COSC 6111 Advanced Design and Analysis of Algorithms
Jeff Edmonds
Assignment: Loop Invariants

First Person:	Second Person:
Family Name:	Family Name:
Given Name:	Given Name:
Student #:	Student #:
Email:	Email:

Do two of questions $\{2, 3, 4, 5, 6\}$ (that you have NOT seen a solution for before) and do 7, 9, and 10.

Problem Name	If Done Old Mark	Check if to be Marked	New Mark
1 Tiling Chess Board			
2 Loop Invariants - Sorted Matrix Search			
3 Loop Invariants - Lake			
4 Recursion Inversions			
5 $d + 1$ Colouring			
6 Loop Invariants - Connected Components			
7 Multiplying			
8 Dividing			
9 Lower Bounds - Sorted Matrix Search			
10 Amortized Analysis			

The questions are designed to test your knowledge of the the LI steps. Be sure to do the key steps.

1. (Removed because too easy.) Tiling Chess Board: You are given a $2n$ by $2n$ chess board. You have many tiles each of which can cover two adjacent squares. Your goal is to place non-overlapping tiles on the board to cover each of the $2n \times 2n$ tiles except for to top-left corner and the bottom-right corner. Prove that this is impossible. To do this give a loop invariant that is general enough to work for any algorithm that places tiles. Hint: chess boards color the squares black and white.

2. Loop Invariants - Sorted Matrix Search: Search a Sorted Matrix Problem: The input consists of a real number x and a matrix $A[1..n, 1..m]$ of nm real numbers such that each row $A[i, 1..m]$ is sorted from left to right and each column $A[1..n, j]$ is sorted from top to bottom. The goal is to determine if the key x appears in the matrix. Design and analyze an iterative algorithm for this problem that examines as few matrix entries as possible. Careful, if you believe that a simple binary search solves the problem. Later we will ask for a lower bound and for a recursive algorithm.

3. Loop Invariants - Lake: You are in a lake of radius one. You can swim at a speed of one and can run infinitely fast. There is a smart monster on the shore who can't go in the water but can run at a speed of four. Your goal is to swim to shore arriving at a spot where the monster is not and then run away. If you swim directly to shore, it will take you 1 time unit. In this time, the monster will run the distance $\Pi < 4$ around to where you land and eat you. Your better strategy is to maintain the most obvious loop invariant while increasing the most obvious measure of progress for as long as possible and then swim for it. Describe how this works.

4. Recursion Inversions: Given a sequence of n distinct numbers $A = \langle a_1, a_2, \ldots, a_m \rangle$, we say that a_i and a_j are inverted if $i < j$ but $a_i > a_j$. The number of inversions in the sequence A, denoted $I(A)$, is the number of pairs a_i and a_j that are inverted. $I(A)$ provides a measure of how close A is to being sorted in increasing order. For example, if A is already sorted in increasing order, then $I(A)$ is 0. At the other extreme, if A is sorted in decreasing order, every pair is inverted, and thus $I(A)$ is $\binom{n}{2}$. Describe a divide and conquer algorithm that finds $I(A)$ in time $\Theta(nlogn)$. You should assume that the sequence A is given to us in an array such that we can test the value of a_i in unit time, for any i. Note that the obvious algorithm is to test all pairs a_i and a_j; that algorithm runs in $\Theta(n^2)$ time. Hint: you should modify merge sort to also compute $I(A)$. The work each stack frame requires an iterative algorithm. Describe this iterative algorithm using loop invariants.

5. $d + 1$ Colouring: Exercise 1.6.2: Given an undirected graph G such that each node has at most d neighbors, colour each node with one of $d+1$ colours so that for each edge the two nodes have different colours. Hint: Don't think too hard. Just colour the nodes. What loop invariant do you need.

6. Loop Invariants - Connected Components: The input is a matrix I of pixels. Each pixel is either black or white. A pixel is considered to be connected to the four pixels left, right, up, and down from it, but not diagonal to it. The algorithm must allocated a unique name to each connected black component. (The name could simply be 1,2,3,..., to the number of components.) The output consists of another matrix $Names$ where $Names(x, y) = 0$ if the pixel $I(x, y)$ is white and $Names(x, y) = i$ if the pixel $I(x, y)$ is a part of the component named i. The algorithm reads the matrix from a tape row by row as follows.

```
loop $y = 1 \ldots h$
    loop $x = 1 \ldots w$
        ⟨loop−invariant⟩: ??
        if( $I(x, y)$=white )
            $Names(x, y) = 0$
        else
```

```
                ???
            end if
        end loop
    end loop
end algorithm
```

The image may contain spirals and funny shapes. Connected components may contain holes that contain other connected components. A particularly interesting case is the image of a comb consisting of many teeth held together at the bottom by a handle. Scanning the image row by row, one would would first see each tooth as a separate component. As the handle is read, these teeth would merge into one.

