

# Masters Exam

Spring 2005  
Department of Computer Science  
University of Arizona

This exam consists of ten problems, from which you will do **six questions**. The questions are in three areas:

- Theory (CSc 520, 545, 573),
- Systems (CSc 552, 553, 576), and
- Applications (CSc 522, 525, 533, 560).

Answer **two questions from each area**. If more than two questions are attempted in an area, the two highest scores are used.

You have two hours for the exam. Books or notes are not permitted. At the end of the exam, submit only your solutions in the envelope provided; do not include scrap paper.

Write your answers on the paper provided separately. Start the answer to each question on a new sheet of paper. On each page you hand in, write the **problem number** in the upper-left corner, and the **digits** of your CS ID number in the upper-right corner. Results will be posted using the CS ID numbers.

Some problems may ask you to write an algorithm or a program. Unless directed otherwise, use informal pseudocode. You may use programming constructs that help present your solution provided their meaning is clear and they do not make the problem trivial. For example, when describing code that iterates over the elements of a set  $S$ , you may use a construct such as

for  $x \in S$  do statement

## Solutions

## Theory 1 CSc 520 (Programming Language Theory)

Consider the following program:

```
program M;
  var m,n : integer;
  var a : array [1..2] of integer;

  procedure P(x,y)
    var m : integer;
  begin
    m := 1; n:= 2;
    a[m] := 4;
    x := x + 2;
    y := y + 5;
  end;

begin
  a[1] := a[2] := m := 2;
  n := 1;
  P(a[m],a[n]);
  print(a[1],a[2]);
end
```

What values will be printed by this program using

- (a) (3 points) call by *value*,
- (b) (4 points) call by *name*, and
- (c) (3 points) call by *reference*?

Briefly explain your answers.

### Solution

- (a) 4 2
- (b) 4 9
- (c) 9 4

## Theory 2 CSc 545 (Algorithms)

The *Discrete Knapsack Problem* is defined as follows. A thief is robbing a store that has  $n$  items. The  $i$ th item is worth  $v_i$  dollars and weighs  $w_i$  pounds, where  $v_i$  and  $w_i$  are integers. He wants to take as valuable a load as possible, but he can carry at most  $W$  pounds in his knapsack, for some integer  $W$ . Which items should he take?

In the *Continuous Knapsack Problem*, the setup is the same, but the thief can take fractional amounts of items, rather than having to take all or nothing of an item.

- (a) (3 points) Which of the two problems exhibit the *optimal-substructure* property? Briefly argue why or why not.
- (b) (3 points) Which of the two problems exhibit the *greedy-choice* property? Briefly argue why or why not.
- (c) (4 points) Describe a *dynamic programming* solution for the Discrete Knapsack Problem that runs in  $O(nW)$  time.

## Solution

(Solution by Stephen Kobourov.)

- (a) Both problems have the optimal substructure property. For the discrete version, consider the most valuable load that weighs at most  $W$  pounds. If we remove the most valuable item  $j$  from this load, the remaining load must be the most valuable load weighing at most  $W - w_j$  that the thief can take from the  $n - 1$  original items, excluding  $j$ . For the continuous version, consider that if we remove a weight  $w$  of one item  $j$  from the optimal load, the remaining load must be the most valuable load weighing at most  $W - w$  that the thief can take from the  $n - 1$  original items plus  $w_j - w$  pounds of item  $j$ .
- (b) The continuous version exhibits the greedy-choice strategy but the discrete version does not. The greedy algorithm for the continuous version is to sort the items by cost/weight value and take as much as possible starting from the most valuable item, in order. The algorithm runs in  $O(n \lg n)$  time, because of the sorting. To see that the discrete version does not have the greedy choice property, consider an example:
  - Item 1: cost \$60, weight 10, value 6.
  - Item 2: cost \$100, weight 20, value 5.
  - Item 3: cost \$120, weight 30, value 4.
  - Capacity of knapsack:  $W = 50$ .

The greedy approach will take items 1 and 2 for a total value of \$160, whereas the optimal solution is to take items 2 and 3 for a total value of \$220.

