University of Illinois at Urbana-Champaign
Department of Computer Science

Final Examination

CS 225 Data Structures and Software Principles
Fall 2009
7-10p, Tuesday, December 15

Name:
NetID:
Lab Section (Day/Time):

- This is a **closed book** and **closed notes** exam. No electronic aids are allowed.
- You should have 9 problems total on 19 pages. The last two sheets are scratch paper; you may detach them while taking the exam, but must turn them in with the exam when you leave.
- Unless the problem specifically says otherwise, (1) assume the code compiles, and thus any compiler error is an exam typo (though hopefully there are not any typos), (2) assume you are NOT allowed to write any helper methods to help solve the problem, nor are you allowed to use additional arrays, lists, or other collection data structures unless we have said you can, and (3) assume the best possible design of a particular implementation is being used.
- Please put your name at the top of each page.

Problem	Points	Score	Grader
1	20		
2	10		
3	20		
4	15		
5	20		
6	20		
7	20		
8	15		
9	20		
Total	160		

1. **[Choices, Choices! – 20 points].**

MC1 (2pts)

Suppose your goal is determine whether or not a graph contains a vertex that is connected to no other vertices. How long does the best possible algorithm take if the graph is implemented using adjacency lists? an adjacency matrix? (As always, assume there are n vertices and m edges.)

(a) $O(1)$ for lists and $O(n)$ for the matrix.
(b) $O(deg(v))$ for lists and $O(n)$ for the matrix.
(c) $O(n)$ for lists and $O(n^2)$ for the matrix.
(d) $O(m)$ for lists and $O(n^2)$ for the matrix.
(e) None of these answers is correct.

MC2 (2pts)

When should a pointer parameter `p` be a reference parameter? (That is, when would it be more appropriate for a parameter list to be `(myType * & p)` rather than `(myType * p)`?)

(a) When the function changes `p`, and you want the change to affect the actual pointer argument.
(b) When the function changes `p`, and you do NOT want the change to affect the actual pointer argument.
(c) When the function changes `*p`, and you want the change to affect the object that is pointed at.
(d) When the function changes `*p`, and you do NOT want the change to affect the object that is pointed at.
(e) When the pointer points to a large object.

MC3 (2pts)

Suppose we have implemented the `Stack` ADT as a singly-linked-list with head and tail pointers and no sentinels. Which of the following best describe the running times for the functions `push` and `pop`, assuming there are $O(n)$ items in the list, and that the bottom of the stack is at the head of the list (all pushing and popping occurs at the tail)?

(a) $O(1)$ for both functions.
(b) $O(n)$ for both functions.
(c) $O(1)$ for `push` and $O(n)$ for `pop`.
(d) $O(n)$ for `push` and $O(1)$ for `pop`.
(e) None of these is the correct choice.

MC4 (2pts)

Which of the following statements is true for a B-tree of order m containing n items?

(i) The height of the B-tree is $O(\log_m n)$ and this bounds the total number of disk seeks.

(ii) A node contains a maximum of $m - 1$ keys, and this bounds the number of disk seeks at each level of the tree.

(iii) Every Binary Search Tree is also an order 2 B-Tree.

Make one of the following choices.

(a) Only item (i) is true.

(b) Only item (ii) is true.

(c) Only item (iii) is true.

(d) Two of the above choices are true.

(e) None of choices (i), (ii), or (iii) are true.

MC5 (2pts)

In our *uptree* disjoint sets implementation, suppose we employ Union-by-Size and no path compression. The running times of the relevant functions are:

(a) $O(1)$ for `setUnion`, and $O(\log n)$ for `Find`.

(b) $O(n)$ for `setUnion`, and $O(\log^* n)$ for `Find`.

(c) $O(\log n)$ for `setUnion`, and $O(\log n)$ for `Find`.

(d) $O(1)$ for `setUnion`, and $O(\log^* n)$ for `Find`.