(a) Give the classic *more of the input loop invariant* algorithm. Don't worry about its running time.

(b) This version of the question is easier in that the matrix $Names$ need not be produced. The output is simply the number of connected black components in the image. However, this version of the question is harder in your computation is limited in the amount of memory it can use. For example, you don't remember pixels that you have read if you do not store them in this limited memory and you don't have nearly enough memory to store them all. The number of components may be Θ(the pixels) so you cant store all of them either. How little memory can you get away with?

(c) In this this final version, in addition to a small amount of fast memory, you have a small number of tapes for storing data. Data access on tapes, however, is quite limited. A tape is in one of three modes and cannot switch modes mid operation. In the first mode, the data on a tape is read forwards one data item at a time in order. The second mode, is the same except it is read backwards. In the third mode, the tape is written to. However, the command is simply $write(data)$ which appends this data to the end of the tape. Data on a tape cannot be changed. All you can do is to erase the entire tape and start writing again from the beginning. An algorithm must consist of a number of passes. The first pass reads the input from the input tape one pixel at a time row by row in order. As it goes, the algorithm updates what is store in fast memory and outputs data in order onto a small number output tapes. Successive passes can read what was written during a previous pass and/or the input again. The last pass must write the required output $Names$ onto a tape. You want to use as little fast memory and as few passes as possible. For each pass clearly state the loop invariant.

7. Multiplying and Dividing using Adding: My son and I wrote a JAVA compiler for his grade 10 project. We needed the following.

Suppose you want to multiply $X \times Y$ two n bit positive integers X and Y. The challenge is that the only operations you have are adding, subtracting, if, and while. The kindergarten algorithm, $4 \times 3 = 3 + 3 + 3 + 3$ takes 2^n time. We want to be able to do it in $\mathcal{O}(n)$ add operations. Note that it takes $\mathcal{O}(n)$ bit operations to add and hence this algorithm will require $\mathcal{O}(n^2)$ bit operations.

The high school algorithm multiplies each bit of X with each bit of Y shifts appropriately and adds. This takes $\mathcal{O}(n^2)$ bit operations as required. The reason we cannot implement this algorithm is that we do not know how to access the bits of X and Y.

To help, I will give you the loop invariants, the measure of progress, and the exit condition.

We have values i, x, a, $u[]$, and $v[]$ such that

LI0': $u[j] = 2^j$, (for all j until $u[j] > X$)

LI0": $v[j] = u[j] \times Y$. Note $v[j] - u[j] \times Y = 0$

LI0: Y does not change.

LI1: $X \times Y = x \times Y + a$

LI2: $x \geq 0$

LI3: $x < 2^i = u[i]$

Measure of progress: i decreases by 1 each iteration.

Exit Condition: $i = 0$

To help, I will also tell you how to establish loop invariants LI0' and LI0".

```
$u[0] = 1$
$v[0] = Y$
$j = 0$
while( $u[j] \leq X$ )
        $u[j+1] = u[j] + u[j]$
        $v[j+1] = v[j] + v[j]$
        $j = j + 1$
end while
```

Use the steps laid out in class to complete the description of the algorithm.

8. Suppose you want to integer divide X/Y two n bit positive integers X and Y, i.e. find values a and r so that $X = a * Y + r$ and $r \in [0, Y)$. The challenge is that the only operations you have are adding, subtracting, if, and while. The kindergarten algorithm, finds 13/3 by continuing to subtract 3 from 13 until it becomes less than 3. This, however, takes 2^n time. We want to be able to do it in $\mathcal{O}(n)$ add operations. Note that it takes $\mathcal{O}(n)$ bit operations to add and hence this algorithm will require $\mathcal{O}(n^2)$ bit operations.

   The high school algorithm shifts Y until it is one step from being bigger than X and then subtracts. This is repeated. This takes $\mathcal{O}(n^2)$ bit operations as required. The reason we cannot implement this algorithm is that we do not know how to access the bits of X and Y.

   To help, I will give you the loop invariants, the measure of progress, and the exit condition.

   We have values i, x, a, $u[]$, and $v[]$ such that

   LI0': $u[j] = 2^j$, (for all j until $v[j] > X$)

   LI0": $v[j] = u[j] \times Y$. Note $v[j] - u[j] \times Y = 0$

   LI0: Y does not change.

   LI1: $X = a \times Y + x$

   LI2: $x \geq 0$

   LI3: $x < (2^i) \times Y = v[i]$

   Measure of progress: i decreases by 1 each iteration.