- (c) Define

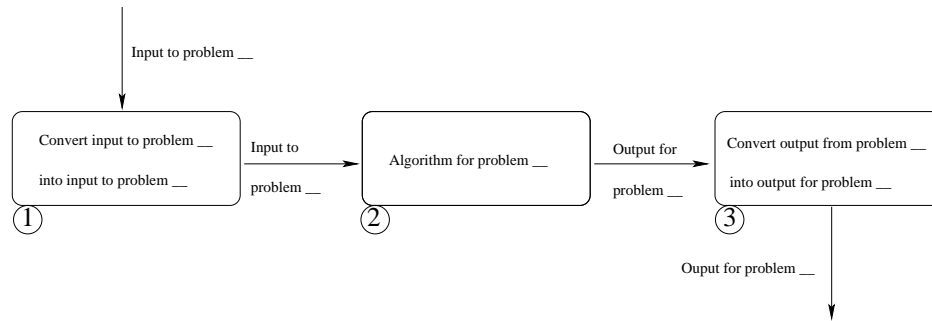
$$c[i, w] = \begin{cases} 0, & \text{if } i = 0 \text{ or } w = 0; \\ c[i - 1, w], & \text{if } w_i > w; \\ \max(v_i + c[i - 1, w - w_i], c[i - 1, w]), & \text{if } i > 0 \text{ and } w \geq w_i; \end{cases}$$

where  $c[i, w]$  is the value of the solution for items  $1, 2, \dots, i$  and maximum weight  $w$ . In other words, if the thief takes item  $i$ , he takes  $v_i$  value and he can choose from items  $1, 2, \dots, i - 1$  up to the weight limit  $w - w_i$ , and get  $c[i - 1, w - w_i]$  additional value. On the other hand, if he decides not to take item  $i$ , he can choose from items  $1, 2, \dots, i - 1$  up to the weight limit  $w$ , and get  $c[i - 1, w]$  value. The better of the two choices should be made, which accounts for the third case above. The table to be filled is of size  $n \times W$  and can be filled in row-major order. The computation takes constant time per table cell, so the total amount of work is  $O(nW)$ . To recover the items that we took to arrive at the solution (in the bottom-rightmost cell) we can trace back from that cell, recovering the choices that were made.

### Theory 3 CSc 573 (Theory of Computation)

Suppose we are given a problem  $A$  that we conjecture is NP-complete. We wish to use another problem  $B$ , which is known to be NP-complete, to prove that  $A$  is NP-complete. This question asks you to illustrate the steps of a standard reduction proof that proves the NP-completeness of  $A$ . In particular,

- (a) (1 point) Should we show that  $A \leq_p B$  or  $B \leq_p A$ ?
- (b) (2 points) Fill in the blanks in the diagram below.



- (c) (1 point) For the correctness of the proof, it is important that computations in two of the three boxes (1), (2), (3) above be polynomial in the input size. Which two?

Now that we are done with the structure of the proof, let's see what we have accomplished. In one sentence each, give the most accurate answers you can to the following questions.

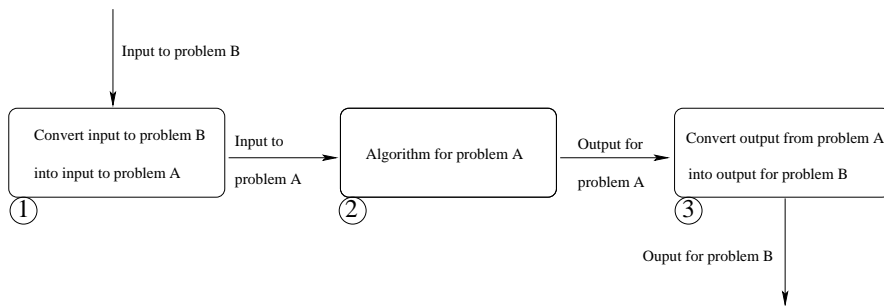
- (d) (1 point) What can you say about the time-complexity of problem  $A$ ?
- (e) (1 point) If someone presents you with a polynomial-time algorithm for problem  $B$  does it contradict the proof outlined above?
- (f) (1 point) What are the greater implications of the existence of a polynomial-time algorithm for problem  $B$ ?
- (g) (3 points) Show that Vertex Cover (deciding the set  $\{\langle G, k \rangle: \text{graph } G \text{ has a vertex cover of size at most } k\}$ ) is NP-complete using the fact that Clique (deciding the set  $\{\langle G, \ell \rangle: \text{graph } G \text{ contains a clique of size at least } \ell\}$ ) is NP-Complete.