(e) None of the above is correct.

MC6 (2pts)

Suppose we run depth first search on a connected, undirected graph with n vertices and $5n+2$ edges. How many edges will be labelled "back" edges?

(a) $3n + 4$

(b) $4n + 3$

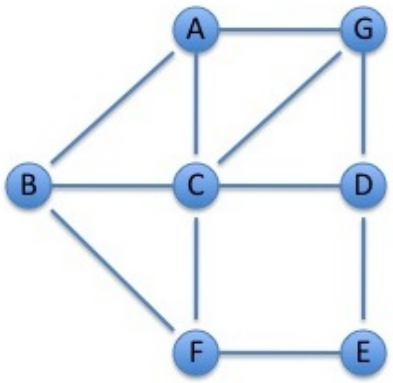
(c) $5n + 2$

(d) $6n + 1$

(e) The answer cannot be determined from the information given.

MC7 (2pts)

Depth first search, when run on the following graph, beginning at node A, classifies edge (c, f) as _____. Assume that the edge iterators are set up to process adjacent vertices in the (left to right) order given in this adjacency list representation of the graph.



A	B	C	G		
B	A	C	F		
C	A	B	D	F	G
D	C	E	G		
E	D	F			
F	B	C	E		
G	A	C	D		

(a) "back"

(b) "cross"

(c) "discovery"

(d) "predecessor"

(e) None of these is the correct choice.

MC8 (2pts)

Suppose we implement Kruskal's algorithm using a sorted array as our priority queue and an adjacency matrix graph. What is the tightest worst case running time of the algorithm? (As usual, $|V| = n$, $|E| = m$, and you may assume disjoint set functions run in constant time.)

(a) $O(n^2)$

(b) $O(n + m)$

(c) $O(m \log n)$

(d) $O(n^2 + m \log n)$

(e) None of the above accurately describes the running time of Kruskal's algorithm.

MC9 (2pts)

Suppose we implement an algorithm to determine whether or not a simple undirected graph is connected using a modification of a graph algorithm we saw in class. What is the tightest worst case asymptotic running time of the best algorithm to do this?

(a) $O(n^2)$

(b) $O(n+m)$

(c) $O(n \log n + m \log n)$

(d) $O(n)$

MC10 (2pts)

Which of the following array-based representations of a forest of up-trees could not possibly result from a sequence of `union` and `find` operations if union-by-size and path compression are employed?

(a)

0	1	2	3	4	5	6	7
-1	-1	-1	-1	-1	-1	-1	-1

(b)

0	1	2	3	4	5	6	7
-8	0	0	0	0	0	0	0

(c)

0	1	2	3	4	5	6	7
-2	0	-2	2	-2	4	-2	6

(d)

0	1	2	3	4	5	6	7
-8	0	1	2	3	4	5	6

(e) All of these are valid forests of up-trees.

2. **[A Little C++ – 10 points].**

(a) (5 points) Complete the following (silly) function implementation so that no memory is leaked.

```
void facebook() {
    int linkedin = 7;
    int * hi5 = &linkedin;
    int * orkut = new int[4];
    int * friendster = orkut;
    int ** bebo = new int * [3];
    bebo[0] = &friendster[3];
    bebo[1] = new int;
    bebo[2] = bebo[0];

    // add code to free each piece of dynamically
    // allocated memory exactly once









}
```

(b) (3 points) Write the line of C++ code that declares a variable named `album` whose type is a vector of dynamic arrays of BMPs.

(c) (2 points) Briefly describe the two instances when the destructor is called.

-

-

3. **[1D-Search – 20 points].**

In this problem you will describe a C++ class that supports nearest-neighbor search among 1-dimensional points consisting of a single real value. The abstract data type corresponding to the class definition of `OneDSearch`:

- `OneDSearch()`: creates an empty 1-dimensional search structure.
- `insert(double x)`: inserts the real number `x` into the structure.
- `query(double y)`: returns the real number in the data structure that is closest to `y`. If the data structure is empty, return a large number constant called `BIG`.