   Exit Condition: $i = 0$

   To help, I will also tell you how to establish loop invariants LI0' and LI0".

```
$u[0] = 1$
$v[0] = Y$
$j = 0$
while( $v[j] \leq X$ )
        $u[j+1] = u[j] + u[j]$
        $v[j+1] = v[j] + v[j]$
        $j = j + 1$
end while
```

   Use the steps laid out in class to complete the description of the algorithm.

9. Lower Bounds Loop - Sorted Matrix Search: The input consists of a real number x and a matrix $A[1..n, 1..m]$ of nm real numbers such that each row $A[i, 1..n]$ is sorted and each column $A[1..n, j]$ is sorted. The goal is to determine whether or not x appears in the matrix. You must prove a lower bound that matches your upper bound from the earlier question. Use the technique of specifying an input I and defining the set J of indexes that can be changed to give a different answer.

   (a) The lower bound is easier when the matrix is square, ie. $n = m$. Start there.

   (b) When $m >> n$, the upper bound you came up with from the previous question is likely wrong. A later question gives a recursive algorithm that accesses $T(n, m) = n \log_2(\frac{m}{n})$ elements when $m >> n$. Prove a matching lower bound.

10. Amortized Analysis: Suppose we have a binary counter such that the cost to increment the counter is equal to the number of bits that need to be flipped. For example, incrementing from $100, 111, 111_2$ to $101, 000, 000_2$ costs 7. Consider a sequence of n increments increasing the counter from zero to n. Some of these increments, like that from $101, 111, 010_2$ to $101, 111, 011_2$ costs only one. Others, like that from $111 \ldots 111_2 = 2^{\lfloor \log_2 n \rfloor} - 1 \approx n$ to $1000 \ldots 000_2$ costs $\mathcal{O}(\log n)$. This one is the worst case. Given that we generally focus on worst case, one would say that operations cost $\mathcal{O}(\log n)$. It might be more fair, however, to consider the average case. If a sequence of n operations requires at most $T(n)$ time, then we say that each of the operations has *amortized cost* $\frac{T(n)}{n}$.

   (a) Suppose the counter begins at zero and we increment it n times. Compute the Theta of the amortized cost per increment.

   Hint: Let $T_{x,j} = 1$ if incrementing value x requires flipping the j^{th} bit.

   (b) Suppose that we want to be able to both increment and decrement the counter. Starting with a counter of zero, give a sequence of n of operations where each is either an increment and decrement operation, that gives the highest amortized cost per operation (within Θ). (Don't have the counter go negative.) Compute this amortized cost.

   (c) To fix the problem from part (b), consider the following redundant ternary number system. A number is represented by a sequence of trits $X = x_{n-1} \ldots x_1 x_0$, where each x_i is 0, +1, or -1. The value of the number is $\sum_{i=0}^{n-1} x_i \cdot 2^i$. For example, $X = 101_2$ is 5 because $4 + 1 = 5$. However, $X = (1)(1)(-1)_2$ is also 5 because $4 + 2 - 1 = 5$. Also $X = (1)(-1)(0)(-1)(1)(1)(1) = 64 - 32 - 8 + 4 + 2 + 1 = 31$.

   The process of incrementing a ternary number is analogous to that operation on binary numbers. One is added to the low order trit. If the result is 2, then it is changed to 0, and a carry is propagated to the next trit. This process is repeated until no carry results. For example, $X = (1)(-1)(0)(-1)(1)(1)(1) = 31$ increments to $X + 1 = (1)(-1)(0)(0)(0)(0)(0) = 64 - 32 = 32$. Note this is a change of $8 - 4 - 2 - 1 = 1$. Decrementing a number is similar. One is subtracted from the low order trit. If it becomes -2 then it is replaced by 0, and a borrow is propagated. For example, $X + 1 = (1)(-1)(0)(0)(0)(0)(0)$ decrements to $X = (1)(-1)(0)(0)(0)(0)(-1) = 64 - 32 - 1 = 31$. Note that this increment followed by a decrement resulted in a different representation of the number 31 than the one we started with.

   The cost of an increment or a decrement is the number of trits that change in the process. Our goal is to prove that for any sequence n of operations where each is either an increment and decrement operations, starting with a counter of zero, the amortized cost per operation is at most $\mathcal{O}(1)$. Let $T(t)$ be the total number of trit changes that have occurred during the first t operations. Let $Z(t)$ be the number of non-zero trits in our counter after the t operations. For example, if the resulting counter is $X = (1)(-1)(0)(-1)(0)(1)(1)$ then $Z(t) = 5$. We will define $P(t) = T(t) + Z(t)$ to be our *measure of progress* or *potential function* at time t.

   Bound the maximum amount that $P(t)$ can change in any one increment or decrement operation.

   (d) Use induction (or loop invariants) to bound $P(t)$. In doing so, prove that for any sequence n of operations where each is either an increment and decrement operations, starting with a counter of zero, the amortized cost per operation is at most $\mathcal{O}(1)$.