CS 216 Exam 1 – Spring 2003 - SOLUTION

Name:_______________________  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.

It is an Honor Code violation to discuss this exam with ANYONE (including other students who have already taken it) until after 5:30pm Tuesday, Feb 25.

Good Luck!

<table>
<thead>
<tr>
<th></th>
<th>MAX</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>103</td>
<td></td>
</tr>
</tbody>
</table>

Write and sign pledge after taking the exam:
1. (4 points) Complete the definition of big-Oh notation:

Definition: \( T(N) = O(f(N)) \) if:

there are positive constants \( c \) and \( n_0 \) such that \( T(N) \leq cf(N) \) when \( N \geq n_0 \).

2. (4 points) Explain the definition above to a programmer who has never heard of it. Feel free to use diagrams. (A diagram would be good here but is too hard to draw in MS Word)

When we say \( T(N) = O(f(N)) \) (read “\( T(N) \) is Big-Oh \( f(N) \)” or “\( T(N) \) is Order \( f(N) \)”), we are saying that \( T(N) \) (a function representing our algorithm) grows no faster than \( f(N) \). With big-Oh we are concerned about the performance of an algorithm for large values of \( N \), (e.g. sorting 1,000,00 elements is more interesting than sorting 10 elements). So we are NOT saying that if you graph \( T(N) \) and \( f(N) \), that \( f(N) \) will always be above \( T(N) \). We are instead saying that for large values of \( N \), (specifically for all values of \( N \) greater than some \( n_0 \)), that \( f(N) \) will be an asymptotic “upper bound” for \( T(N) \). I say “upper bound” in quotes because for \( f(N) \) to be graphed above \( T(N) \) then you may need to throw in a constant factor \( c \). This is o.k. because in general with big-Oh notation we ignore constants in our calculations.

Why do we ignore constant factors? Well actually big-Oh ignores lots of details: programming language, compiler, operating system, hardware, and programmer skills. We also assume that all operations take the same amount of time – e.g. a floating point division takes as long as an integer add, or an access to memory (when in reality these can take very different amounts of time). These details will often change and are hard to measure accurately. Instead we focus on how our algorithm performs as problem size \( N \) grows. Big-Oh is useful for figuring out where the bottlenecks in code might be and for choosing between two algorithms without having to implement and run them.

Many students are confused about worst case. The worst case for a particular sorting algorithm might be if the data is given to you sorted in reverse order. For that algorithm, this is the situation that will likely lead to the worst performance. Big-Oh doesn’t say what the worst case is. However if we use big-Oh to give an upper bound on the performance of our algorithm on the worst case data set, then we are also providing an upper bound on its performance on arbitrary inputs.

3. (2 points) How does big-omega (\( \Omega \)) differ from big-Oh?

\( T(N) = \Omega(f(N)) \) if there are positive constants \( c \) and \( n_0 \) such that \( T(N) \geq cf(N) \) when \( N \geq n_0 \).

Big-Omega (\( \Omega \)) provides an asymptotic lower bound on the performance of \( T(N) \). If we use Big-Omega to bound the performance of the best case data set, then we are also providing a lower bound on arbitrary data sets.
4. (3 points) With respect to big-Oh notation, order the following rates of growth from fastest rate of growth to slowest rate of growth: log n, 1000, n log n, \( n^3 \), \( 2^n \), \( n \)

<table>
<thead>
<tr>
<th>fastest rate of growth</th>
<th>slowest rate of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2^n ), ( n^3 ), ( n \log n ), ( n ), ( \log n ), ( 1000 )</td>
<td></td>
</tr>
</tbody>
</table>

5. (4 points) Your friend proposes a data structure to represent the temperature in Fahrenheit. She wants to use 5 bits to do this. a) How many different temperatures can we represent? and b) What is the range of positive and negative integer temperatures we can represent in two’s complement?

a) \( 2^5 = 32 \) different temperatures

b) -16 to 15 is the range

6. (4 points) In reality, temperatures we need to represent range from -10 F to 90 F. From cs216 we learned that biased notation can represent an asymmetrical range like this. a) What is the minimum number of bits we would need to represent this range, and b) What would the bias be, assuming that the temperature will NEVER go below -10 F, but might occasionally go above 90 F?

a) \( 2^6 = 64, 2^7 = 128 \), we need at least 7 bits

b) Bias = 10. (-10 will be represented as 000000.)

