ algorithm for finding the kth smallest element in the combined set of $O(m + n)$ elements.
You may not physically combine the two arrays in another array and call an algorithm (either SEL or something you wrote) on this third array. This is because there is no extra memory available to build this third array.

Problem 3: Optimal Compatible Job Scheduling

The *one-processor scheduling problem* is defined as follows:
The input is a set of n jobs, $J_1, J_2, J_3, \ldots$.
Each job has a start time s_i and a finish time f_i with $s_i < f_i$..
When a job is run, it takes over the processor in the interval (s_i, f_i) and no other job can be run on the processor during that period.

A subset S of the jobs is *compatible* if no two of them overlap, i.e., they can all be run on the processor without interfering with each other. Compatible subsets of maximal size are *optimal*

A greedy algorithm works by
1) *First ordering the jobs according to some ordering rule and setting* $S = \emptyset$.
2) Running through all the jobs in the given order
    If current job J_i is compatible with S, i.e., $S \cup \{J_i\}$ is compatible,
        Add J_i to S by setting $S = S \cup \{J_i\}$

Note that this algorithm depends strongly upon the order rule used in (1). For each of the following ordering rules, either prove that the resulting greedy algorithm constructs an optimal subset or give a counterexample for which it doesn't (a counterexample would be a specific set of jobs and a demonstration that the constructed subset is not optimal)

(a) Jobs are ordered by nondecreasing start-time s_i

(b) Jobs are ordered by non-decreasing finish-time f_i

(c) Jobs are ordered by non-decreasing size $(f_i - s_i)$.

Problem 4: Huffman Coding

- What is a Huffman code for the set of weights $1, 2, 3, 4, 5, 6, 7, 8$?

- What is a Huffman code for the set of frequencies $1, 2, 4, 8, 16, 32$?

- Consider the family of frequency sets $F_n = \{1, 2, 4, 8, \ldots, 2^n\}$.
  Give a general structure for a Huffman code for the frequency set F_i. Prove that this structure is correct and give the cost of this Huffman code as a function of n (the cost of the code is the weighted external path length of the associated tree).

Problem 5: Monotone Matrices

An $n \times n$ matrix is *monotone* if each of its rows and columns are sorted in nondecreasing order. For example,

$$\begin{pmatrix} 1 & 12 & 17 & 19 \\ 15 & 18 & 20 & 25 \\ 16 & 23 & 26 & 30 \\ 24 & 27 & 31 & 35 \end{pmatrix}$$

is monotone. Given monotone matrix A and value a, the *membership problem* is to test whether $a \in A$ or $a \notin A$. More specifically $MEM(a, x_1, y_1, x_2, y_2)$ will return `TRUE` if there exist i, j such that $a == A[i, j]$ where $x_1 \leq i \leq x_2$ and $y_1 \leq j \leq y_2$ and `FALSE` otherwise. (Note that we are labelling the rows so that the top row is 1 and the bottom one N, while the left column is 1 and the right column is N.) For example, in the given matrix, $MEM(31, 1, 1, 4, 4)$ == `TRUE` while $MEM(31, 1, 1, 3, 3)$ == `FALSE`. Consider the following algorithm:

MEM$(\mathbf{a}, \mathbf{x_1}, \mathbf{y_1}, \mathbf{x_2}, \mathbf{y_2})$:

```
If ((x1 > x2) OR (y1 > y2))                          % Check Validity of indices
      Return(FALSE)

Else if ((x1 == x2) AND (y1 == y2))                  % Check if only one element
      {   If (a == A[x1, y1]) Return(TRUE)
          Else Return(FALSE)
      }

Else if ((x1 + 1 == x2) AND (y1 == y2))              % Check if 2X1 submatrix
      {   If ((a == A[x1, y1]) OR (a == A[x2, y1]))  % Added Oct 19, 2014
              Return(TRUE)
          Else Return(FALSE)
      }

Else if ((x1 == x2) AND (y1 + 1 == y2))              % Check if 1X2 submatrix
      {   If ((a == A[x1, y1]) OR (a == A[x1, y2]))  % Added Oct 19, 2014
              Return(TRUE)
          Else Return(FALSE)
      }

Else if ((x1 + 1 == x2) AND (y1 + 1 == y2))          % Check if 2X2 submatrix
      {   If ((a == A[x1, y1]) OR (a == A[x1, y2]) OR (a == A[x2, y1]) OR (a == A[x2, y2]))
              Return(TRUE)
          Else
              Return(FALSE)
      }

Else  {   u = ⌊(x1+x2)/2⌋ ; v = ⌊(y1+y2)/2⌋ ;       % Recurse on smaller matrices
      If (Mem(a, u, y1, x2, v) == TRUE)              % upper-right submatrix
          Return(TRUE)
      If (Mem(a, x1, v, u, y2) == TRUE)}             % lower-left submatrix
          Return(TRUE)
      If (a ≤ A[u, v])                               % upper-left submatrix
          If  (Mem(a, x1, y1, u, v) == TRUE)
              Return(TRUE)
      If (a > A[u, v])                               % lower-right submatrix
          If  (Mem(a, u, v, x2, y2) == TRUE)
              Return(TRUE)
      Return(FALSE)
      }
```

(a) Show that this algorithm (i) terminates and (ii) correctly solves the membership problem for monotone matrices.

(b) Analyze the running time of the algorithm when run on an $N \times N$ monotone matrix, i.e., when $MEM(a, 1, 1, N, N)$ is called. You may assume that N is always of the form $N = 2^k + 1$, k an integer, if that makes the analysis easier. The running time should measure the worst case number of comparisons made and be given using $O()$ notation. Your running time should be as precise as possible, e..g, if the running time is actually $\Theta(n^2)$ then $O(n^2 \log n)$ would not be tight enough but $O(n^2)$ would be.