Answer the questions in the spaces provided on the exam. If you run out of room for an answer, continue on the back of the page.
Note that you must justify all answers!

Name: _____

U#: _____

Question	Points	Score
1	25	
2	25	
3	25	
4	25	
Total:	100	

1. You have 75 minutes to complete the exam.
2. I recommend you use pencil on the exam, but blue or black ink are also acceptable.
3. You may NOT use calculators, cell phones or other electronic devices while taking the exam.
4. The exam is closed books and closed notes. If you turn in a completed feedback form at the start of class, you may use 1 page of prepared notes (single-sided).

1. [25 points] Consider the *shortest rook-path* problem. In this problem, we have some set of integer coordinates in 2D plane (at least 2), and we want to find the sequence of *legal moves* that minimizes the distance travelled from the first point (a) to the last (b), where a *legal moves* must be horizontal or vertical (of any length). Prove that the following algorithm does not find the shortest route between a and b. You may draw your counterexample or list the set of coordinates. You will receive partial credit if you can show an example with at least 4 points and describe the solution that the algorithm finds, even if it is not a counterexample.

Input: points: set of at least two 2D coordinates
Input: n: number of coordinates in points
Output: path: sequence of coordinates from points that describe the shortest legal rook-path from a to b

```
1  Algorithm: ShortPath
2  if n < 2 then
3  |  error "Input must have at least 2 points!"
4  for i = 1 to n-1 do
5  |  if i > 1 and points[i] has not been linked yet then
6  |  |  continue;
7  |  if points[n] is horizontal or vertical from points[i] then
8  |  |  Add a link from points[n] to points[i];
9  |  |  soln = (n);
10 |  |  p = n;
11 |  |  while p ≠ 1 do
12 |  |  |  p = point linked to by points[p];
13 |  |  |  Add p to the beginning of soln;
14 |  |  end
15 |  |  return soln;
16 |  else
17 |  |  for j = i+1 to n-1 do
18 |  |  |  if points[j] is unlinked and either horizontal or vertical from points[i] then
19 |  |  |  |  Add a link from points[j] to points[i];
20 |  |  end
21 |  end
22 end
23 error "There is no rook-path from a to b!"
```

Solution: Consider points = {(0,0), (10,0), (10,1), (1,0), (1,1)}. Since (10,0) is horizontal from (0,0), we will add a link from (10,0) to (0,0). In the second iteration, we will add a link from (10,1) to (10,0), and in the third iteration, we will find that data[n]—(1,1)—is horizontal from (10,1), so we will backtrace the links and return the route (0,0)–(10,0)–(10,1)–(1,1) (total length = 20). However, the path (0,0)–(1,0)–(1,1) (total length = 2) is the shortest legal rook-path, so the solution is incorrect in this case.

2. [25 points] Use induction to prove the correctness of the following recursive algorithm to multiply two natural numbers, for all integer constants $c \geq 2$

Algorithm 1: `Multiply`(y, z)

```
  /* Return the product yz */
1 if z == 0 then
2 |  return 0
3 else
4 |  return (Multiply(cy, ⌊z/c⌋) + y · (z mod c))
5 end
```

Solution:

We will prove that this algorithm returns the correct answer by induction on z.

(*Base Case*) If $z = 0$, then lines 1–2 of the algorithm return 0. Also, $yz = y(0) = 0$, so the algorithm returns the correct value when $z = 0$.