## Solution

(Solution by Stephen Kobourov.)

(a)  $B \leq_p A$ .

(b) The blanks should be,



(c) Boxes (1) and (3).

(d) Since  $B$ , a known NP-Complete problem, is polynomially reducible to  $A$ , problem  $A$  is NP-hard.

(e) No, there is no contradiction if  $B$  can be solved in polynomial time.

(f) The greater implication would be that  $P=NP$ .

(g) First show that Vertex-Cover can be verified in polynomial time (which is easy). Then show a reduction from Clique to Vertex-Cover as follows. The reduction uses the complement  $G^*$  of  $G$  which can be computed in polynomial time. Let  $n = |V|$ . If  $G$  has a clique  $V' \subseteq V$  with  $|V'| \geq \ell$  then  $V - V'$  is a vertex cover in  $G^*$  of size at most  $n - \ell$ . Conversely, if  $G^*$  has a vertex cover  $V' \subseteq V$ , where  $|V'| \leq k$ , then  $V - V'$  is a clique in  $G$  of size least  $n - k$ .

## Systems 1 CSc 552 (Operating Systems)

A *log-structured file system* (LFS) batches newly-written data into large *segments*, and writes each segment to a contiguous region of free space on the disk. A *cleaner* garbage-collects the live data in old segments to create free regions for new segments. In contrast, the traditional FFS-based UNIX file layout clusters blocks from the same file and files from the same directory in the same cylinder group.

- (a) (3 points) What is the rationale for LFS, as given by the authors of the paper?
- (b) (2 points) Describe an access pattern for which you would expect LFS to perform better than FFS.
- (c) (2 points) Describe a file access pattern for which you would expect FFS to perform better than LFS.
- (d) (3 points) The original LFS cleaner used a greedy algorithm for choosing segments to clean: the segment with the fewest live data is cleaned first. This resulted in a poor LFS performance. Why?

### Solution

(Solution by John Hartman.)

- (a) LFS is write-optimized, whereas FFS is read-optimized. Hardware trends favor LFS because the performance gap between memory and disk is diverging, and memory is getting larger relative to disk (note that this last point proved not to be true). Large memories improve cache hit rates for reads, but don't improve write performance because data must eventually be written to disk for reliability. LFS uses the disk efficiently by performing large writes. It can also defer reorganization (cleaning) until idle periods. FFS, on the other hand, is optimized for reads and may incur many seeks to organize data on disk when it is written.
- (b) There are several possibilities:
  - Write a file's blocks randomly, then read them in the same order. LFS will perform sequential (efficient) I/Os for both the reads and writes, FFS will perform sequential I/Os for the reads, but random for the writes.
  - Write data to many files in different directories. The I/Os will be sequential under LFS, but FFS will potentially require seeks to different cylinder groups.
  - Write the data randomly many times, but read it once (in any order). LFS will perform sequential I/Os for the writes, FFS will perform random I/Os.
- (c) Write a file randomly, then read it sequentially many times. LFS will perform random I/Os for the reads, whereas FFS will perform sequential I/Os.
- (d) Data written to a file system tend to be deleted or overwritten quickly. If a segment contains very few live data, it is probably a newly-created "hot" segment that will be short-lived. Cleaning a hot segment is of little benefit, because its data will die soon and

the segment will essentially clean itself. On the other hand, data in a cold segment aren't likely to die in the near future, so the free space in the segment cannot be reclaimed unless the segment is cleaned. Therefore, the cleaning decision should be based on the amount of free space in a segment, weighted by the age of the live data.

## Systems 2 CSc 553 (Compilers)

*Inline expansion* (also known as *inlining* or *procedure integration*) has become a popular optimization technique. The idea is to replace a call to a procedure  $P$  with  $P$ 's body.

- (a) (2 points) Expanding a procedure inline will sometimes result in the program executing faster, sometimes slower. List all the potential sources of slowdowns and speedups.
- (b) (2 points) It is generally a difficult problem to decide which procedures should be expanded inline at which call-sites. However, there are some situations where it is *always* a win to perform an expansion. Name one such situation.
- (c) (3 points) Assume an imperative language such as C++ or Pascal, with nested procedures, call-by-value and call-by-reference (**var**) parameters (or the logical equivalent of reference parameters, e.g., **&x** in C++), integer and array data types, and the normal set of control structures. Give a pseudo-code algorithm for an *inline expansion optimization pass*. The algorithm should preserve the semantics of the original program and avoid redundant expansions.
- (d) (3 points) Assume an object-oriented, Java-like language. How would your algorithm for Part (c) have to be extended to be able to expand a method call `m.P()` inline? What other optimization techniques would be helpful in improving the quality of the resulting code?