7. (4 points) Write the following infix expression as a prefix expression:

\[
(((a-g) + (e \div b)) \times c)
\]

\[
* \ + \ - \ a \ g \ / \ e \ b \ c \]

\[
\times \ /
\]

\[
+ \ c \ /
\]

\[
- \ /
\]

\[
/ \ /
\]

\[
/ \ /
\]

\[
/ \ /
\]

\[
/ \ /
\]

\[
da \ g \ e \ b\]
8. (10 points total) Describe the running time of the following pseudocode in Big-Oh notation in terms of the variable \( n \). \textit{Show your work for partial credit.}

a) \[
\text{int test(int k) \{}
\quad \text{if (k == 0)}
\quad \quad \text{return 1;}
\quad \text{else}
\quad \quad \text{return (test(k-1) + test(k-1))}
\}\]
\text{answ = test(n);}
\text{O}(2^n)

b) \[
\text{int super(int k) \{}
\quad \text{for (int i = 0; i < k; ++i) \{}
\quad \quad \text{return k;}
\quad \}\}
\text{answ = super(n);}
\text{O}(1)

c) \[
\text{for (int j = 0; j < n; ++j) \{}
\quad \text{cin >> a;}
\quad \text{for (int i = 0; i < j; ++i) \{}
\quad \quad \text{b = b + a * c;}
\quad \quad \text{for (int k = 0; k < n; ++k) \{}
\quad \quad \quad \text{c = b + c;}
\quad \quad \}\}
\quad \}\}
\text{O}(n^2)

d) \[
\text{for (int i = 0; i < 217; ++i) \{}
\quad \text{if (i < 100) \{}
\quad \quad \text{for (int j = 0; j < n; ++j) \{}
\quad \quad \quad \text{j >> cout;}
\quad \quad \}\}
\quad \}\}
\text{O}(n)

e) \[
\text{b = 1000;}
\text{for (int i = 0; i < n * n; ++i) \{}
\quad \text{for (int j = 0; j < i; ++j) \{}
\quad \quad \text{sum = sum + i;}
\quad \quad \text{for (int k = 0; k < b; ++k) \{}
\quad \quad \quad \text{a[k] = a[k] + sum;}
\quad \quad \}\}
\quad \}\}
\text{O}(n^4)
9. (6 points total) What is the representation of each of the following in the indicated radix? Be sure to show your work.

a) $169_{12}$ in decimal

$= 1 \times 12^2 + 6 \times 12^1 + 9 \times 12^0$

$= 144 + 72 + 9 = 225$

b) $3230_4$ in hex

$= 3 \times 4^3 + 2 \times 4^2 + 3 \times 4^1 + 0 \times 4^0$

$= 3 \times 64 + 2 \times 16 + 3 \times 4 = 192 + 32 + 12 = 236_{10}$

$236/16 = 14 \text{ rem } 12$

$14/16 = 0 \text{ rem } 14 = \text{ EC}_{16}$

OR $11_{10} 10_{10} 11_{10} 00 \rightarrow \text{(group the digits to get: EC}_{16})$

c) $2A6_{17}$ in radix 10

$= 2 \times 17^2 + 10 \times 17^1 + 6 \times 17^0$

$= 2 \times 289 + 10 \times 17 + 6 = 578 + 170 + 6 = 754_{10}$

10. (6 points total) Consider the positive binary integer represented in two’s complement: $0101110101110011_2$.

a) Express this binary number in octal

$0 101 110 101 110 011_2 = 56563_8$

b. Express this binary number in hexadecimal

$0101 1101 0111 0011_2 = 0x \text{ 5D73}$

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.

$1010 0010 1000 1101_2$
11. (3 points) Draw the binary search tree created by inserting these values in this order:

```
6 8 2 4 3 0 1 9 5 7
```

```
6
 / \
2 8
 / \ / \ 
0 4 7 9 \\
/ / \
1 3 5
```

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

```
6 2 0 1 4 3 5 8 7 9
```

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

```
1 0 3 5 4 2 7 9 8 6
```

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

OR

```
7
 / \
2 8
 / \ / \\
0 4 9 \\
/ / \
1 3 5
```

```
5
 / \
2 8
 / \ / \\
0 4 7 9 \\
/ / \
1 3
```
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) Finding the maximum element in the AVL tree:

1) **Find the rightmost element and return its value: start at the root and go right until there are no more right children.**

2) **No real worst case – the tree is balanced, but the maximum value could be the deepest node in the tree.**

3) **O(log N) – the worst case height of tree/depth of a node.**

b) (3 points) Print all the values in a binary search tree (BST) from smallest to largest.

1) **Do an in-order traversal and print each value as you visit it.**

2) **No worst case, must visit all nodes regardless of the shape of the tree.**

3) **O(N) – must visit all nodes.**

c) (3 points) Find the minimum value in a stack.

1) **(Assuming that your solution shouldn’t destroy the stack!)**

   If it is a member function, just examine all values currently in the stack, comparing to find the minimum as you go.

   If it is a non-member function, pop the items off of the stack and place them on temporary stack. Then examine the items as you push them back onto the original stack, comparing to find the minimum value as you go.

2) **No worst case, must examine all values in all cases.**

3) **O(N) – must examine all elements at least once in either case. For the non-member version, you must pop all elements O(N) and push onto another stack O(N), and then examine them as you push them back onto the original stack O(N), for a total of O(N).**
16. (3 points) Given the following tree:

```
      32
     /   \
   24    44
     \
   7
```

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)?

