University of Toronto
Faculty of Applied Science and Engineering
csc191s
Computer Algorithms, Data Structures and Languages

**Midterm Test — Solutions**

Tuesday 06 March 2001

Time Allowed: 2 hours; Aids Allowed: none
Examiner: Diane Horton

Family Name: ___________________________ Given names: ___________________________

Student #: ___________________________

- There are 13 pages, including this one.
- Do no remove any sheets from this test book. Answer all questions in the space provided. Use the reverse of a page if necessary. No additional sheets are permitted.
- Write your name and student number above, and at the top of each sheet of this test book.
- For questions that involve writing code, comments are not necessary. If you need to call a standard function but can’t remember the correct order of arguments, just indicate the meaning of each argument.

<table>
<thead>
<tr>
<th>Part A: _____ / 12</th>
<th>Part F: _____ / 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part B: _____ / 8</td>
<td>Part G: _____ / 6</td>
</tr>
<tr>
<td>Part C: _____ / 14</td>
<td>Part H: _____ / 20</td>
</tr>
<tr>
<td>Part D: _____ / 5</td>
<td>Part I: _____ / 13</td>
</tr>
<tr>
<td>Part E: _____ / 6</td>
<td>Part J: _____ / 8</td>
</tr>
</tbody>
</table>

Total _____ / 100
Part A [12 marks in total] — ADTs

Here are three possible implementations of the Priority Queue ADT:

- Implementation A: A singly-linked list, kept in order with the highest priority items at the front. Within a priority, the items are ordered according to when they were inserted, with the oldest items nearer the front. The only instance variable is a pointer to the front of the linked list.

- Implementation B: A singly-linked list, ordered only according to when the items were inserted, with the oldest items nearer the front. The only instance variable is a pointer to the front of the linked list.

- Implementation C: An array with one entry per priority level. Position $i$ of the array stores a pointer to the first node in a linked list of items with priority $i$. The linked list is kept in order according to when the items were inserted, with the oldest items nearer the front. The only instance variable is the array.

1. Suppose we have a priority queue with $p$ different priority levels and a total of $n$ items. Fill in the table below to show the big-oh time complexity of enqueue and dequeue under each implementation, in the worst case. Assume the functions are written efficiently.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>enqueue</th>
<th>dequeue</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>B</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>C</td>
<td>$O(n)$</td>
<td>$O(p)$</td>
</tr>
</tbody>
</table>

2. Under what conditions would implementation C be the best of the 3 choices?

   $p$ must be finite and not too large, and we must be using a reasonably high percentage of the priorities. (Still, we would want the additional pointers mentioned below.)

3. Is there an additional instance variable that would improve the big-oh time complexity of implementation A? Explain.

   A pointer to the end of the linked list wouldn’t help because you still need to search for the right spot to enqueue a new item in, based on its priority. But an array of pointers to the last node at each priority level would permit $O(1)$ enqueue.

4. Is there an additional instance variable that would improve the big-oh time complexity of implementation C? Explain.

   A pointer to the first non-empty entry would allow $O(1)$ dequeue. A pointer to the tail of each linked list would allow $O(1)$ enqueue.
Part B [8 marks in total] — Templates

Consider this class template:

```cpp
// File: Bunch.h

#ifndef BUNCH_H
#define BUNCH_H

template <class Type>
class Bunch {
    private:
        Type contents[100];
        int size;
    public:
        Bunch(){size = 0;}
        void insert(Type *item);
        int count(Type *item);
    }; // endclass
#endif

// File: bunchImp.h

#ifndef BUNCHIMP_H
#define BUNCHIMP_H

#include "bunch.h"

template <class Type>
void Bunch<Type>::insert(Type *item) {
    if (item->floopy())
        contents[++size] = *item;
}

template <class Type>
int Bunch<Type>::count(Type *item){
    int ans = 0;
    for (int i=0; i<size; i++)
        if (*((contents[i]) < *item) {
            count++;
            contents[i].shrink();
        }
    return count;
}
#endif
```

1. Why is it better to make the name of the file on the right end in `.h` rather than `.cpp`?
   Because we don’t compile it.

2. Suppose a class called Silly has been defined. Write a program that makes a Bunch of Sillys and then calls functions `insert` and `count`. It doesn’t matter what your program does. When you need to instantiate a Silly, use its no-argument constructor.

