CS 216 Exam 1 – Spring 2002

Part 1 – Closed Book

****KEY****

Name _______________________  Section _______________________

Email Address _________________  Student ID # ___________________

This exam is in two parts. Once you are finished with part 1, hand it in, and you will be given part 2. Part 1 is worth 73 points and Part 2 is worth 27 points. You will have an hour and fifty minutes total to complete both parts.

Part 1: Closed note, closed book. You are not to speak with anyone except the Instructor or a teaching assistant for any reason except an emergency during the exam. You may use calculators.

Part 2: Open book, open notes. You may not speak with anyone except the Instructor or a teaching assistant for any reason except an emergency during the exam. Calculators are allowed. You may not borrow notes or textbooks from other students.

Good Luck!

<table>
<thead>
<tr>
<th>PART</th>
<th>MAX</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Define and give an example of each:

1. a) (2 points) abstract data type

   A set of values and a set of operations on those values, e.g., a binary search tree, a stack, or a queue.

   b) (2 points) container

   A collection of elements and operations on those elements. Typical operations are empty?, full?, size(), insert, remove, and walkthrough. Containers are a type of ADT. Examples include lists, stacks, queues.

   c) (2 points) LIFO data structure

   Last-in, First-out: A structure that only allows elements to be added and removed from one “end.” Example: a stack, and its push and pop operations.

   d) (2 points) Depth of a node in a binary search tree

   The length of the unique path from the root to the node.

   Example:

   ![Depth Diagram](image)

   DEPTH = 0
   DEPTH = 1
   DEPTH = 2

2. (2 points) Give two criteria we use in deciding which data structure to use:

   - efficiency of the data structure and its operations in terms of time and space
   - whether operations supported by the data structure match the ones we need

3. (2 points) Give two things that Big-O time comparison ignores (that in reality may have an affect on running time):

   Acceptable answers:
   - Constant coefficients
   - Lower-order terms
   - Compiler optimizations
   - Hardware performance
4. [2 points] You have a computer with 512 addressable words of memory. What is the smallest number of bits required to address all words in your computer?

9 bits, since $2^9 = 512$.

5. [2 points each] In the Exam1 architecture, words are 8 bits long.
   a) Show how the number 10 would be represented in twos complement in an 8-bit word.

   \[0000\ 1010_2\]

   b) Show how -6 would be represented in twos complement in an 8-bit word.

   \[1111\ 1010_2\]

   c) Show how the result of adding 10 and -6 would be represented in an 8-bit word.

   \[0000\ 0100_2\]

6. [3 points] Write the following infix expression as a prefix expression:

\[((a + b) \times c) - d\]

\[- * + a b c d\]
7. (8 points total) Describe the running time of the following pseudocode in Big-Oh notation:

```cpp
int test(int n) {
    if (n < 10) {
        cout << "too small";
    } else {
        for (int i = 0; i < n; ++i) {
            cout << "just right";
        }
    }
    return n;
}
```

a) for (int i = 0; i < n; ++i) {
    a = test(i);
}

\[ O(n^2) \quad \text{[since test() runs in } O(n) \text{ time]} \]

b) for (int i = 0; i < 100 * n; ++i) {
    for (int j = 0; j < i; ++j) {
        cin >> b;
        a = a + b;
    }
}

\[ O(n^2) \]

c) for (int j = 0; j < 5000; ++j) {
    b = a * c;
    for (int i = 0; i < j; ++i)
        a = a + b;
    for (int i = 0; i < n; ++i)
        c = b + c;
}

\[ O(n) \]

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

\[ O(n^4) \]
8. (6 points) What is the representation of each of the following in the indicated radix? Be sure to show your work.

a) $256_{7}$ in decimal

\[ 6 + 5*7 + 2*7^2 = 6 + 35 + 98 = 139_{10} \]

b) $1230_{4}$ in hex

\[ 1230_{4} = 0110 1100_{2} = 0\text{x}6\text{C} \]

c) $9G_{18}$ in radix 10

\[ 16 + 9*18 = 178_{10} \]