Implementation notes: Your data structure should perform each operation in worst-case $O(\log n)$ time. Your answers will be graded on correctness, efficiency, clarity, and succinctness.

(a) (7 points) Write the entire public portion of the class definition for `OneDSearch`, paying particular attention to `const` correctness and, depending on your implementation, responsible memory management.

```
class OneDSearch {
public:




private:  // you don't have to write this
};
```

(b) (3 points) The best implementation of this class most nearly resembles which of the following data structures from our course?

BST AVL Tree Hash Table SkipList Heap KD-Tree DFS BFS

(c) (7 points) Write the C++ code for public member function `query(double y)`. Be sure to comment your code so we know what you're doing. You may write a private helper function, if you'd like.

(d) (3 points) Suppose you want to add a function `rangeQuery(int a, int b)` that returns a list of all the values in the structure between `a` and `b` (inclusive). The tightest, worst-case running time of `rangeQuery` in terms of n, the number of values in the structure is...

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$

4. **[Hashing – 15 points].**

(a) (6 points) Suppose the following keys, k, are inserted in *some* order into an initially empty linear probing hash table of size 7, using the following table of hash values ($h(k)$):

k	$h(k)$
A	3
B	1
C	4
D	1
E	5
F	2
G	5

Circle the table(s) below that could be the result of a sequence of insertions of the data above.

I.

0	1	2	3	4	5	6
G	B	D	F	A	C	E

II.

0	1	2	3	4	5	6
B	G	D	F	A	C	E

III.

0	1	2	3	4	5	6
E	G	F	A	B	C	D

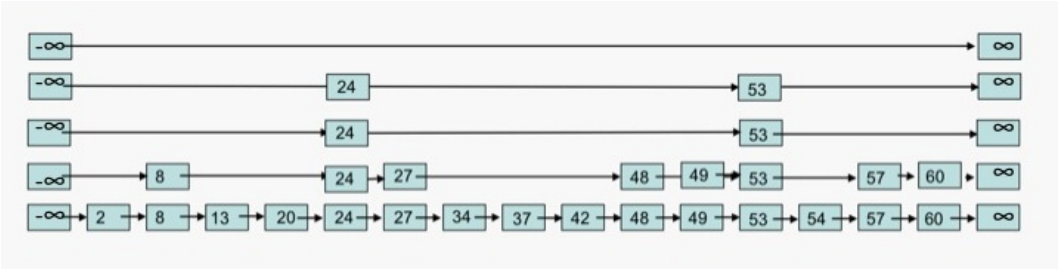
(b) (9 points) Suppose a hash table of size N handles collisions by separate chaining, and that it employs periodic resizing whenever necessary. In order to argue that lookup times are constant time operations on average, we have to make some assumptions. List those assumptions here:

- (assumption 1 about the hash function)
- (assumption 2 about the hash function)
- (assumption 3 about the hash function)
- (assumption 1 about the resizing scheme)
- (assumption 2 about the resizing scheme)

5. **[Skip Lists – 20 points].**

(a) (5 points) In class we examined a particular scheme for creating the "towers" of a skip list using coin flips. What is the probability we see a sequence of coin flips corresponding to a tower containing exactly 3 pointers?

(b) (5 points) Suppose we want to insert the key `35` into the structure below, and we've flipped coins to determine that the tower containing `35` should have exactly 3 pointers. In the diagram, neatly mark the nodes that are compared when the function call `insert(35)` is made. Please mark each node in every level at which a comparison takes place (so that some nodes will have multiple comparisons). In addition, place a small `x` on the pointers in the original structure that have to be reassigned in order to insert the new node.



(c) (3 points) Suppose you are flipping coins to create a tower and that each flip is the last one with probability p, and with probability (1-p) you keep going. Then what is the probability a tower contains exactly k pointers?

