CSci 115, Spring 2015, Midterm 1

Name (please print): _____ ID #: _____

- This is a 75-minute closed-book exam. There are **five** questions worth a total of 100 points. Budget your time so you get to each question.
- Please read **all** of the instructions carefully before beginning to work on a problem, and if anything is not completely clear, please raise your hand and ask.
- All code is C++.

1. _____
2. _____
3. _____
4. _____
5. _____

Total _____

(for grader use only)

Question 1 (20 = 6+6+8 points). Mathematical preliminaries.

(a) Find a closed-form solution to the sum $\sum_{i=0}^{n} 6i^2+2i+1$ as a polynomial in n.

(b) Find a solution to the recurrence relation

F(0) = 2

F(n+1) = F(n) + 3n + 1

as a formula involving the summation operator $\sum_{i=1}^{n}$. You do **not** need to find a closed-form solution.

(c) Prove the following identity by mathematical induction (on n): $\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$

Question 2 (14 = 8 + 6 points). Asymptotic analysis.

(a) For each of the following pairs of functions f(n) and g(n) below, fill in the middle column with either o, O, Ω, or Θ to indicate whether f is in o(g), O(g), Ω(g), or Θ(g).

f(n) is	**o, O, Ω, or Θ**	**of g(n)**
$\log(n^2)$		$\log(n^3)$
2^n		3^n
$(\log n)^5$		$n^{1/5}$
n^n		2^n

f(n) is	**o, O, Ω, or Θ**	**of g(n)**
log 100		100
n log n		$n^{11/10}$
$5n^2+16n+2$		n^2
n^n		n!

(b) Determine Θ for the running time of the following code fragments (as a function of n):

```
sum = 0;
for (j = 0; j < n; j++)
   for (k = j; k > 0; k /= 2)
      sum++;
```

Θ(_____)

```
sum = 0;
for (j = 0; j < n; j++)
   for (k = j; k < n – j; k++)
      sum++;
```

Θ(_____)

```
sum = 0;
for (j = 0; j < n; j += j)
   for (k = 0; k < j; k ++)
      sum++
```

Θ(_____)

Question 3 (24 = 8+8+8 points). Linear data structures

(a) Complete the definition of this function to insert an integer into a singly linked list of integers.

```
class Link {
public:
   int element;
   Link *next;
   Link() {}
}

void insert(Link *here, int val) {









}
```

(b) Complete the definition of this function to delete (and free) a node from a doubly-linked list. You may assume that these lists have extra header and trailer nodes:

```
class Link {
public:
   int element;
   Link *prev, *next;
   Link() {}
}

void delete(Link *this_one) {









}
```

(c) Complete the definition of this function to insert an integer into a array-based circular queue of integers, where elements are inserted at the front and taken out at the back. This data structure has the following invariants that should be maintained by this function:

- `front` is the first index **not** currently in the queue,
- `back` is the index of the last element in the queue, or is equal to `front` if the queue is empty,
- both `front` and `back` are >= 0 and < `MAX_SIZE`.

The function should return `true` if the insert is successful and `false` if not (i.e., the queue is full).

```
class circQueue {
public:
   int elements[MAX_SIZE];
   int front, back;
}

bool insert(circQueue &q, int val) {









}
```

Question 4 (26 = 8 + 18 points). Tree-based data structures

(a) Complete the definition of this function to delete the maximum element in a BST, store its value in `val`, and return the root of the resulting BST. You can assume that `root != NULL`.

```
class BSTNode {
   int element;
   BSTNode *left, *right;
}

BSTNode *deletemax(BSTNode* root, int &val) {









}
```

(b) Complete the definitions of the following five functions for an array-based **min-heap** of integers, using no helper functions except `swap`:

```
class minHeap {
public:
   int elements[MAX_SIZE];
   int n;   // number of elements in heap

   bool isLeaf(int pos) { return _____________________________ ; }

   int leftChild(int pos) { return _____________________________; }

   int rightChild(int pos) { return _____________________________; }

   int parent(int pos) { return _____________________________; }

   void swap(int pos1, int pos2) {
      int temp = elements[pos1];
      elements[pos1] = elements[pos2];
      elements[pos2] = temp;
   }
}

// deletes minimum element in heap, storing the value in val;
// returns true if successful, false if not (i.e., heap was empty)
bool deletemin(minHeap &h, int &val) {









}
```

Question 5 (16 = 8+8 points). Applications.

(a) Here is a recursive definition of the Tower of Hanoi program:

```
void TOH(int n, Pole start, Pole goal, Pole temp) {
   if (n == 0) return;
   TOH(n-1, start, temp, goal);  // recursive call
   cout << "move " << start << " to " << goal << endl;   // move
   TOH(n-1, temp, goal, start);  // recursive call
}
```

that could be called, for example, as `TOH(5,1,2,3)` to move 5 disks from pole 1 to pole 2, using pole 3 as a temporary. And here is a stack ADT suitable for implementing this function without recursion:

```
class TOHstack {
public:
   void push(bool toh, int n, int start, int temp, int goal);
   void pop(bool &toh, int &n, int &start, int &temp, int &goal);
   bool isEmpty();
   TOHstack();
}
```

The `push` function pushes all five of its arguments onto the stack as a group, and `pop` pops them back off again, storing them into the reference parameters. The boolean `toh` is `true` when the record represents a recursive call to `TOH` with the indicated arguments, and is `false` when it represents the printing of a move from `start` to `goal` (in which case the values of `n` and `temp` are immaterial).

Implement a non-recursive version of `TOH` below that uses a `TOHstack` to simulate the recursive calls; for example, a call to `TOHnonrec(5)` should print the same thing as `TOH(5,1,2,3)`.

```
void TOHnonrec(int n) {









}
```

(b) Draw the Huffman tree that results from running the `buildHuff` algorithm on an alphabet with the following weights:

Letter	Q	Z	F	M	T	S	O	E
Frequency	2	3	10	10	10	15	20	30