9. (6 points) Consider the positive binary integer represented in two’s complement: $0000001101010101_{2}$.

a) Express this binary number in octal

\[ 0 000 001 101 010 101_{2} = 01525 \]

b. Express this binary number in hexadecimal

\[ 0000 0011 0101 0101_{2} = 0\text{x}0355 \]

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.

\[ 1111 1100 1010 1011_{2} \]
10. (4 points) Let T be a binary search tree whose is

Inorder traversal is: 0 1 2 3 4 5 6 7 8 9

Preorder traversal is: 1 0 6 2 5 3 4 7 9 8

Draw T. Please draw a circle around your final answer.

Note: the inorder traversal is actually not necessary to determine the shape of this tree since you know it’s a binary search tree. (That the inorder traversal is in numerical order and that it is a binary search tree imply one another.) Given that fact, the easiest way to draw this tree is to ignore the inorder traversal and just use the preorder traversal and the BST property. You can use the inorder traversal to check your work. Of course, for a tree that’s not a BST, you need both the inorder and preorder traversals to correctly determine the shape of the tree.
11. (3 points each) I tell you that the STL vector class is implemented as an array. For each operation give 1) How you think the operation is most likely implemented in the STL vector, 2) WHAT exactly is the worst case scenario and 3) what is the worst case Big-Oh running time of this scenario.

a) Reading an element of a vector with the [ ] operator.
   1. index into the array
   2. all cases behave the same
   3. O(1)

b) push_back(T val) – inserts a copy of val after the last element in the vector.
   1. If the array is big enough, write to array element [size] and increment the array’s size. Otherwise, copy the entire array to a larger block of memory and then do the write described above.
   2. Worse case: array is too small and must be copied
   3. O(n)

c) size() – returns the size of the vector.
   1. size is maintained internally each time elements are added to or removed from the vector, so size() just returns the value of that internal state variable.
   2. all cases behave the same
   3. O(1)

d) insert(iterator pos, T val) – inserts a copy of val prior the element in the list referred to by pos.
   1. Must move all other elements down and put val into the correct array element afterward, assuming there is enough space in the array. If the array was too small, the array must be copied to a larger block of memory before moving the elements down to create an open spot for val.
   2. Worst case: insert at [0] or the array was full and we had to copy it
   3. O(n) in either event
12. (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)?

This is not an AVL tree because the AVL property is violated at node 77. 77’s left sub-tree height is –1 and its right sub-tree height is 1, which differ by more than 1.

13. (4 points) Your friend has to write code to make sure that the parenthesis in an arithmetic expression are balanced, i.e. for every left parenthesis there is a matching right parenthesis. She wants to use a tree to do this. You argue that if all you want to do is match parentheses, then a stack will work fine.

How will you use a stack to do this?

Each time you get a “(“, push it onto the stack. Each time you get a “)”, pop one “(“ off the stack. If the stack ever underflows, then there was an unmatched “)”; if there are “(“’s left on the stack when the end of the input has been reached, there were unmatched “(“’s. If no underflows occur and the stack is empty when the end of the string is reached, the parentheses match.
14. (3 points) When we say: \( T(N) \) is \( O(f(N)) \) we mean:

a) For all values of \( N \), \( T(N) \) has a value < \( f(N) \).

b) For values of \( N \geq \) some positive constant \( n_0 \), \( T(N) \) has a value < \( f(N) \).

c) For all values of \( N \geq \) some positive constant \( n_0 \), \( T(N) \) is within a constant factor of \( f(N) \).

d) For all values of \( N \geq \) some positive constant \( n_0 \), \( T(N) = f(N) \).

(c) is the correct answer.

15. a) (2 points) Write an equation to calculate the address of \( A[i] \) in a one-dimensional array:

\[
&A[i] = \text{address of } A + i \times \text{sizeof(element in } A)\]

b) (2 points) Write an equation to calculate the address of \( A[i][j] \) in a two-dimensional array stored in row-major order:

\[
&A[i][j] = \text{address of } A + (i \times \text{number of columns + } j) \times \text{sizeof(element in } A)\]
CS 216 Exam 1 – Part II – Open book

16. (6 points) Assume we are using the 32-bit IEEE single precision floating point format as described in class and used in lab.