(d) (3 points) Let random variable `X` denote the number of pointers in a single tower. Compute `E[X]`, the expected number of pointers in a tower if the probabilities are as described in part c). Show your work!

(e) (2 points) Define a skip list to have height (H) equal to k if the maximum level of any tower is level $k-1$, and suppose towers are created by flipping fair coins. We have derived an upper bound on the height of an n item skip list:

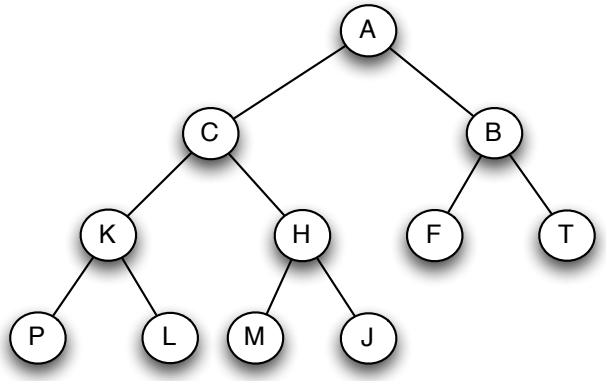
$$P(H > k) \leq \_\_\_\_\_$$

Circle the correct bound: $\frac{n}{2^k}$ $1-(\frac{n}{2})^k$ $(\frac{1}{k})^n$ $(1-\frac{1}{kn})$ $(\frac{1}{2})^{nk}$

(f) (2 points) Use the result from part e) to give a bound on the probability that a skip list containing 2^{20} keys (approx. 1m) has height greater than 60?

6. **[Binary Heaps – 20 points].**

(a) (8 points) Consider the following minHeap.



The minHeap above was the result of a sequence of `insert` and `removeMin` operations. Assume that the last operation was an `insert`. Which of the following keys could have been the one inserted last? Circle all possible keys.

A B C F H J K L M P T

(b) In many graph algorithms we need to be able to update or change the value of a key in a heap. Function `changeKey(int index, const kType & newKey)` changes the key in position `index` to value `newKey` and restores the heap property. The partial interface for the `Heap` class is here:

```
template <class kType>
class Heap{
public:
    ...
    void insert(const kType & k);
    kType & removeMin();

private:
    vector <kType> theHeap;
    int size;

    int leftChild(int i);
    int rightChild(int i);
    int parent(int i);

    void heapifyUp(int i);
    void heapifyDown(int i);
    ...
};
```

i. (7 points) Write function `changeKey`. Be sure to comment your code!

```
void Heap<kType>::changeKey(int index, const kType & newKey) {
```

```
}
```

ii. (3 points) What is the tightest bound on the worst case running time of `changeKey`?

$$O(1) \quad O(\log n) \quad O(n) \quad O(n \log n) \quad O(n^2)$$

iii. (2 points) Is `changeKey` a public or private member function of the `Heap` class?

7. **[Priority Queues – 20 points].**

Consider the following code fragment:

```
minPQ <Integer> pq;
for (int i = 0; i < n; i++) {
    pq.insert(a[i]);
    if (pq.size() > k) pq.removeMin();  // <3 <3 <3 <3 <3
}
for (int i = 0; i < k; i++)
    cout << pq.removeMin() << endl;
```

Assume that `a[]` is an array containing `n` integers, `n` $\geq$ `k`≥ 1, and `minPQ` is implemented using a binary heap.

(a) (5 points) What does the code output?

(b) (5 points) What is the tightest bound on its worst case running time?

$O(k \log k)$ $O(k \log n)$ $O(n \log k)$ $O(n \log n)$ $O(nk)$ $O(n^2)$

Now suppose we remove the line marked `<3 <3 <3 <3 <3`, and we implement `minPQ` as an unsorted array.

