COMP 550 Algorithms and Analysis
Spring 2015
Mid-Term 3 (Sample)

60 min

Name _____

PID _____

Honor Pledge:

I have not given nor received unauthorized assistance in completing this exam.

Signature _____

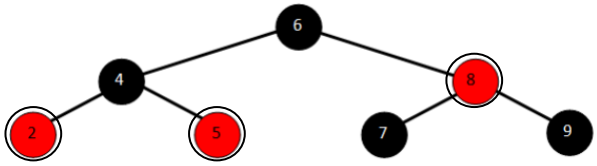
Note:

(1) All “lg”s are based 2 if unspecified.

(2) Points of subproblems are evenly distributed within each problem unless specified.

(3) Your score will be put on the second page – to protect your privacy.

1. (20') You are given the following Red-Black Tree, where red nodes are double circled. Draw the tree after deleting "9", "8", "7", respectively. (Hint: Showing it step by step is encouraged, which may earn you more points when your final tree looks wrong.)



2. (20') Sorting with search trees. Given a set of **n** numbers, we could sort them by the following two steps: (i) Insert them one by one in the appearance order into a search tree, and (ii) Perform an in-order tree walk to print all elements in sorted order.

```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2      INORDER-TREE-WALK(x.left)
3      print x.key
4      INORDER-TREE-WALK(x.right)
```

(a) (4') What's the time complexity of in-order tree walk (asymptotically, as a function of **n**)?

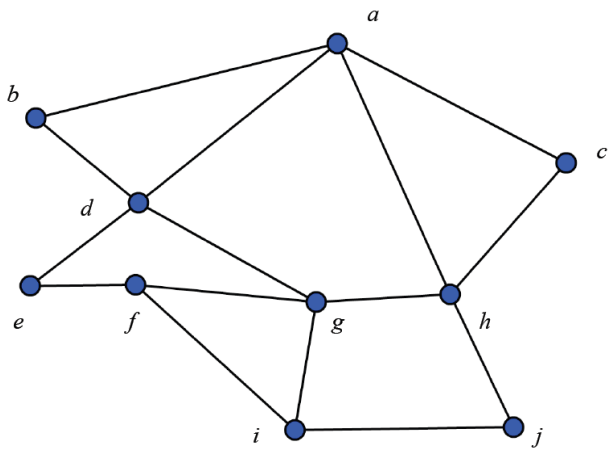
(b) (5') What's the worst case time complexity of BST sorting (asymptotically, as a function of **n**)? Briefly explain why.

(c) (11') What restrictions (rules) do we need to follow in order to make the BST sorting algorithm *stable* (Hint: specifically, during each insertion)? Briefly prove your claim. Note: ***stable*** sorting algorithms maintain the relative order of records with equal keys (i.e., values).

3. (14') Perform a breadth-first search to the graph below, with vertex **a** as the source.

(a) Draw the breadth-first search tree.

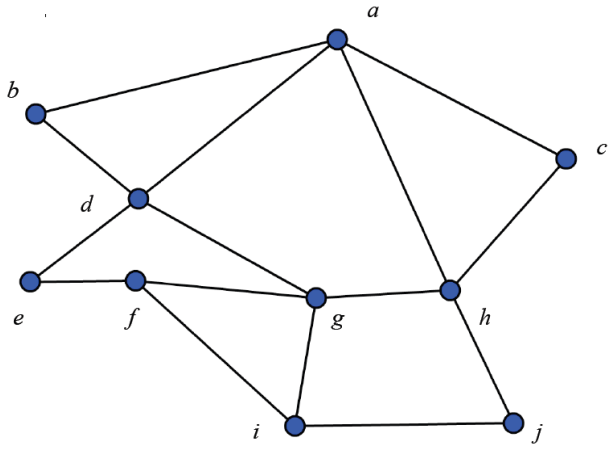
(b) Give the in-queue order of all nodes.



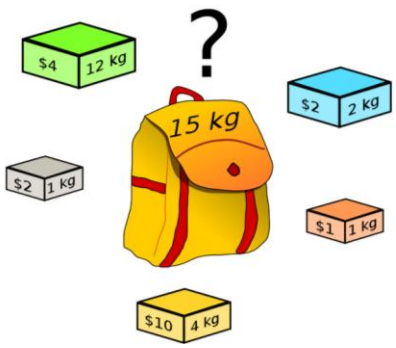
4. (16') Perform a depth-first search to the graph below, with vertex **a** as the source. Assume that the inner for loop of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically.

(a) Draw the depth-first search tree.

(b) Show the discovery and finishing times for each vertex



5. (30') 0/1 Knapsack Problem.



Given n items, pack the knapsack to get the *maximum* total value. Each item i has some weight w_i>0 and some value b_i>0. Total weight that we can carry is no more than some fixed number W (all w_i , b_i and W are integer values).

The Problem, in other words, is to find the maximum $\sum_{i \in T} b_i$, subject to $\sum_{i \in T} w_i$ <= W.

(a) (5') One way is to explore all possible combinations, and select the one with maximum total value that satisfies the constraint (total weight does not exceed W). What's the time complexity of the algorithm?

(b) (10') Now we use Dynamic Programming to solve the problem. Assume all items are labeled from 1 to n. Define S_k = {items labeled 1, 2, ..., k}. Form a recursion of B[k,w], which is the maximum value draw from subset S_k, where w represents the exact weight for the optimal subset of items from 1 to k. (Hint: when w < w_k, the k_{th} item can't be part of the optimal solution; while when w >= w_k, we need to consider both cases that either the k_{th} item is or is not part of the optimal solution.)

(c) (10') Run your algorithm on the following data: n = 4 (# of elements), W = 5 (max weight), Elements (weight, benefit): (2,3), (3,4), (4,5), (5,6)

(d) (5') What is the time complexity of your algorithm?