Homework 9

Discrete Structures
CS 173 [B] : Fall 2012

Released: Sat Nov 17
Due: Wed Nov 28, 3:00 PM

Attention: Part of Homework 9 is on Moodle. The rest of the homework problems are given below. They need to be submitted in hardcopy.

In problems involving pseudocode, the intention is that you be able to analyze the code entirely by hand. If you want, you may verify your analysis by coding up the algorithms in a computer language of your preference; but this may take longer (and is prone to having undetected bugs if you do not know what the expected behavior of the the algorithm is) than working it out by hand for a few examples.

Pseudocode conventions: Blocks are indicated by indentation (without any {s and }s). := indicates the assignment operator and = denotes the equality predicate. Elements in an array or list are indicated using subscripts. For instance, if A is an array or a list, A_i denotes the ith element in this list (which is often denoted as $A[i]$ in programming languages, when A is an array). All standard arithmetic operations are considered to take constant time (though in reality, this is true only as long as the numbers are small enough to fit into the registers of your machine).

---

1. **Algorithm analysis** [15 points]

   Consider the following mystery function, which takes as input a list of real numbers and outputs a non-negative real number.

```
1: function Foo(A_1, A_2, ... A_n: real numbers)
2:     d := |A_1 − A_2|
3:     for i := 1 to n do
4:         for j := 1 to n do
5:             q := |A_i − A_j|
6:             if i ≠ j and q < d then
7:                 d := q
8:     return d
```

   (a) Give a brief English description of what the function foo computes.

   (b) How many times are lines 5 and 6 executed, as a function of n?

   (c) Express the running time of the algorithm expressed as $\Theta(f(n))$ for a simple function f? Briefly justify your answer.

   (d) This is a really bad algorithm for this task. Write pseudocode for a function with a better big-O running time.

   [Hint: First sort the input.]

   (e) Express the running time of your new algorithm from part d as a $\Theta(\cdot)$ expression. (You can use the fact that sorting can be done in $\Theta(n \log n)$ time. How much additional time do you take? What is the overall time?)

2. **Algorithm analysis** [10 points]

Consider the following algorithm which is used to define a sequence $S_{r,q}$ for positive integers r and q. The algorithm below takes r, q and n as inputs and outputs the nth term of the sequence $S_{r,q}$.

```
function SEQUENCE(r, q, n : positive integers)
    if n = 1 then
        return r
    else if n is odd then
        return SEQUENCE(r, q, n − 1) + (n − 1)q
    else
        return SEQUENCE(r, q, n − 1) − (n − 1)q
```

(a) (5 points) Give the first 7 values of the sequence $S_{r,q}$ (for any r, q) and a general formula for the nth value. (Note that this formula can be used to implement SEQUENCE(r, q, n) in constant time.)

(b) (3 points) Suppose that $T(n)$ returns the running time of SEQUENCE (for some fixed r, q). Give a recursive definition of $T(n)$.

(c) (2 points) Express the running time of SEQUENCETERM as $\Theta(f(n))$ for a simple function f. Justify your answer.

3. **Algorithm analysis** [15 points]

Here is the pseudocode for a function named CS. It takes an array of n integers as input, where each integer is from the set $\{1, \ldots, k\}$.

```
1:  function CS(k: integer;  A: array of n integers–each integer being in the range 1 to k)
2:      C := empty array with k entries
3:      for i := 1 to k do                          ▷ Initialize C to all 0s
4:          C_i := 0
5:      for j := 1 to n do                          ▷ A sweep over A, while updating C
6:          C_{A_j} := C_{A_j} + 1
7:      for i := 2 to k do                          ▷ A sweep over C, while updating C
8:          C_i := C_i + C_{i−1}
9:      B := empty array with n entries             ▷ For storing the output
10:     for j := n down to 1 do                     ▷ A reverse sweep over A, while updating B and C
11:         B_{C_{A_j}} := A_j                      ▷ Store the current element of A in a certain location in B
12:         C_{A_j} = C_{A_j} − 1
13:     return B
```

(a) (5 points) Execute the algorithm CS$(6, A)$ where $A = (2, 3, 5, 2, 1, 3, 2)$. Your answer must include:

i. the contents of C right before line 7 is executed,

ii. the contents of C right before line 9 is executed, and

iii. the final output.