   ```cpp
   #include "bunchImp.h"
   #include "silly.h"

   int main(void){
       Bunch<Silly> b;
       Silly s();
       b.insert(&s);
       int n = b.count(&s);
       return 0;
   }
   ```

3. Suppose your program is in file `test.cpp`. Write the simplest command that will compile your program using `g++`.
   ```bash
g++ test.cpp silly.cpp
   ```

4. Suppose your program compiles and runs without crashing. What must be true about Silly?
   It must have a shrink function and a floopy function, and it must have overloaded the ‘<’ operator.
Part C [14 marks in total] — Big-oh

1. (2 marks each)

What is the worst-case big-oh time complexity of each of the following problems? Assume that the problems are solved efficiently using appropriate algorithms.

Given a queue containing \( q \) items (and represented as a circular array), and a stack containing \( s \) items (and represented as a linked list), remove the head of the queue and push it onto the stack.

**Answer:** \( O(1) \)

Given a tree of \( n \) nodes and height \( h \), return the number of occurrences of a given object.

**Answer:** \( O(n) \)

Given a binary search tree of \( n \) nodes and height \( h \), return the maximum value in the tree.

**Answer:** \( O(h) \)

2. (3 marks)

Suppose that we have declared all the variables that appear in the code below, and that we have read appropriate values into the integers \( r \) and \( s \).

```c
m = r;
p = 15;
for (int i = 0; i < r; i++) {
    k = m;
    while (k > 0) {
        if ( (p%4) == 1 ) {
            for (int j=1; j<s; j++) {
                cout << "hello" << endl;
            }
        } else {
            k--;
        }
    }
    k--;
}
```

What is the worst-case big-oh time complexity of this code? Give the strongest answer you can, that is, the tightest bound.

**Answer:** \( O(r^2) \)
3. (1 mark each; -1 for each incorrect answer)
   Which of the following are true?

   - $2^n$ is $O(n^2)$ \quad TRUE \quad FALSE
   - $n^2$ is $O(2^n)$ \quad TRUE \quad FALSE
   - $n \log n$ is $O(n)$ \quad TRUE \quad FALSE
   - $3^n$ is $O(2^n)$ \quad TRUE \quad FALSE
   - $n \log n$ is $O(n^3)$ \quad TRUE \quad FALSE

Part D [5 marks in total] — QuadTrees

Suppose we have constructed a quadtree for a $2^r$ by $2^r$ image.

1. What is the minimum height that the quadtree may have? Recall that the height of a tree is the number of nodes on the longest path from the root to any leaf.

   \[ 1 \]

2. What is the maximum height that the quadtree may have?

   \[ r + 1 \]

3. What is the minimum number of nodes that the quadtree may have?

   \[ 1 \]

4. What is the maximum number of nodes that the quadtree may have?

   \[ \sum_{i=0}^{r} 4^i \]

5. What is the big-oh time complexity of computing the area of this quadtree, in the worst case?

   \[ O(4^r) \]
Part E [6 marks in total] — Stacks

Below is the outline of a very simple stack class for a stack of integers.

```cpp
#ifndef STACK_H
#define STACK_H

class Stack {
private:
  // Data members omitted.
public:
  // Construct me as an empty stack.
  Stack();

  // Pop off my top item and return it
  // Precondition: I am not empty.
  int pop();

  // Push n into me.
  void push(int n);

  // Return whether or not I am empty.
  bool isEmpty();
};
#endif
```

Write a function, to go outside the class, that deletes and returns the bottom integer from a stack.

```cpp
int deleteBottom(Stack *s){
  Stack other;
  while(!s->isEmpty()) {
    other.push(s->pop());
  }
  int answer = other.pop();
  while (!other.isEmpty()){
    s->push(other.pop());
  }
  return answer;
}
```
Part F [8 marks in total] — Design by Contract

Below is a class that counts integers.

```cpp
// File: Counter.h
#ifndef COUNTER_H
#define COUNTER_H

class Counter {
private:
  int n;
  int* count;
  int numNums;
  int total;

public:
  Counter(int capacity);
  void tally(int i);
  int report(int i);
  double average();
  int biggestICanCount();
};
#endif
```

1. Write an appropriate representation invariant for this class.

   • \( n \geq 1 \).
   • the capacity of `count` is \( n \).
   • `count[i]` is the number of times that \( i \) has been tallied (for all \( i \) from 0 to the end of array `count`).
   • `numNums` is the number of times `tally` has been called (or equivalently, the number of numbers that have been tallied). `numNums \geq 0`.
   • `total` is the total value of all the numbers that have been tallied (or equivalently, \( \sum_{i=1}^{n-1} \text{count}[i] \times i \)). `total \geq 0`.

