1. [2 marks] If an algorithm runs in O(*nlgn*), it is not possible to find another algorithm that performs *worse* on the same task:

   a) True
   b) False

2. [2 marks] Which of the following are true (circle all that apply);

   a) Insertion sort is always a stable sort.
   b) Mergesort is a fragile sort.
   c) Quicksort and mergesort are O(n^2) in the worst case.
   d) Quicksort is O(n^2) in the average case, but O(*nlgn*) in the best case.
   e) Since they are divide & conquer algorithms, Mergesort and Quicksort tie in the best case.
   f) Insertion sort is constant time in the best case.
   g) All of the above are false.

3. [2 marks] What is the most appropriate big-0 complexity for the following:

```
for k=1 to 10n
{
    print k
    print n*n
}
```

   a) O(*n*).
   b O(*10n*).
   c) O(1).
   d) O(*k*).
   e) O(n^2)
   f) None of the above.

4. [2 marks] What is the most appropriate big-0 complexity of the task of finding *and* deleting an element in a singly linked list in the worst case?
   a) O(*n*)
   b) O(1)
   c) O(*n*+1)
   d) O(n^2)
   e) None of the above.

5. [2 marks] Tail recursion (when compared to regular recursion) is:
   a) More efficient because there is no need to store state information across the recursive steps
   b) More efficient because a tail recursive call must be stored on the heap
   c) Equivalent to regular recursion, but greatly increases readability
   d) Less efficient because the state information must be stored across each recursive step
   e) Less efficient because a tail recursive call must be stored on the heap.

6. [6 points total]

Suppose $f(x) = \frac{1}{x^2}$ s.t. : $f : R \to R$

a) What is the domain of *f*?

a) What is the codomain of *f*?

a) What is the range of *f*?

a) Is *f* onto, one-to-one, bijective, or neither? Show your justification.

7. [6 points] Consider the following small C++ code segment, which compiles and runs. After each line, tell us the contents of `x` and `y`. If at any point you cannot determine the contents, write "*unknown*".

```
#include <iostream>
using namespace std;

int main ()  {
  int x = 5, y = 15;
  int * p1, * p2;

  p1 = &x;      // x contains _____; y contains _____

  p2 = &y;      // x contains _____; y contains _____

  *p1 = 10;     // x contains _____; y contains _____

  *p2 = *p1;    // x contains _____; y contains _____

  p1 = p2;      // x contains _____; y contains _____

  *p1 = *p2+10; // x contains _____; y contains _____

  return 0;
}
```

8. [4 points] Compute Big-O for the following (be sure to show *c*; you may assume n_0 is 0):

$$f(x) = x! * x$$

9. [10 points] Write a function `merge` with the following specification (you may assume that sorted means from smallest to largest integer):

```
void merge( int a[], int n1, int b[], int n2, int c[])
```

Preconditions: `a` is a sorted array of integers, indexed from 0 to `n1`-1; `b` is a sorted array of integers, indexed from 0 to `n2`-1.

Postconditions: `c` contains all the items from `a`, plus all the items from `b`, in sorted order, indexed from 0 to `n1+n2`-1; that is, the array c contains the merge of the items in `a` and `b`. You may assume that c is declared to be of size `n1+n2`.

Use the space given to plan your algorithm before you start coding.

Hint: Recall the merge process that we discussed in class. There were two steps: comparing `a` and `b` and adding the smallest element to `c`, and handling the remaining data once `a` or `b` was empty.

Q3. *Continued*.

10. [6 marks total] This question is about program correctness. Consider the following function:

```
int sum(int n)
{
    int s = 0;
    for (int i = 1; i <= n; i++)
     {
        s = s + i;
     }
    return s;
}
```

a. [2 mark] Identify and state the loop invariant.

I(n) :

b. [4 marks] Use your loop invariant to prove that the function correctly computes the sum of a series of n integers, where $n \geq 1$.

11.[4 points] Apply quicksort to the following array. Using the first element in the array as the pivot, draw a tree showing the array after each split and partition. Circle each pivot after the partition is complete. You are done splitting once you reach a single element leaf. At each partition, be sure to preserve the relative order of the elements by simply shuffling them to the left or to the right to accommodate the pivot.

10	12	7	8	21	15	11	2	-3

DO NOT DETACH THIS SHEET.

Scratch space: