CS 216 Exam 1 – Fall 2004 - SOLUTION

Name:_______________________  Lab Section: _________________

Email Address:_________________  Student ID # _________________

This exam is closed note, closed book. You will have an hour and fifty minutes total to complete the exam. You may NOT use calculators.

Good Luck!!

<table>
<thead>
<tr>
<th></th>
<th>MAX</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>104</td>
<td></td>
</tr>
</tbody>
</table>

Write and sign pledge after taking the exam:
Define each term and give an example that explains it:

1. (2 points) excess B

   Also known as biased notation. Method of representing integers where integer value K is represented as K + B, where B is the bias. In single precision IEEE floating point excess 127 is used to represent the exponent. Thus the exponent -120 is represented as 7 or: 00000111

2. (2 points) height of an AVL tree

   The length of the longest path from the root to a leaf node. The height of the tree in problem #16 is four. The height of a tree of a single node is zero.

3. (2 points) FIFO data structure

   A first-in-first-out data structure is one in which the first item inserted will be the first item removed. An example is a queue. After the sequence: enqueue(1), enqueue(2), enqueue(3), a call to dequeue() will return the value 1.

4. (2 points) abstract data type

   A set of values and the set of operations allowable on those values. Examples: stack, queue, list.
5. (2 points) Order the following rates of growth from slowest rate of growth to fastest rate of growth: \( n^8, n \log n, n, 2^n, 300, \log n \)

\[
300 \quad \log n \quad n \quad n \log n \quad n^8 \quad 2^n
\]

6. (3 points) Name 3 factors that are ignored by big-Oh notation.

Hardware, Operating system, programming language used, assumes that all operations take the same amount of time, ignores constant factors, throws out lower ordered terms.

7. (4 points) Fill in the blanks in the definition of big-Oh notation:

\[
T(N) = O(f(N)) \text{ if:}
\]

there are positive constants \( c \) and \( n_0 \) such that: \( T(N) \leq c \cdot f(N) \)

when: \( N \geq n_0 \)
8. (10 points total) Describe the running time of the following pseudocode in Big-Oh notation in terms of the variable \( n \). Assume all variables used have been declared. 

*Show your work for partial credit.*

```c
int vote_count(int k) {
    int sum;
    for (int i = 0; i < 500; ++i)
        sum = sum + (i * k);
    return sum;
}
```

a) \( \text{answ} = \text{vote\_count}(n) \); \( O(1) \)

b) \( \text{int sum; if (n < 100) cout << "whoa!";} \) else
   for (int i = 0; i < n; ++i)
       sum += \text{vote\_count}(n);

\( O(n) \)

c) for (int j = 4; j < n; ++j) {
    \text{cin >> val;}
    for (int i = 0; i < j; ++i) {
        b = b * val;
        for (int k = 0; k < n; ++k)
            c = b + c;
    }
}

\( O(n^3) \)

d) for (int i = 0; i < n * n; ++i) {
    sum = sum/n;
    for (int j = 0; j < i; ++j)
        \text{j >> cout;}
}

\( O(n^4) \)

e) for (int i = 0; i < n; ++i) {
    for (int j = 0; j < i * n; ++j)
        sum = sum + i;
    for (int k = 0; k < n + n; ++k)
        a[k] = a[k] + sum;
}

\( O(n^3) \)
9. (6 points total) What is the representation of each of the following in the indicated radix? Be sure to show your work.

a) $12_7$ in decimal

$$= 1 \times 7^1 + 2 \times 7^0$$

$$= 7 + 2 = 9_{10}$$

b) $1132_8$ in hex

$$= 1 \times 8^3 + 1 \times 8^2 + 3 \times 8^1 + 2 \times 8^0$$

$$= 1 \times 512 + 1 \times 64 + 3 \times 8 + 2 \times 1 = 512 + 64 + 24 + 2 = 602_{10}$$

$$\frac{602}{16} = 37 \text{ rem } 10$$

etc.  

$$= 25A_{16}$$

001 001 011 010 $\rightarrow$ 0010 0101 1010 (group the digits to get: $25A_{16}$)

c) $2E_{15}$ in radix 10

$$= 2 \times 15^1 + 14 \times 15^0$$

$$= 2 \times 15 + 14 \times 1 = 30 + 14 = 44_{10}$$

10. (6 points total) Consider the positive binary integer represented in two’s complement: $0110010110000111_2$.

a) Express this binary number in octal

0 110 010 110 000 111 $= 62607_8$

b. Express this binary number in hexadecimal

0110 0101 1000 0111 $= \text{x6587}$

c. Negate the number (i.e. give the two’s complement representation of a negative version of the same number) Use the same number of bits.

1001101001111001_2
11. (3 points) Draw the binary search tree created by inserting these values in this order:

```
4  1  9  6  7  2  5  0  8
```

```
4
  1
  0  2
  5
  6
    7
      8
```

12. (2 points) Give a pre-order traversal of your tree shown above:

```
4  1  0  2  9  6  5  7  8
```

13. (2 points) Give a post-order traversal of your tree shown above:

```
0  2  1  5  8  7  6  9  4
```

14. (3 points) Delete the root of the tree shown above using one of the methods described in class. Draw the new tree here:

```
1
  2
  0
  5
```

```
1
  5
  0
  2
  6
    7
      8
```
15. For each operation below give: 1) How you would most efficiently implement the operation, 2) Describe the worst case scenario (e.g. “The worst case occurs when the value you are looking for is not in the list”) and 3) What is the worst case Big-Oh running time of this scenario. 

State any assumptions you make.

a) (3 points) Find the maximum value stored in an AVL tree.

- Starting at the root, traverse to the right until hit a NULL right pointer (this is the largest value in the tree).
- All cases are roughly the same as the tree is balanced, but in the worst case the longest path in the tree is from the root to the largest value.
- O(log N) as this is the worst case height of the tree (length of longest path from the root to a leaf)

b) (3 points) Pop a value from a stack implemented as an array.

- return the value currently contained in the top of stack location, increment (or decrement) the top of stack location so this value is no longer pointed to.
- All cases are the same.
- O(1) as updating the top of stack location and returning the value are just constant time operations that do not depend on how many elements are currently in the stack.

c) (3 points) Find the total number of values stored in a queue implemented as a doubly linked list.

Two options:

Keep a “size” field updated:
- assume that there is a field that keeps track of how many elements are currently stored in the queue. This field is updated (in constant time) on enqueue and dequeue operations.
- All cases are equal
- O(1) to return the value stored in the field

Count the number of elements at the time it is requested:
- starting at either the head or tail, traverse to the end of the list, counting the number of elements as you go.
- All cases are equal
- O(N) as you must traverse the entire list, doing a constant time operation at each node (incrementing a count).
16. (3 points) Given the following tree:

Is it an AVL tree? If not, circle the node(s) where the AVL property is violated. Why or why not (must answer for any credit)?

No, at node 30, the height of the left subtree is 1 and the height of the right subtree is -1, so their difference is > 1 thus violating the AVL structure property.

17. (3 points) Given the following tree:

Is it an AVL tree? If not, circle the node(s) where the AVL property is violated. Why or why not (must answer for any credit)?

No, at nodes 20 and 70, the height of the left subtree is -1 and the height of the right subtree is 1, so their difference is > 1 thus violating the AVL structure property.
18. (7 points) Assume we are using the 32-bit IEEE single precision floating point format as described in class and used in lab. The mantissa has 24 bits including the hidden bit. There is one sign bit and there are eight exponent bits. The exponent is stored in excess 127.

What decimal floating point number is represented by the following 32 bits? SHOW YOUR WORK!

1000 1010 1111 0000 0000 0000 0000 0000

a) Is this a positive or negative number?  
   **Negative (sign bit is 1)**

b) What is the exponent (in base 10)?

   \[ 00010101_2 = 21_{10} \]

   \[ 21 - 127 = -106 \]

c) What is the value of the mantissa (in base 10)?

   \[ 1.111_2 \]

   \[ 1 + 1/2 + 1/4 + 1/8 = 1 + 0.5 + 0.25 + 0.125 \]

   \[ = 1.875 \]

d) What is the total value?  
   Note: you may leave your answer in the form: \[ value_{10} \times base^{exponent} \]  
   Where you specify value, base and exponent.

   \[ -1.875 \times 2^{-106} \]
19. (20 points) This question tests your understanding of stacks and pointer manipulation. You must implement a stack ADT in C++. The underlying representation of the stack should be the Node class as described below used in a singly linked list. Your stack should store integers and should handle errors (printing an error message is fine).