The height of 44’s left subtree = -1, the height of the right subtree = 1, thus they differ by more than one and therefore violates the AVL property.

17. (3 points) Given the following AVL tree:

```
      7
     /   \
   6     8
  / \   /   \
 5   7 9
```

Insert the value 4 into the AVL tree above, doing any necessary rotations to maintain the AVL property.

(requires a single rotation)
18. (3 points) Given the following AVL tree:

\[
\begin{array}{c}
5 \\
3 \\
2 \\
6 \\
8 \\
\end{array}
\]

Insert the value 7 into the AVL tree above, doing any necessary rotations to maintain the AVL property.

*(requires a double rotation)*

\[
\begin{array}{c}
5 \\
3 \\
2 \\
6 \\
7 \\
8 \\
\end{array}
\]

\[
\begin{array}{c}
3 \\
2 \\
6 \\
8 \\
7 \\
\end{array}
\]

\[
\begin{array}{c}
5 \\
3 \\
2 \\
6 \\
7 \\
8 \\
\end{array}
\]
19. (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!

0011 0011 0011 0000 0000 0000 0000 0000

a) Is this a positive or negative number? positive

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

0110 0110 is stored in excess 127 (i.e. any value is represented as value + 127)

Thus 64 + 32 + 4 + 2 = 102
If value + 127 = 102, then value must be -25.

The exponent is -25.

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

The mantissa is: 1.0112 which translates to:

\[
1 + \frac{1}{4} + \frac{1}{8} = \frac{11}{8} \\
1 + .25 + .125 = 1.375_{10}
\]

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.

\[1.1375_{10} \times 2^{-25}\]
20. (21 points total) a) (12 points) This question tests your understanding of queues and pointer manipulation. You must implement a queue ADT in C++. The underlying representation of the queue should be the Node class as described below. Your queue 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 implement all the functions with \( \rightarrow \) in front of them.

class Node {
    public:
        Node(int value) {val=value; prev = next = NULL;};
        int val;
        Node *prev;
        Node *next;
    };

class Queue {
    public:
        \( \rightarrow \) Queue(); // constructor
        \( \rightarrow \) void enqueue(int value); // enqueues value into the queue
        \( \rightarrow \) int dequeue(); // returns and removes the value at
            // the head of the queue
        \( \rightarrow \) bool isEmpty(); // returns true if the queue contains
            // no elements.
    private:
        Node *head;
        Node *tail;
        int size;
    };

Queue::Queue() {
    head = NULL;
    tail = NULL;
    size = 0;
}

void Queue::enqueue(int value) {
    Node *temp = new Node(value);
    if (isEmpty())
        head = tail = temp;
    else {  // there is at least one element in the queue
        tail->next = temp;
        temp->prev = tail;
        tail = temp;
    }
    size++;
}

int Queue::dequeue() {
    if(isEmpty()) {
        cout << "Queue is empty!" << endl;
        return -1;
    } else {  // Find a new head node
        Node *temp = head;
        int temp_val = temp->val;
        head = head->next;
        size--;
        if (isEmpty())
            head = tail = NULL;
        else
            head->prev = NULL;
        // avoid a memory leak.
        delete temp;
        return temp_val;
    }
}

bool Queue::isEmpty() {return (size==0);}
b) (5 points) Implement a member function `void printQueue(bool chron)` in C++. `printQueue` should print out all elements currently in the queue in the direction specified by its `chron` parameter. If `chron` is true, then print the items out in the order they were originally inserted into the queue. If the `chron` is false, then print the items out in reverse chronological order. For example:

```cpp
Q.enqueue(1);
Q.enqueue(2);
Q.enqueue(3);
Q.printQueue(true);   // Should print: 1 2 3
Q.printQueue(false);  // Should print: 3 2 1
```

Note: When you return from `printQueue` the original queue should be unchanged!

```cpp
void Queue::printQueue(bool chron) {
    Node *temp;
    if (chron) { // print from head to tail
        temp = head;
        while (temp) {
            cout << temp->val << "  ";
            temp = temp->next;
        }
    }
    else {  // print from tail to head
        temp = tail;
        while (temp) {
            cout << temp->val << "  ";
            temp = temp->prev;
        }
    }
}
```
c) (2 points) Give the worst case big-O running time of your implementation for each of the following operations (in terms of $n$ where $n$ is the number of elements currently in the queue). You MUST explain briefly how you got your answer.

**enqueue(int value)** – $O(1)$ – since we keep a pointer to the tail, the only thing required is a constant number of pointer manipulations (independent of the size of the queue).

**int dequeue( )** – $O(1)$ – since we keep a pointer to the head, the only thing required is a constant number of pointer manipulations (independent of the size of the queue).

d) (2 points) Give the worst case big-O running time of your implementation of **printQueue** (in terms of $n$ where $n$ is the number of elements currently in the queue). For full credit you must explain briefly how you got your answer.

$O(n)$ – printing either chronologically or reverse chronologically, we must only touch each element once.