## Solution

(Solution by Saumya Debray.)

- (a) Reasons for speedups:
  - The overhead of call, return, register save/restore, parameter passing, is removed.
  - Inlining opens up a larger scope (perhaps consisting of code from several procedures) for the optimizer, which may allow it to do a better job.

Reasons for slowdowns:

- Larger procedure bodies will have a higher register pressure. Hence more spill code may have to be inserted.
  - A larger executable can have negative effects on the paging and icache-behavior of the program.
- (b) Situations when it is always a win to inline a procedure are: (1) when the body of the expanded procedure is smaller than the code to call it, and (2) when there is exactly one call site for the called procedure.
  - (c) An algorithm is:
    - (1) Build the call graph for the program.

- (2) Using some heuristic (based on the size of the caller and callee, profiling information, etc.) decide which procedures to expand at which call-sites.
- (3) **FOR** each call  $P(a_1, \dots, a_n)$  in  $Q$  to inline procedure  $P(f_1, \dots, f_n)$ , in reverse topological order of the call graph **DO**
- (i) Replace the call  $P(a_1, \dots, a_n)$  with  $P$ 's body.
  - (ii) Replace references to call-by-reference formal  $f_k$  with a reference to the corresponding actual parameter  $a_k$ .
  - (iii) For each call-by-value formal parameter  $f_k$  create a new local  $c_k$ . Insert code to copy the call-by-value actual  $a_k$  into  $c_k$ . Replace references to the call-by-value formal  $f_k$  with a reference to its copy  $c_k$ .
  - (iv) For each of  $P$ 's local variables  $l_k$  create a new local  $v_k$  in  $Q$ . Replace references to local variable  $l_k$  with a reference to  $v_k$ .

(d) The call  $m.P()$  would be expanded as follows:

```

if (m instanceof class1)
    code for class1::P
else if (m instanceof class2)
    code for class2::P
else ...

```

with one test for each possible run-time type of  $m$ . This can be improved on by using an *inheritance hierarchy analysis* or a dataflow *type propagation* analysis, to cut down on the set of possible types of  $m$ .

## Systems 3 CSc 576 (Architecture)

- (a) (2 points) In presence of indirect branches the *Branch Target Buffer* (BTB) does not perform very well. Explain the reasons for this behavior.
- (b) (4 points) For a small INSTRUCTION cache, a cache using direct mapping can often provide lower miss rates than one using full associativity with LRU replacement. Explain why this is possible using an example.
- (c) (4 points) Provide a sequence of instructions that implement an *acquire lock* operation using *load link (ll)* and *store conditional (sc)* instructions.

### Solution

(Solution by Rajiv Gupta.)

- (a) An indirect branch typically has many possible targets. Thus each time an indirect branch is executed, the target taken may be different from the one taken during its previous execution. Since the BTB only remembers the branch target taken during the most recent execution of the indirect branch, it often fails to provide the correct target information.
- (b) Let's assume that we have a loop with sequential code requiring  $n + 1$  cache blocks. Further assume that the cache size permits  $n$  blocks to reside in the cache at any given time. In case of fully associative cache each cache access will result in a miss because the block kicked out will be the one to be accessed next as this block is also the least recently used block (e.g., access to block  $n + 1$  will evict block 1 which will be accessed next). In direct mapped cache only the accesses to the single pair of loop blocks that map to the same cache block will cause repeated misses. All other accesses will not result in any misses. Hence the direct mapped cache will perform better than a fully associative cache.
- (c) One possible answer is:

```
        ;lock available: 0(R1) contains 0
        ;lock held: 0(R1) contains 1
li      R4, #1
try:    mov   R3, R4
        ll   R2, 0(R1)
        sc   R3, 0(R1)
        beqz R3, try
        mov  R4, R2
        bnez R4, try
        ;lock acquired
```