What positive decimal floating point number is represented by the following 32 bits?

SHOW YOUR WORK!

0101 0001 1000 1000 0000 0000 0000 0000

- **Sign bit = 0** [positive number]
- **Exponent = 101 0001 12** = 163 – 127 = 36_{10}
- **Mantissa = 000 1000 0000 0000 0000 0000_{2} = 0.0625_{10}

**Significand = 1+mantissa = 1.0625_{10}

**Value = significand \times 2^{exponent} = 1.0625_{10} \times 2^{36} = 17 \times 2^{32} = 73014444032_{10}
17. (15 points) Write a routine that given two sorted lists A and B, will produce a new list C that consists of the intersection of the elements in the two lists.

a) (5 points) Explain your general idea of how you would do this. Assume that A, B, and C are stored in STL vectors. Recall the following about STL vectors:

- `size()` – returns the number of elements in the vector
- `A[i]` – accesses the value stored in position i of A.
- `push_back(val)` – inserts a copy of val after the last element in the vector

Step through A and B while `i<A.size()` and `j<B.size()` [where i is an index for A and j is an index for B]. If `A[i] == B[j]`, `C.push_back(A[i])` and increment i and j. Otherwise, if `A[i] > B[j]`, increment j, or if `A[i] < B[j]`, increment i (thereby allowing the array that has “gotten behind” to “catch up”). Iterate.

b) (2 points) What is the worst case Big-Oh running time of your solution and why? *Be sure to state what N is.*

O(n) where n is the total number of elements in A and B.
c) (8 points) Write a C++ function to create the intersection list C. 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.

```cpp
vector<int> intersection (vector<int> A, vector<int> B) {

    vector<int> C;
    int i = 0;
    int j = 0;

    while (i < A.size() && j < B.size()) {

        if (A[i] == B[j]) {
            C.push_back(A[i]);
            i++;
            j++;
        } else if (A[i] < B[j]) {
            i++;
        } else /* A[i] > B[j] */ {
            j++;
        }
    }
    return C;
}
```
18. (6 points) A palindrome is a string that reads the same forwards as backwards. Describe how you would implement a function that, given a string, determines whether it is a palindrome. Your function can use only one stack and one queue. Your function should return true if the string is a palindrome and false otherwise.

For example:

\[ \text{palindrome(“abba”) } \rightarrow \text{true} \]

\[ \text{palindrome(“bbba”) } \rightarrow \text{false} \]

\[ \text{Read each letter from the string one at a time, pushing it onto the stack and also enqueueing it, keeping track of the size of the string as you go along. Once that’s done, divide the size by two and dequeue that many items from the queue. For each item dequeued, also pop the element off the top of the stack. If the two are ever unequal, return false. If you reach size/2 elements dequeued and popped and they all matched, return true.} \]
18. (6 points) Describe how you would implement a stack using a single queue. Specifically describe how you would implement push and pop.

**Push: Enqueue**

**Pop: Iteratively dequeue items from the front of the queue and immediately re-enqueue until the element that was originally last in the queue is now at the front. Dequeue that item and return it.** The only question is how to know when you’ve reached the last item in the queue; here you have a few options. If you know how many items are in the queue, you can just count as you go along.

If you don’t know how many items are in the queue, you could start the pop process by enqueuing some “magic token” (say, for example, a $ dollar sign), and watching for that magic token each time you dequeue an item. Making this work requires that you store the last two items that were dequeued, rather than the single most recent as in the known-size option above. The full algorithm would be this: iteratively dequeue items from the queue, keeping track of the last two that were dequeued. If the most recent one was $, return the second most recent. Otherwise, re-enqueue the second most recent item; the most recent then becomes the second most recent, and iterate.

Pledge for the open and closed book exam: On my honor, I have neither given nor received unauthorized aid on this exam.

_________________________
Sign your name