2. Write an appropriate external comment for function `report`.

   • Returns the number of times that \( i \) has been tallied.
   • Preconditions: \( 0 \leq i \leq \text{biggestICanCount}() \)
     (which is the upper range of this NumberCounter).
   • This answer should not mention any member variables or reveal anything about the implementation of the class.
Part G [6 marks in total] — Recursion

Consider the following function, and the BinaryIntNode class it uses.

```cpp
int BST::mystery(IntNode* root)
{
    if (root->left == 0 && root->right == 0)
        return root->data;
    else if (root->left == 0)
        return mystery(root->right);
    else if (root->right == 0)
        return mystery(root->left);
    else
        return mystery(root->left) + mystery(root->right);
}
```

// File IntNode.h:

```cpp
#ifndef INTNODE_H
#define INTNODE_H
class IntNode {
public:
    IntNode* left;
    IntNode* right;
    int data;
};
#endif
```

1. (2 marks)
What value is returned if we call mystery with a reference to the root of this binary tree:

```
         99
          / \
        42   18
       /   / \
      1   9   7
     /     / \
    12   8   3
     /   / \
    5   3   4
     /   \
    11
```

**Answer:** 30.

2. (1 mark)
How many calls to mystery occur, in total, as a result of that initial call to mystery? Include the initial call in your total.

**Answer:** 12.

3. (3 marks)
Write an appropriate external comment for mystery.

*Returns the sum of the values in all the leaves of the tree rooted at 'root'. Precondition: root is not null.*
Part H [20 marks in total] — Trees and Recursion

1. (3 marks)
What would be the output if we printed the tree below in ...

```
   42
  /  \
 99   18
 /  \
1   9
/  \
12  8   7
/  \\  \
5   3  4
```

- preorder: 42 99 1 12 8 5 11 18 9 7 3 4
- inorder: 12 1 11 5 8 99 42 9 3 7 4 18
- postorder: 12 11 5 8 1 99 3 4 7 9 18 42

2. (1 mark each; -1 for each wrong answer)
Circle each operation that be done just as easily without recursion as with it:

- (a) Counting all occurrences of an integer in a binary tree.
- (b) **Counting all occurrences of an integer in a binary search tree.**
- (c) Counting all occurrences of an integer in a doubly-linked list.
- (d) Determining whether an integer occurs in a binary tree.
- (e) **Determining whether an integer occurs in a binary search tree.**

3. (3 marks)
Describe how you would implement in-order traversal of a binary tree without using recursion.

*Keep a stack to keep track of the nodes that I’m not yet done with. Push as I go down and pop as I go up. For each thing on the stack, remember which of its subtrees I’ve already traversed.*

4. (9 marks)
Recall that the height of a tree is the number of nodes on the longest path from the root to any leaf.

Let the **imbalance of a node** be the absolute value of the difference between the heights of its left and right subtrees. For example, the imbalance of the node labelled 99 in above is 4, and the imbalance of node 1 is 2.

Let the **maximum imbalance of a tree** be the maximum of the imbalances of all of its nodes. The maximum imbalance of an empty tree is 0.

On the next page is a node class and the declaration of a BST class that uses it.
// File BST.h:

#ifndef BST_H
#define BST_H

#include "IntNode.h"

class BST {
private:
    IntNode* root;
public:
    // Declarations of other public functions omitted.

    // Return my maximum imbalance.
    int maxImbalance();
};

#endif

// File IntNode.h:

#ifndef INTNODE_H
#define INTNODE_H

class IntNode {
public:
    IntNode * left;
    IntNode * right;
    int data;

};

#endif

Write function maxImbalance. You may add helper functions to class BST, but do not change
the code in any other way.

int BST::maxImbalance(){
    int h; // height of me.
    int i; // maxImbalance of me.
    BST::maxImbalance(root, h, i);
    return i;
}

// This function should be declared "static" in the BST.h file:

void BST::maxImbalance(IntNode * root, int& height, int& mi){
    if (root == 0) {
        height = 0;
        mi = 0;
    } else {
        // The height of my left and right subtrees, and their own
        // max imbalances.
        int lh, rh, lmi, rmi;
        maxImbalance(root->left, lh, lmi);
        maxImbalance(root->right, rh, rmi);
        height = max(lh, rh) + 1;
        mi = max( abs(lh-rh), max(lmi, rmi) );
    }
}
Part I [13 marks in total] — Files

1. (5 marks)
Consider the following program:

```cpp
#include <fstream>
int main(void){

    unsigned char u = 17;
    char c = 'A';
    char *p = new char[10];
    p[0] = 'A';
    p[1] = 'B';

    fstream out;
    out.open("outfile", ios::out | ios::bin);
    out << c;
    out.write(&c, sizeof(char));
    out.write(&u, sizeof(unsigned char));
    out.write(&p, 2*sizeof(char));
    out.write(p, 2*sizeof(char));
    
    return 0;
}
```

For each of the following statements, show exactly what would be written to the file, byte by byte. Write your answer in octal, using 3 digits per byte. Assume sizeof(char) and sizeof(unsigned char) are 1. If there is not enough information to answer, explain why.

