*COMP 410 Spring 2016*

**Final Exam**

This exam is closed book, notes, calculators, cell phones, classmates, everything but your own brain. You have 180 minutes to complete the exam but a well prepared student will more likely need 120 mins. Do all your work on these exam pages. Please sign here (and print under it) pledging that the work you submit is your own:

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name (print): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem 1 (4%)**

For this problem consider the di-graph to the right:

This question is about node search order. When you have a choice of next nodes, use *alphabetic order* of node name to resolve the choice (if you can go to C or E next, choose C first for example).

1. **Starting with A, write the order in which the nodes are visited in a depth-first search:**

A B F G K E H C D use a stack to manage depth first

1. **Starting with A, write the order in which the nodes are visited in a breadth-first search:**

A B C F H D G E K use a queue to manage breadth first

**Problem 2 (10%)**

Below are two hash tables. The left one uses linear probing to resolve collisions. The right one will hash into lists to resolvecollisions.

1. **First compute the hash value for each word and write it below the word in the blank.**

 maple elm oak birch plum myrtle ash fir holly beech

hash: \_\_3\_\_ \_\_0\_ \_11\_ \_5\_\_\_ \_8\_\_\_ \_\_5\_\_ \_8\_\_ \_5\_\_ \_12\_ \_\_5\_\_

Use this hash function: take the position in the alphabet of the third letter, and mod that by table size. For example, the word *roster,* third letter is S, which has alphabet position 19. To get the final hash, mod 19 by table size.

*an alphabet, with ordinal positions, for your viewing pleasure*

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26

***1 2 3 4 5 6 7 8 9 0 11 12 0 1 2 3 4 5 6 7 8 9 10 11 12 0 mod 13***

1. **Then fill in each table with the hashed input items in the order given left to right**

 linear hash into lists

 probing

 0: \_elm\_\_\_\_\_\_ 0: \_\_\_--> elm

 1: \_\_\_\_\_\_\_\_\_\_ 1: \_\_\_\_\_

 2: \_\_\_\_\_\_\_\_\_\_ 2: \_\_\_\_\_

 3: \_maple\_\_\_\_ 3: \_\_\_--> maple

 4: \_\_\_\_\_\_\_\_\_\_ 4: \_\_\_\_\_

 5: \_birch\_\_\_\_ 5: \_\_\_--> birch 🡪 myrtle 🡪 fir 🡪 beech

 6: \_myrtle\_\_\_ 6: \_\_\_\_\_

 7: \_fir\_\_\_\_\_\_ 7: \_\_\_\_\_

 8: \_plum\_\_\_\_\_ 8: \_\_\_--> plum 🡪 ash

 9: \_ash\_\_\_\_\_\_ 9: \_\_\_\_\_

 10: \_beech\_\_\_\_ 10: \_\_\_\_\_

 11: \_oak\_\_\_\_\_\_ 11: \_\_\_--> oak

 12: \_holly\_\_\_\_ 12: \_\_\_--> holly

1. \_\_T\_\_\_ **T or F: The linear probing table (left) is over full and needs to be extended**
2. **\_\_F \_\_\_ T or F:  for the lists table (right) is smaller than  for the probing table (left).**

**Problem 3 (3%)**

What is the “Big Oh” time complexity of this program when run on an integer N ? Be as accurate as you can. For example, O( 5 \* N^5 ) is more accurate that O(N^5) if the 5 multiplier applies.

 **function foo ( N ) {**

 **if (N <= 1) return 1; answer: \_\_\_\_O(N)\_\_\_\_\_\_\_\_\_\_**

 **else {**

 **return N + foo(N-1);**

 **}**

 **}**

**Problem 4 (3%)**

What is the “Big Oh” time complexity of this program when run on an integer N ? Be as accurate as you can. For example, O( 5 \* N^5 ) is more accurate that O(N^5) if the 5 multiplier applies

 **function foo ( N ) {**

 **if (N <= 1) return 1; answer: \_\_\_O(2^N)\_\_\_\_\_\_\_\_\_**

 **else {**

 **return foo(N-1) - foo(N-2);**

 **}**

 **}**

**Problem 5 (3%)**

What is the “Big Oh” time complexity of this program when run on an integer N ? Be as accurate as you can. For example, O( 5 \* N^5 ) is more accurate that O(N^5) if the 5 multiplier applies.