Another is:

```
        ;lock available: 0(R1) contains 0
        ;lock held: 0(R1) contains 1
```

```
try: ll    R2, 0(R1)
     bnez  R2, try
     li    R2, #1
     sc    R2, 0(R1)
     beqz  R2, try
     ;lock acquired
```

## Applications 1 CSc 522 (Parallel and Distributed Programming)

- (a) (2 points) Define the term “barrier synchronization point.”
- (b) (4 points) Give the code that two processes,  $P_i$  and  $P_j$ , would execute to form a two-process symmetric barrier. Also declare and initialize shared variables that you use. The processes should use flags or counters and spin waiting (busy waiting).
- (c) (4 points) Show the structure of an 8-process butterfly or dissemination barrier. You do not need to give any code, but show clearly which process interacts with which in each stage of the barrier.

### Solution

- (a) A barrier synchronization point is a point in the code of every process (thread) that all processes must reach before any proceed.

- (b) Shared variables: `arrive[i] = 0`, `arrive[j] = 0`

Code for process  $i$ :

```
await (arrive[i] == 0);    # safety check
arrive[i] = 1;            # announce arrival
await (arrive[j] == 1);   # wait for other process
arrive[j] = 0;           # clear other's flag
```

The first line is needed so the code can be reused at the next barrier. If it is not there, then process  $i$  could set `arrive[i]` to 1 before process  $j$  clears the flag.

An alternate way to code this is to increase integers counters.

Shared variables: `arrive[i] = 0`, `arrive[j] = 0`

Code for process  $i$ :

```
arrive[i]++;
await (arrive[j] >= arrive[i]);
```

The first and fourth lines of the flag-based solution are not needed, because the counters continually increase. Note, however, that `>=` is needed in the `await` statement. Checking only for equality could lead to livelock, because the equality point might be missed.

- (c) You need three stages for an 8-process barrier. The connections in a dissemination barrier are as follows:

```
stage 1: connect 1 to 2, 2 to 3, 3 to 4, ..., 7 to 8, 8 to 1
stage 2: connect 1 to 3, 2 to 4, ..., 6 to 8, 7 to 1, 8 to 2
stage 4: connect 1 to 5, 2 to 6, ..., 4 to 8, 5 to 1, 6 to 2, ...
```

In a butterfly barrier, connections are symmetric, as follows:

```
stage 1: connect 1 to 2, 2 to 1, 3 to 4, 4 to 3, ..., 7 to 8, 8 to 7
stage 2: connect 1 to 3, 3 to 1, 2 to 4, 4 to 2, 5 to 7, ..., 8 to 6
stage 4: connect 1 to 5, 5 to 1, 2 to 6, 6 to 2, ..., 5 to 8, 8 to 5
```

A dissemination barrier is better in general because it works for any number of processes. A butterfly barrier requires a power of 2.

## Applications 2 CSc 525 (Networking)

Let  $A$  and  $B$  be two stations attempting to transmit on a shared Ethernet network. Each has a steady queue of frames ready to send:  $A$ 's frames will be numbered  $A_1, A_2$ , etc., and similarly for  $B$ . Let  $T = 51.2 \mu\text{sec}$  be the exponential backoff base time. Suppose  $A$  and  $B$  simultaneously attempt to send their frame 1.

- (a) (2 points) What are the odds that  $A$  will succeed on the next round? That  $B$  will succeed? That they will collide again?
- (b) (3 points) Suppose  $A$  wins and transmits frame  $A_1$ . When  $A$  attempts to send frame  $A_2$  and  $B$  attempts (again) to send frame  $B_1$ , the frames collide. What are the odds this time for  $A$  winning?  $B$  winning? Another collision?
- (c) (3 points) Suppose  $A$  wins and transmits frame  $A_2$ . Suppose further that  $A$  continues to win the future decisions following collisions. Give a reasonable lower bound for the probability that  $A$  is the winner of these backoff "races." Include an explanation of how close your estimate is, if it is not an exact lower bound.
- (d) (2 points) If  $A$  wins all of these races, what will happen to frame  $B_1$ ? What then happens when  $B$  attempts to send frame  $B_2$ ?

### Solution

(Solution by Patrick Homer.)

