**CSCI 2270**

**Review for final exam. All 3 previous review sheets are also important.**

**Given the graph with vertices A, B, C, D, E, and F, and the following edges:**

**A: F, B**

**B: A, C C: B, D**

**D: C, E**

**E: D, F**

**F: E, A**

1. **Draw the graph as vertices and edges.**

1. **In a depth first search of the above graph, what vertices will it pop off the stack in a search starting at A and ending at F?**
   1. Using Depth-First Search, the order of nodes searched from A -> F would be:

A -> F

1. **In a breadth first search of the above graph, what vertices will it pop off the queue in a search starting at C and ending at D?** 
   1. Using Breadth-First Search, the order of nodes searched would be:

C -> B -> D;

1. **What’s better about breadth first search than depth first search?**
   1. On average, Breadth-First Search will find a path to the target destination in the fewest number of stops. Depth

1. **Which takes longer, breadth first or depth first search?** 
   1. This ultimately depends on the type and organization of data being searched.

1. **Given a hash table of size 17 (this tells you the hash function to use) that uses open addressing plus a search for the next open slot, add the pairs:**

**138, “Frodo”**

**241, “Pippin”**

**070, “Merry”**

**104, “Tom Bombadil”**

**106, “Dick Cheney”**

**Draw the final table when you are done.**

**What problem is getting worse here?**

**How would your answer change if you used double hashing with a second hash function of modulo 5?**

**How would your answer change if you used chained hashing?**

1. **Explain, in simple English, how a buffer overrun hack works.** 
   1. A buffer overrun works by overwriting the value in memory where the program stores the address of where to return after executing the current method. A buffer overrun hack replaces the current method’s return address value with the address of code which would not have otherwise been run.

1. **What is the difference between a deep copy and a shallow copy? How can you write a test to tell which one you have? How do pointers and shallow copies relate to each other?**
   1. A deep copy is and identical twin of an object. All of their member values are equal however each object is entirely distinct from the other. A shallow copy involves two pointers which point at the same object. To determine whether two pointers are deep or shallow copies, you can mutate one (add to, modify member variable, remove, etc.) and check to see if the object referenced by the other pointer has been affected by this change. If the object referenced by the second pointer has changed, you’re looking at a shallow copy. If you mutate one object and the second remains unchanged, you’ve got a deep copy on your hands.

1. **How can you tell if 2 heaps in array form have all of the same elements?** 
   1. Check to see if the heaps have the same size, then check the root element for equality, remove it, re-heapify, then repeat until the end of the heaps are reached or until two roots are found to be different

1. **Why do big\_numbers benefit from a trim() function? When is such a function useful in HW2?** 
   1. big\_numbers benefit from a trim() function’s ability to remove zeros that precede any digits of value. This is handy because there are several cases where leading zeros can be introduced while handling big\_numbers , such as subtraction (e.g. 999-990 would result in 009) and the string constructor (PEBCAK ERR).

1. **If we didn’t write big\_number’s operator =, but we used the default version that C++ gives us instead, will we leak memory?**

1. **Give me an example of the scenario in question 11 causing a crash at runtime.**

1. **Why do we have the rule that heaps must be complete trees?**

1. **Given the array 1 4 6 8 3 2 7 5 9 0, show me how quicksort could degrade to quadratic performance in the first 3 partition steps.** 
   1. **First pass – arr = {1,4,6,8,3,2,7,5,9,0};**
      1. Chosen pivot –> arr[0];
      2. Partitions: {1} , {4,6,8,3,2,7,5,9,0};
   2. **Second pass – sub-partition = {4,6,8,3,2,7,5,9,0};**
      1. Chosen pivot -> arr[8];
      2. Partitions: {4,6,8,3,2,7,5,9} , {0};
   3. **Third pass – sub-partition = {4,6,8,3,2,7,5,9};**
      1. Chosen pivot -> arr [0];
      2. Partitions: {4} , {6,8,3,2,7,5,9}
2. **Given a load factor of 25%, what is the general performance (in terms of expected slots checked) of a doubly-hashed hash table?**

1. **When can a load factor exceed 100%? Why does this happen?** 
   1. Using a value of a linked list or array associated to a key give you the ability to store a load factor greater than 100%