(b) (5 points) Give a brief description in English of:

[Your answer to this part should be in terms of an arbitrary input A, not the specific one in (a).]

i. what is stored in C right before line 7 is executed.

ii. what is stored in C right before line 9 is executed.

iii. (Extra 5 points) what the function CS computes as output.

(c) (5 points) If you treat k as a constant, then express the running time of CS, as $\Theta(f(n))$ for a simple function f. Justify your answer.

4. **Algorithm analysis** [15 points]

Here is the pseudocode for a function named M3. It takes as input a list of integers A. The list is expected to have 3 or more elements. What's happening in line 7 is that M3 is calling itself on a list derived from A by (temporarily) removing the kth element (denoted by A_k) from it. You should assume that it takes only constant time to remove A_k temporarily from the list. (In practice this can be achieved by using a linked-list; for the sake of simplicity, extra details required for this are omitted from the pseudocode below.)

```
1:  function M3(A: list of n positive integers with n ≥ 3)
2:      if (n = 3) then
3:          return A_1 + A_2 + A_3
4:      else
5:          bestval := 0
6:          for k := 1 to n do
7:              newval := M3(A \A_k)
8:              if (newval > bestval) then
9:                  bestval := newval
10:     return bestval
```

(a) (4 points) Describe in English what M3 computes.

(b) (4 points) Suppose that $T(n)$ is the running time of M3 on an input list of length n. Give a recursive definition for $T(n)$.

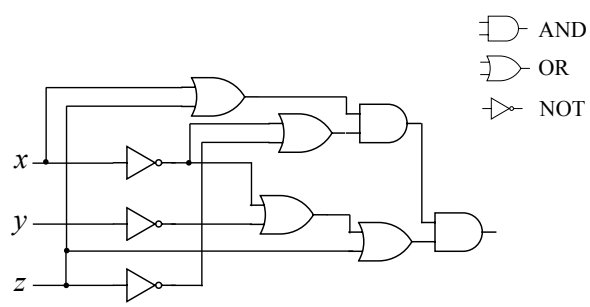
(c) (4 points) How many leaf nodes are there in the recursion tree for $T(n)$? Briefly explain or show work.

(d) (3 points) Does M3 run in $O(2^n)$ time? Briefly explain why or why not.

(e) (Extra Credit: 5 points) How fast can an algorithm be if it computes the same function as M3 computes?

5. **Satisfying a Boolean Circuit.** [15 points]

Below is a boolean circuit with three input bits (marked as x, y, z) and a single output bit. Find all inputs that "satisfy" the circuit: i.e., all triples (x, y, z) such that when its inputs are set to (x, y, z), the circuit outputs 1 (i.e., True).



Note that dark dots are used to indicate wires connected to each other ("fan-out"). Otherwise, crossing wires are not connected to each other.

[Hint: You can evaluate the circuit on all 8 inputs if you want to. Alternately you can investigate certain parts of the circuit logically: for example, which inputs (x, z) make the the top AND gate output 1.]

6. **Solving Search Problems using Decision Oracles** [Extra Credit: 15 points]

In this problem, you need to devise a simple algorithm for efficiently solving a sudoku puzzle, but with the help of a "decision" oracle. In a sudoku puzzle you are given an $n \times n$ grid with some cells already filled in with numbers in the range 1 to n. Your goal is to fill in the rest of the cells with numbers in the range 1 to n, so that the grid satisfies some rule (for our purposes, the specific rule doesn't matter), or to discover that this is not possible (which is, in real-life puzzles, promised not to be the case).

You can query the decision oracle with a sudoku puzzle and it will only answer yes/no to indicate if there exists a solution for the sudoku puzzle or not.

Devise a strategy to solve any sudoku puzzle by making $O(n^3)$ queries to the decision oracle.

[Hint: Note that the oracle works no matter how many cells are already filled in. You can query it with partially filled grids to check if you are on the right track. Your algorithm will not need to rely on the exact rules of sudoku.]

[Note: This is an instance of solving a search problem ("find a solution") by making use of a sub-routine that solves only the decision problem ("does there exist a solution"). When this is possible we say that the search problem can be *reduced* to the decision problem. In that case, the search problem "is not much more complex than" the decision problem, or alternately, the decision problem is "almost as complex as" the search problem.]