- (a) Both  $A$  and  $B$  generate a random choice from  $\{0, 1\}$ .  $A$  has a 25% chance of winning,  $B$  has a 25% chance of winning, and there is a 50% chance of another collision.
- (b)  $A$  generates a random choice from  $\{0, 1\}$  and  $B$  generates a random choice from  $\{0, 1, 2, 3\}$ . There are now eight combinations. The only winning choice for  $B$  is to pick 0 and have  $A$  pick 1. There is a  $1/8$  chance that  $B$  will win. There are two tie choices: both pick 0 and both pick 1. All the other combinations have  $A$  winning; there is a  $5/8$  chance this will happen.
- (c) Two ways to approach this problem: compute the odds of  $A$  winning, or compute the odds of  $B$  losing. The odds that  $A$  wins:

$$\prod_{3 \leq n \leq \text{numpackets}} \left(1 - \frac{3}{2^n}\right).$$

There are also times when  $A$  and  $B$  will tie and try again. Even in these cases,  $A$  still has the greater chance of winning. The formula above is then the smallest chance that  $A$  will win. The odds of  $A$  winning are somewhat larger in practice since  $A$  will win most of the ties.

The other approach is to compute the odds that  $B$  wins outright, then subtract this number from 1 to get the odds of either a tie or  $A$  winning:

$$1 - \prod_{3 \leq n \leq \text{numpackets}} \left(1 - \frac{1}{2^n}\right).$$

The actual odds of  $A$  winning will lie between these two bounds.

- (d) After a specified number of attempts,  $B$  will report an error and stop attempting to transmit frame  $B_1$ . The actual number of tries is not specified in the Ethernet standard. A typical number of attempts is 16; though, the value of  $n$  used in the exponential backoff is capped at a smaller value, typically 10. When  $B$  attempts to send  $B_2$ , a collision can again occur between  $B_2$  and the packet that  $A$  is trying to send. This time,  $B$  starts over and chooses a random choice from  $\{0, 1\}$ , just like  $A$  will be doing.  $B$  now has an even chance of “winning” and being able to transmit.

### Applications 3 CSc 533 (Graphics)

For many of the questions below, you are asked to describe how appearance (RGB) changes, either as a consequence of a change to the world, or as the viewer moves about and/or changes viewing direction. If you think that there is no change, then state that explicitly, and explain why. In the case that the changes are linked to a viewer moving about and changing the viewing direction, you may find it helpful to consider these two actions separately.

- (a) (3 points) Suppose that there is only surface in your “world”, which occupies the entire plane  $z = 0$ . Further suppose that there is only one light source (at infinity) in the direction  $(1, 1, 1)$ . Assume the naive (diagonal) model of illumination, where the effect of a light on a surface can be modeled by multiplying each component of the light color by the reflectance. The color of the light is  $(200, 200, 200)$ , as defined by the color of perfect white diffuse reflector perpendicular to the light direction. The surface has a diffuse (Lambertian) reflectance  $(0.5, 0.3, 0.1)$ . Consider a viewer at  $(10, 10, 10)$  looking in the direction  $(0, 0, -1)$  (down). What is the RGB that a graphics program should create to simulate what the viewer sees? (Show your work).
- (b) (1 point) Describe the change of the appearance (RGB) of that surface as the location of the viewer and their viewing direction changes (assume that  $z$  is always positive).
- (c) (4 points) Now suppose that the surface is a dielectric with some specular reflection added to the diffuse reflection described in Part (a). Now describe the change of the appearance of the surface as the viewer moves about and changes the direction that they are looking (again assume that  $z$  is always positive). In particular, unless you argue that there is no change at all, specify when the surface will look maximally bright. Also describe how the chromaticity changes. (Recall that chromaticity is color without brightness—i.e.,  $(1, 2, 3)$  and  $(10, 20, 30)$  have the same chromaticity, but different color as the second one is 10 times brighter).
- (d) (2 points) Now suppose that a perfectly absorbing (black) diffuse disk (circle) is added to the scene, with radius 3, location  $(10, 10, 10)$ , and surface normal parallel to  $(1, 1, 1)$ . Is there any change to the appearance of the plane? If so, give a rendering technique which will exhibit the change that you mention.

### Solution

(Solution by Kobus Barnard.)