You will be graded mostly on the correctness of the ideas of your solution rather than exact C++ syntax, but your solution should be clear. Correct C++ code is the best way to ensure we understand your solution. You may NOT use the STL in any way for this question. You should use the header file provided below. You do not need to implement copy constructors, destructors, or operator= for these classes, but otherwise the routines you implement should handle memory management appropriately.

You should implement all the functions with \texttt{\~} in front of them.

```cpp
class Node {
public:
    Node(int value): val(value), next(NULL) {};
    int val;
    Node *next;
};

class Stack {
public:
    Stack(); // constructor
    void push(int value); // pushes value onto the stack.
    int pop(); // returns and removes the value on the top of the stack.
    int top(); // returns the value on the top of the stack without removing it.
    bool isEmpty(); // returns true if the stack contains no elements.
private:
    // Add data members here.
    Node *topOfStack;
};
```
Stack::Stack()
    topOfStack = NULL;
}

void Stack::push(int val){
    Node *temp = new Node(val);
    if (isEmpty()) { // Add the first item to the stack.
        topOfStack = temp;
    }  else {
        temp->next = topOfStack;
        topOfStack = temp;
    }
}

// returns -1 if empty
int Stack::pop() {
    Node *temp;
    int return_val;
    if (!isEmpty()) { // At least one item is on the stack
        temp = topOfStack;
        topOfStack = topOfStack->next;
        return_val = temp->val;
        delete temp;
    }  else {   // empty stack
        cout << "Empty list";
        return_val= -1;
    }
    return return_val;
}

// returns -1 if empty
int Stack::top() {
    if (!isEmpty()) { // At least one item is on the stack
        return topOfStack->val;
    }  else {   // empty stack
        cout << "Empty list";
        return -1;
    }
}

bool Stack::isEmpty() {
    return !topOfStack;
}
20. (8 points) Implement a non-member function:

```cpp
void print_stack_inorder(Stack my_stack) {
    Stack tempStack;
    int temp_val;
    int current_min;

    while(!my_stack.isEmpty()) { // Do nothing w. empty stack
        current_min = my_stack.top();

        // Pop all values off of my_stack and push onto tempStack.
        while(!my_stack.isEmpty()) {
            temp_val = my_stack.pop();
            tempStack.push(temp_val);
            if (temp_val < current_min)
                current_min = temp_val;
        }

        // Pop all values off of tempStack and push onto my_stack.
        while(!tempStack.isEmpty()) {
            temp_val = tempStack.pop();
            // Print out the current min and don’t push back on.
            if (temp_val == current_min)
                cout << current_min << " ";
            else
                my_stack.push(temp_val);
        }
    }
    cout << endl;
}
```

OR for recursive solution, see next page.
void print_stack_inorder(Stack my_stack) {
    Stack tempStack;
    int temp_val;
    int current_min;

    if (my_stack.isEmpty()) return;
    current_min = my_stack.pop();
    while (!my_stack.isEmpty()){
        temp_val = my_stack.pop();
        if (temp_val < current_min) {
            tempStack.push(current_min);
            current_min = temp_val;
        } else tempStack.push(temp_val);
    }
    cout << current_min;
    print_stack_inorder(tempStack);
}

21. (2 points) What is the worst case big-Oh running time of your pop method and why?

O(1) – pop only involves constant time operations – updating pointers, deleting a single node. No operation depends on the number of elements in the stack (N).

22. (3 points) What is the worst case big-Oh running time of your print_stack_inorder function and why?

O(n^2) – The function consists of an outer while loop that has n iterations – one iteration for each value printed out. Inside of that outer loop are two other loops (not nested) that each take O(n) iterations (first time through the outer loop they each take n iterations, then n-1, n-2, n-3,…then finally only one iteration the last time through the outer loop). Summing these iterations we get (n * (n+1))/2 or O(n^2). Similarly multiplying the number of iterations of the outer loop (n) times the amount of work per iteration: n + n = O(n), we get O(n^2).

For recursive solution, there are n calls to a routine that takes time n on each call (maximum of n pops and pushes per call). So also O(n^2).