 **function foo ( N ) {**

 **if (N <= 1) return 1; answer: \_\_\_O(2N) or O(N)\_\_\_\_\_\_**

 **else {**

 **return foo( foo (N-1) );**

 **}**

 **}**

**Problem 6 (10%)**

 For this question, you will select from this list of items:

(A) heap sort (G) bucket sort

(B) no efficient solution is known (H) is efficiently solvable

(C) bogo sort (I) dense graph

(D) can be solved in constant time (J) merge sort

(E) has no efficient solution (K) no solution is possible

(F) sparse graph (L) bubble sort

For each of the following definitions, select the item above that best matches (put the corresponding letter in the blank); an item above may be used more than once:

1. \_\_J\_\_\_\_ stable sort that is O(N log N) worst case
2. \_\_A\_\_\_\_ unstable sort that is O(N log N) worst case
3. \_\_G\_\_\_\_ sort algorithm that is O(N) worst case
4. \_\_B\_\_\_\_ traveling salesman problem
5. \_\_B\_\_\_\_ find a Hamiltonian path in a graph
6. \_\_I\_\_\_\_ complete graph with 12 vertices
7. \_\_H\_\_\_\_ find a minimum spanning tree for a graph
8. \_\_K\_\_\_\_ halting problem
9. \_\_B\_\_\_\_ graph isomorphism problem
10. \_\_F\_\_\_\_ graph formed from the circuits in a VLSI design (millions of

 transistors as nodes, edges as wire connections between them)

**Problem 7 (4%)**

2

**True or False:** Consider the graph G to the right, **G:**

1

then answer T or F (true or false) for each of these:

1

1

3

1. \_\_T\_\_ G has exactly one minimum spanning tree
2. \_\_F\_\_ G has fewer than 4 spanning trees

3

1. \_\_T\_\_ G is a planar graph

2

1

1

1

1. \_\_F\_\_ G is a complete graph

**Problem 8 (8%)**

Consider the following 2 connected undirected graphs:



1. **In (g1) is there an Euler path? NO**

 How do you know, using Euler's theorems? >2 odd vertices

 If there is one, show one:

1. **In (g1) is there an Euler circuit? NO**

 How do you know, using Euler's theorems? >0 odd vertices

 If there is one, show one:

1. **In (g2) is there an Euler path? YES**

 How do you know, using Euler's theorems? Exactly two odd vertices, they are E and B

 If there is one, show one: E A B D G C E F C B F A G B

1. **In (g2) is there an Euler circuit? NO**

 How do you know, using Euler's theorems? >0 odd vertices

 If there is one, show one:

**Problem 9 (10%)**

**True or False:**

1. \_F\_\_\_ A graph that is already a tree has no minimum spanning tree
2. \_F\_\_\_ A complete graph is always planar
3. \_T\_\_\_ If the number of nodes in a tree is an exact power of 2, then that tree is a bi-partite graph.
4. \_T\_\_\_ If the number of nodes in a tree is odd then that tree is a bi-partite graph.
5. \_F\_\_\_ Every graph that is a tree has no Euler path.
6. \_F\_\_\_ Careful programming can produce a hash function that will work well for all data types.
7. \_F\_\_\_ Carefully programmed, quicksort is a stable sort algorithm.
8. \_T\_\_\_ Carefully programmed, insertion sort is a stable sort algorithm.
9. \_F\_\_\_\_ The splay function in a splay tree always shortens the longest path by at least 1
10. \_F\_\_\_\_ In a balanced AVL tree, the shortest path and the longest path differ by at most 1