(*Induction Step*) Suppose that the algorithm returns the yz for any $z \leq k$. We use the induction hypothesis to show that the algorithm returns the correct value when $z = k + 1$. From our knowledge of basic arithmetic, when we divide $k + 1$ by c, we will get a quotient $q = \left\lfloor \frac{k+1}{c} \right\rfloor$ and a remainder $r = (k + 1) \bmod c$, where $k + 1 = qc + r$. As such, $y(k + 1) = y(qc + r) = cyq + yr = cy\left(\left\lfloor \frac{k+1}{c} \right\rfloor\right) + y((k + 1) \bmod c)$. In particular, since $c \geq 2$, $\left\lfloor \frac{k+1}{c} \right\rfloor < k + 1$, so Multiply$(cy, (\lfloor \frac{k+1}{c} \rfloor))$ should return the correct product by the induction hypothesis. Thus, the return statement in line 4 should return $cy\left(\left\lfloor \frac{k+1}{c} \right\rfloor\right) + y((k + 1) \bmod c) = cyq + yr = y(k + 1)$. Therefore, by induction, Multiply(y, z) returns yz for all $z \geq 0$.

3. [25 points] Analyze the following algorithm to determine asymptotic upper bounds for its worst-case time complexity (i.e., $O(f(n))$). Make your bounds as tight as possible. You may assume that the `floor` function takes $O(1)$ time for any input.
Justify your answer. For full credit you must show all work. Explain (1) the derivation for each line of the algorithm below, (2) how you combine each derivation to represent the asymptotic worst-case time complexity of the algorithm, and (3) how you simplify the resulting expression.

Question 3 continues...

```
void function(int x, int y)
   for (int i = x; i > 0; i = i - 2) {

      output "foobar"

   }

   int j = y

   while (j > 1) {

      output "foobar"

      j = floor(j / 2)

   }

   for (int k = y; k > 0; k--) {

      int m = 1

      while (m < x) {

         output "foobar"

         m = m * 2
      }
   }
}
```

Complexity

Solution:

```
O(x) iterations
O(1)

for loop:  O(x)

O(1)

O(lg(y)) iterations

O(1)

O(1)

while loop:  O(lg(y))

O(y) iterations

O(1)

O(lg(x)) iterations

O(1)

O(1)
inner loop:  O(lg(x))
outer loop:  O(ylg(x))
Overall:  O(x + ylg(x))
```

Solution: The first `for` loop will iterate $\lfloor x/2 \rfloor$ times, which is bounded above by $x/2$ and so $O(x)$ times. The loop body takes $O(1)$ for every iteration, for a total of $O(x)O(1) = O(x)$ time. The second (while) loop iterates until $j \leq 1$. Since j is divided by 2 every iteration of this loop, y would need to double to add one more loop iteration, so this loop makes $\lg y$ iterations. The loop body takes $O(1)$ for every iteration, for a total of $O(\lg y)O(1) = O(\lg y)$ time.

The outer for loop iterates $O(y)$ times. The inner while loop iterates until $m \geq x$, and since m is doubled each iteration, it will take $\lg x$ iterations until $m = x$. The inner loop takes $O(1)$ time each iteration, for a total of $\lg x O(1) = O(\lg x)$ time total. Thus, the outer loop will take $O(y)O(\lg x) = O(y \lg x)$ time total.

The entire program will take $O(x) + O(1) + O(\lg y) + O(y \lg x)$ time. However, since $O(y \lg x)$ dominates $O(\lg y)$ and both dominate $O(1)$ time, this equals $O(x) + O(y \lg x) = O(x + y \lg x)$ time in total.

4. [25 points] Prove that $f_k(n) = O(\lg n)$ for all $k \geq 1$, where $f_k(n) = \lg(n^k)$.

Solution:

Proof. (*Base case*) When $k = 1$, $f_k = \lg n$, which is $O(\lg n)$ by the reflexive property.

(*Inductive step*) Suppose that $f_k = O(\lg n)$, and consider f_{k+1}. $f_{k+1}(n) = \lg(n(n^k))$. By the properties of log, $f(n) = \lg(n) + \lg(n^k) = \lg(n) + f_k(n)$. By the inductive hypothesis and reflexive property of Big-Oh, $f_{k+1}(n) = O(\lg n) + O(\lg n) = O(\lg n)$. Therefore, by induction, $f_k(n) = O(\lg n)$ for all $k \geq 1$. □