# EE 312 Spring ’13 Exam 2 NAME:

37 Points

1. (6 pts) I have two data structures, a Binary Search Tree and a Vector. I want to determine if the two data structures contain the same set of values. I am considering two methods. Please assume that I know that both data structures contain exactly N values and neither data structure contains duplicate values.
   * (2 pts) Method A. In this method, I look at each element in the Vector (from the first to the last) and I search for that element in the Binary Search Tree. I use the ordinary “find” function for the Binary Search Tree. If we assume the elements in the Vector are in a random order and the Binary Search Tree is reasonably balanced (height is O(log N)), then what is the worst case time complexity to determine if the Vector and the BST contain the same values?
   * (2 pts) Method B. In this method, I am first going to sort the elements in the Vector (using MergeSort or some other reasonably fast sorting function). Then I will compare the elements in the two data structures one-by-one, starting with the smallest value in each data structure. If the smallest values are the same, I compare the second-smallest values in the data structures, and so forth until I’ve compared all the values. If we assume that the elements in the Vector are initially in random order (i.e., before I sort them), and if the Binary Search Tree is reasonably balanced, then what is the worst case time complexity to determine if the Vector and BST contain the same values?
   * (1 pts) Note that for parts (a) and (b) I made some assumptions about the BST’s structure and the Vector’s values. What if I can’t make any such assumptions, and I needed to consider the truly worst case. Under the absolute worst case assumptions, what is the time complexity of Method A?
   * (1 pts) Again, under the absolute worst case assumptions, what is the time complexity of Method B?
2. (6 pts) Using the hash function shown below and assuming a Hash Table implementation similar to what we did in class:

uint32\_t hash(char\* s) {

uint32\_t h = \*s;

while (\*s != 0) {

if (\*s > h) { h = \*s; }

s += 1;

}

return h;

}

* + (2pts) What is the worst-case time complexity to create a Hash Table that has a load-factor of at most 0.8 and contains N random strings (i.e., how long does it take to insert N strings into the table)?
  + (2 pts) Again, assuming the hash function above, and assuming the hash table contains N random strings, what is the worst-case time complexity to answer the question, “is the string ‘Craig’ stored in the hash table?” (i.e., what is the worst-case time complexity for the find function?)
  + (2 pts) Which of the following could reduce the number of collisions for the hash table (circle all options that could reduce the number of collisions in the hash table above)
    1. Use a different hash function
    2. Use linked lists to implement the Chains
    3. Decrease the load factor
    4. Increase the load factor

1. (8 pts) In the Vector implementation below, I’ve made a few mistakes in my use of C++ constructs. I want you to fix those mistakes. I’ve enumerated the mistakes, (a), (b), (c), (d) below. What I want you to do is add code (or cross out code) to correct each of these mistakes. Each time you correct one of these mistakes, label your correction with (a), (b), etc. so I know which mistake you think you’re correcting. There may be other mistakes too, just fix these four. Two points for each mistake.
   * The parameter to the copy constructor is incorrect. Fix it.
   * I did not write a destructor (write one).
   * Everything in the class is private. Add more public/private declarations so that the class is well designed.
   * The operator[] function has the wrong return type, and consequently it is not possible for my clients to change elements in the Vector after the elements are first inserted.

struct Vector {

private:

int32\_t\* data;

uint32\_t capacity;

uint32\_t length;

Vector(void) {capacity = 10; data = new int[10]; length = 0;}

Vector(Vector that) {

capacity = length = that.length;

data = new int[capacity];

for (int k = 0; k < length; k += 1) {

data[k] = that.data[k];

}

}

int32\_t operator[](uint32\_t k) { return data[k]; }

};

1. (2 pts) In a Vector, we use a technique called “amortized doubling”. This technique creates a tradeoff between speed and memory consumption. Specifically, because we double the storage, we could end up wasting almost ½ of the memory in the data structure with capacity that we never use. When the Vector is small (e.g., 100 elements), wasting half the storage isn’t such a big deal, but for large Vectors (millions of elements), that’s a lot of memory! I’m considering improving upon this tradeoff by always increasing the capacity by 100 (instead of doubling it) whenever the Vector becomes full. That way, I never waste more than 100 elements, no matter how large the Vector gets. I call this technique, “amortizing hundreds”. Is amortizing hundreds a good technique to use for large Vectors? Explain the tradeoff(s) that result.
2. (2 pts) Chaining is a technique used to handle collisions that occur in a Hash Table. In the Hash Table implementation from class, we elected to implement our chains using Linked Lists. Linked Lists are certainly simple, and they provide O(1) insert cost (but O(N) search time). As an alternative, we could have made each Chain a binary search tree. In this case, the time to insert elements in a chain would increase to O(log N), but the search time would decrease to O(log N). N in this context represents the number of elements in a single chain (i.e., the number of collisions to the same table index). Keeping in mind that search operations are much more common than insert operations, explain why Hash Tables are rarely (i.e., never in practice) implemented using Chains that are Binary Search Trees. Why isn’t the O(log N) search time worth the extra complexity of a Binary Search Tree implementation compared to a Linked List?
3. (8 pts) Consider the following C++ code and indicate what the output is for each of the following short programs. If the program does not produce any output, write “NOTHING”. Do not leave any blank. Some statements may print more than one thing. For each statement write on the line provided the output produced by that statement.

struct Foo {

int x;

Foo(void) { this->x = 10; printf(“basic constructor\n”); }

Foo(int x) { this->x = x; printf(“int constructor\n”); }

Foo(Foo& that) {

this->x = that.x;

printf(“copy constructor\n”);

}

~Foo(void) { printf(“destructor for %d\n”, this->x); }

};

1. int main(void) { \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Foo f;

}

1. int main(void) { \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Foo\* f;

}

1. int main(void) { \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Foo\* f = new Foo(42);

}

1. void doit(Foo g) { \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

int main(void) {

Foo f;

doit(f);

}

1. (2 pts) Draw the binary search tree that is produced when the values 10, 20, 15, 5, 13 are inserted in precisely that order.
2. (2 pts) Show the state of the following binary search tree after the value 30 has been removed (assume no changes were made, other than removing 30, draw the resulting tree to the right).

20

10

5

15

30

25

26

40

1. (1 pts) A standard feature on a smartphone is to organize a “contacts list”. This list contains names, telephone numbers, email addresses, etc. Assume that we want to implement a typical contact list. We want to be able to do all of the following functions
   * Add a new contact to the list
   * Delete a contact from the list
   * Search for a contact based on the name of the person
   * Display the contacts in alphabetical order (by name)
   * Provide an interactive search where, as the user types in a person’s name, we display all the names in the contact list that start with those letters. So, for example as the user has typed “C” in the search box, we display all the contacts who’s name start with C, once the user has typed “Cr”, we display only those contacts who’s names start with “Cr” and so on.

What data structure should I use to implement the contact list? Explain (very briefly, one complete sentence would do nicely).