Note: The ascii value of 'A', is 65 (base 10).

```plaintext
out << c;

101

out.write(&c, sizeof(char));

101

out.write(&u, sizeof(unsigned char));

021

out.write(&p, 2*sizeof(unsigned char));

Not enough information because p is a pointer and we don't know its actual value.

out.write(p, 2*sizeof(unsigned char));

101 102
```
2. (4 marks)
Let \( i \) be `sizeof(int)` and \( c \) be `sizeof(char)`. Suppose we have declared and initialized an integer \( n \).

Suppose that writing \( n \) to a file as text would consume more file space than writing it in binary. What does this tell you about \( n \)?

(\textit{the number of digits in } \( n \) \textit{) \( \times \) \( c \) \textit{is greater than } \( i \)\)

Suppose that writing \( n \) to a file as text would consume \textit{less} file space than writing it in binary. What does this tell you about \( n \)?

(\textit{the number of digits in } \( n \) \textit{) \( \times \) \( c \) \textit{is less than } \( i \)\)

3. (4 marks)
You are designing an operating system similar to Unix but with some variations. You want to have a maximum file size of \( 2^{32} \) blocks. Your blocks are \( 2^{16} \) bytes and your pointers each take 4 bytes.

You are considering an inode that has 12 direct pointers, 2 pointers each to a single-level index and 1 pointer to a 2-level index.

State the maximum number of blocks addressable by your inode. You may give this answer as an equation.

\textbf{Answer:} \( 12 + 2^{15} + 2^{28} \) blocks

\textbf{Answer explanation:}
\( 2^{16} \) bytes per block and 4 bytes pointers means \( 2^{14} \) pointers per block

- first 12 blocks are directly accessible
- each 1-level index gives access to the next \( 2^{14} \) blocks
- 2-level index gives access to the next \( 2^{14} \cdot 2 = 2^{28} \) blocks

\[
\text{total} = 12 + 2 \cdot 2^{14} + 2^{28}
\]

\[
= 12 + 2^{15} + 2^{28}
\]

Is this sufficient to address all the blocks in the file? (circle one) \hspace{1cm} \textbf{YES} / \hspace{0.5cm} \textbf{NO}

\textbf{Answer explanation:} (a back-of-the-envelope calculation)
\( 12 + 2^{15} < 2^{16} \) \( \text{so the total is } < 2 \cdot 2^{16} = 2^{17} \cdot 29 \leq 2^{32} \)

You are considering changing the maximum file size but it must be a power of 2. What is the largest \( n \) for which a file of \( 2^n \) blocks can be addressed by your inode structure?

\textbf{Answer:} 28, since \( 2^{28} < \text{total} < 2^{29} \)
Part J [8 marks in total] — *Encoding*

1. (3 marks; -1 for each wrong answer)
   Below are three separate sets of codes. For each set, circle YES or NO to indicate if the set is instantaneous.

   101, 01, 1001, 1000, 110, 111
   [YES / NO]

   11, 10, 01, 00
   [YES / NO]

   111, 110, 01, 00, 100, 1101, 1011
   [YES / NO]

2. (5 marks)
   Consider a file which contains only four different characters. The relative frequencies of the characters are shown in the table below.

<table>
<thead>
<tr>
<th>character</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20%</td>
</tr>
<tr>
<td>B</td>
<td>75%</td>
</tr>
<tr>
<td>C</td>
<td>4%</td>
</tr>
<tr>
<td>D</td>
<td>1%</td>
</tr>
</tbody>
</table>

   If the file contained $x$ characters in total, and we were to use fixed length binary codes, what is the minimum number of bytes it would take to encode the file?

   **ANSWER:** $2x$ bits = $x/4$ bytes

   If we were to use Huffman coding, how many bytes would be needed to encode the file?

   **ANSWER:** $1.3x$ bits = $(1.3/8)x$ bytes

   Construct a set of valid Huffman codes for the file and show the encoding it gives for the letter C. There are a number of correct answers.

   **ENCODING FOR C:** any 3-digit binary number is a correct answer.

*End of Examination*