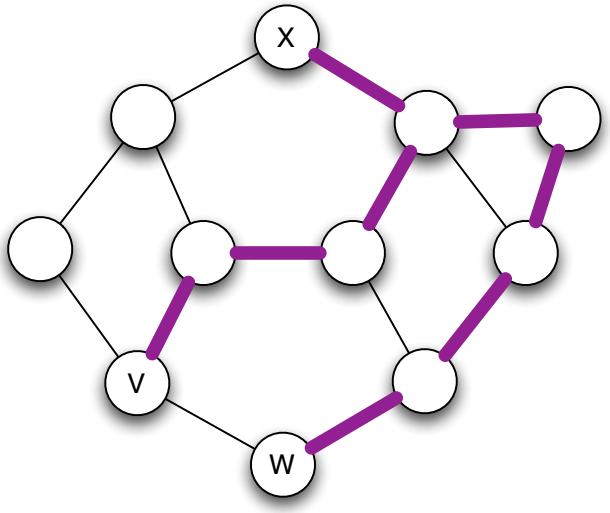
(c) (5 points) What does the code output?

(d) (5 points) What is the tightest bound on its worst case running time?

$O(k \log k)$ $O(k \log n)$ $O(n \log k)$ $O(n \log n)$ $O(nk)$ $O(n^2)$

8. **[Landmark Paths – 15 points].**

Given an undirected graph $G = (V, E)$, and a *landmark* vertex `x`, your task is to design an algorithm that finds the length of the shortest path from one vertex `v` to another vertex `w` passing through vertex `x`. One such landmark path from `v` to `w` is illustrated in the graph below, but notice that it's not the shortest one! (Note that the path *is* allowed to traverse an edge more than once, and each traversal is counted once in the path length, so that the total path length in the graph below is 9.)



(a) (3 points) The solution to this problem involves employing one of the graph algorithms we've discussed in class. Circle the appropriate one:

DFS BFS Prim's MST Kruskal's MST

(b) (2 points) How many times would you employ the algorithm you chose in part (a)?

(c) (2 points) All of the algorithms in part (a) have to start somewhere! Circle the vertex or vertices from which you would start the algorithm.

V W X anywhere

(d) (5 points) Very briefly describe how you would compute and report the shortest landmark path length.

(e) (3 points) What is the worst case running time of the best algorithm for finding the shortest landmark path?

9. **[Algorithms – 20 points].**

Below is partial pseudocode from an algorithm we discussed in class. Assume it is a member function of a graph class consisting of weighted, undirected edges. Also assume that any instance of the graph class contains a collection of vertices V with $|V| = n$ and a collection of edges E with $|E| = m$.

```
graph &  graph::_____________________( vertex s ) {

    for each vertex v in V {
        d[v] = INFINITY;
        labelled [v] = false;
        predecessor[v] = empty;
    }

    d[s] = 0;

    for n iterations {
        vertex v = minimum distance vertex among unlabelled vertices; // (12)
        labelled[v] = true;

        for each vertex w adjacent to v
            if (!labelled[w])
                if (weight(v,w) < d[w]) {
                    d[w] = weight(v,w);                          // (18)
                    predecessor[w] = v;
                }
    }

    graph returnGraph;
    returnGraph.V = V;
    for each vertex v in V
        if (v != s)
            returnGraph.addEdge(v,predecessor[v]);

    return returnGraph;
}
```

(a) (4 points) What is the name of the function described above, and what does it return?

(b) (3 points) In the worst case, how many times does is the assignment statement `d[w] = weight(v,w);` made over the execution of the entire algorithm?

(c) Suppose that the degree of every vertex in the graph is no more than 10.

i. (5 points) Which graph implementation gives the most efficient running time?

ii. (5 points) Which implementation of a priority queue should be used in lines 12 and 18?

iii. (3 points) What is the running time if you implement the function this way? (Your answer should be in terms of n and/or m, and should be as accurate as possible.)

(scratch paper, page 1)

(scratch paper, page 2)