- (a) Lambertian reflection implies an attenuation the light source,  $\hat{\mathbf{s}}$ , falling on a surface with normal  $\hat{\mathbf{n}}$  given by  $\hat{\mathbf{s}} \cdot \hat{\mathbf{n}}$ . In this case,  $\hat{\mathbf{s}} = (1, 1, 1)/\sqrt{3}$  and  $\hat{\mathbf{n}} = (0, 0, 1)$ , so the attenuation factor,  $\hat{\mathbf{s}} \cdot \hat{\mathbf{n}}$  is  $1/\sqrt{3}$ . The naive model gives an absorption attenuation for each channel independently, so we get  $(1/\sqrt{3}) * (200 * 0.5, 200 * 0.3, 200 * 0.1) = (1/\sqrt{3}) * (100, 60, 10)$ .  
*[I suggest a mark for an attempt at doing a dot product between light direction and plane normal, a mark for doing the element-wise color computation, and the third mark for getting the correct answer.]*

(b) There is no change. Because we have an infinite plane and the source is at infinity, changes in viewer position have no effect. More importantly, a property of a Lambertian surface is that it is independent of viewer direction. Alternatively, the process to compute the RGB as done in Part (a) does not use the viewer location or direction, so no matter what, you get the same answer.

(c) Because we have an infinite plane and the source is at infinity, changes in viewer position have no effect. However, there is now a difference due to angular viewing direction because the specularly is like a “fuzzy” mirror reflection, and this is a function of viewing angle. Light coming from a source at  $(1,1,1)$  has a mirror reflection direction of  $(-1,-1,1)$  when hitting a plane with normal  $(0,0,1)$ , and you see it when your viewing direction is the opposite  $(1,1,-1)$ . When you are looking far away from that direction then there is little change in what is seen. As you approach looking back at the mirror direction, more and more of the specular reflection contributes to the color of the surface, reaching a maximum when you are looking back at the mirror direction. This is added to a fixed diffuse component (see Part (b)), so the maximal overall brightness is looking back at the mirror reflection direction. The chromaticity of the specularly is the chromaticity of the light, and so the specular component pushes the chromaticity from the Lambertian reflected one calculated in Part (a) towards that of the light, reaching a maximal effect in the direction of maximal specularly. Since the color of the light is  $(200,200,200)$ , the specularly is whiter than the surrounding surface (which is roughly orange).

*[Two marks for getting the brightness difference and the maximal effect at the mirror direction. There is no need to get the actual directions as specified above, and I suggest ignoring ambiguities regarding which way the light is reflected versus which way you look (the opposite). Two marks for getting the chromaticity correct. One of these for remembering the color of the specularly is the color of the light, and the second for completing the argument to realize that the effect is push the chromaticity towards that of the light in the region of specular reflection.]*

(d) The disk casts a shadow onto the plane, centered at the origin, and those parts will be black. The simplest way to get the shadow is with ray-tracing.

*[One mark for the shadow, and one more for ray tracing. Note that since the disk is black, there is no inter-reflection. Note also that the shadow is not a circle (it is an ellipse), but I would let that slide.]*

## Applications 4 CSc 560 (Databases)

Assume that a join  $R \bowtie S$  is to be performed using the nested loop join algorithm, and that no indexes are available on either relation  $R$  or  $S$ . Let  $B_R$  and  $B_S$  denote the number of blocks of relations  $R$  and  $S$ , and let  $M_R$  and  $M_S$  denote the number of buffers used for accessing relations  $R$  and  $S$  respectively (where  $M_R + M_S = M$ ).

- (a) (5 points) Derive a condition among  $M_R$ ,  $M_S$ ,  $B_R$  and  $B_S$  that allows the join to be computed by a single pass over each of the relations. Explain your answer.
- (b) (5 points) Suppose the above condition is not met. How many buffers should we allocate to the outer relation and how many to the inner one, so that the cost is minimized? Explain your answer.

### Solution

(Solution by Bongki Moon.)

- (a) The condition is

$$M_R + M_S \geq 1 + \min(B_R, B_S).$$

The join can be computed by a single pass if either of the relations can fit entirely in memory. Since at least one buffer page should be used to read in blocks from the other relation, the size of the smaller relation should be no larger than  $M - 1$ .

- (b) The estimated cost of the join is given by

$$B_R + \lceil B_R/M_R \rceil B_S,$$

assuming that relation  $R$  is chosen as an outer relation. The cost will be minimized when  $M_R = M - 1$ . Thus, the outer relation should be given  $M - 1$  buffer pages, and the inner relation one page.