**Problem 10 (10%)**

Consider the unique queue, or UQUE. Let the operations be **new, add, peek, rest, size:**

 **add** puts an item on the tail of the queue

 **peek** returns the head item

 **rest** returns the queue that remains when the head item is taken off

 **size**  tells how many items are in the queue

In a UQUE, we add elements to the queue at the back, and we remove them from the front like a normal queue. However, we do not allow two or more elements in the queue to have the same value. If we do (for example) add(Q,5) and 5 is already in Q, then nothing is added, and the queue remains the same length as before the add. If 5 is not in Q, the length of Q grows by one, and 5 is at the back of the queue.

Describe the data structures you would use to implement the UQUE so that the **“add” operation can be** **done in O(1) worst case time**. Explain how it all works, and justify your claims. Describe the optimal space used by your solution in terms of N (size of the UQUE).

Need a normal Queue (implemented as linked cells) and a hashmap, or hashset. When we do an add (enque) we hash the value first into the hashmap. If the item is already in the hashmap ( take O(1) time on average, but we use it as worst case practically) We do not enque it into the Queue. If it is not in the hashmap, we put it there and do an enque on the item.

Space used is O(N) for the queue, and O(2N) for the hashmap. Total is O(3N) or just O(N).

This problem uses the average case O(1) for hashmap as the worst case… for practical use O(1) is almost always how a hashmap behaves… the worst case O(1) is very unlikely even though theoretically possible

**Problem 11 (9%)**

Consider this adjacency matrix representation for a directed weighted graph (with 0 used for no edge):

 **A B C D E F G**

 **A** 0 4 2 0 0 0 3 too hard to draw in Word

 **B** 0 0 0 3 0 4 2

 **C** 0 1 0 0 0 0 1

 **D**  0 0 0 0 2 1 0

 **E** 0 0 0 0 0 0 0

 **F** 0 0 0 0 2 0 0

 **G**  0 0 0 1 0 2 0

1. **Provide a drawing for this graph** 🡺
2. **Is the graph acyclic? \_\_YES\_\_\_\_**

  **If so, give a topological sort of the vertices in the graph (ignore the edge weights):**

 A C B G D F E

  **If not, identify a cycle:**

**Problem 12 (6%)**



1. **Does this graph have a Hamiltonian path? \_\_YES\_\_\_\_\_**
2. If so, show one: A B D G H X C F E
3. If not, why not?
4. **Does this graph have a Hamiltonian circuit? \_\_NO\_\_\_\_\_**
5. If so show one:
6. If not, why not? The edge H – X is a bridge, and if there is a bridge there is no Hamiltonian circuit

**Problem 13 (10%)**

Consider the undirected graph below. Find a Minimum Spanning Tree using Kruskal’s Algorithm. Draw your final MST in the box to the right of the original graph **using the same node layout** so that it is easy to compare the two graphs. Show your work below for each step of the algorithm.

 **Original graph: MST:**

3

B

A

4

2

3

D

1

4

C

E

2

3

1

1

F

G

**Hard to draw just run the algorithm follow the book or slides**

**MST has edge lengths of 12**

**Problem 14 (10%)**

Consider the construction and use of a minimum binary heap. Show a data sequence containing duplicate values that will show why heap sort is unstable. This means find a data value sequence that will build a heap such that when the min values are taken out, the duplicate values come out in a different order than they went into the heap. The sequence does not have to be long, it just has to have at least 2 duplicate values. To distinguish the duplicates dot notation… i.e., if we have two 8 values, make one of them 8.a and the other 8.b.

Show your input sequence, the construction of the original heap, and the removal of min items enough to show the duplicates coming out in reverse order. Show proper delMin behavior.

**Answer:**

**Input Sequence: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Work:**

***Leaving this for you to work on… I gave it in class long ago.***