University of Illinois at Urbana-Champaign
Department of Computer Science

Final Examination

CS 225 Data Structures and Software Principles
Spring 2010
7-10p, Wednesday, May 12

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 18 pages. The last sheet is scratch paper; you may detach it while taking the exam, but must turn it 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	30		
2	10		
3	20		
4	15		
5	15		
6	10		
7	15		
8	15		
9	20		
Total	150		

[illegible]

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

MC1 (2.5pts)

P

a. 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.

MC2 (2.5pts)

c. 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.

head tail

bottom top

push O(1)

pop O(n)

MC3 (2.5pts)

~~d.~~ e

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 $O(\log_m n)$. ✓

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

(iii) Every order 2 B-Tree is also a Binary Search Tree. ~~✗~~ ✓

Make one of the following choices.

(a) Only statement (i) is true.

(b) Only statement (ii) is true.

(c) Only statement (iii) is true.

(d) Exactly two of the above statements are true.

(e) All of these statements are true.

MC4 (2.5pts)

c.

What is the best definition of a collision in a hash table?

(a) Two entries are identical except for their keys. ✗

(b) Two entries with different data have the exact same key.

(c) Two entries with different keys have the same exact hash value. ✓

(d) Two entries with the exact same key have different hash values.

(e) None of the above is correct.

MC5 (2.5pts)

d.

Which of the following expressions represents the *load factor* for a hash table of size m containing n keys?

(a) $m+n$

(b) $m*n$

(c) m/n

(d) n/m ✓

(e) None of these is the load factor.

$\frac{n}{m}$

MC 6 (2.5pts)

b.

Suppose T is a min-heap storing n keys. Your task is to design an algorithm for reporting all keys in T that are less than or equal to a query key x (which is not necessarily in T). Your algorithm should run in time $O(k)$, where k is the number of keys in T whose value is less than or equal to x. Which of the following algorithms would you adapt to complete this task?

root

left right

(a) `RemoveMin` X

[✓ (b) `PreOrder` tree traversal]

(c) `Union` of disjoint sets X

(d) `Find` in a BST X

(e) None of these solves the problem.

MC7 (2.5pts)

d.

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

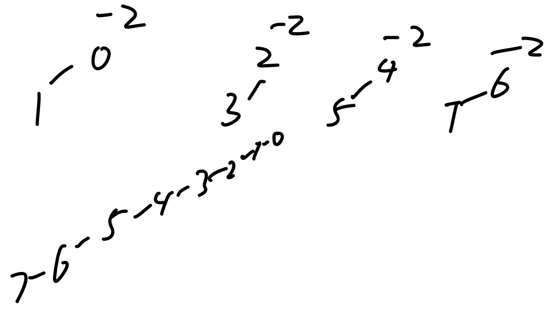
[✓ (c)]

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

(d) X

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.



MC8 (2.5pts)

n=2 17

C.

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

n=3 15

(a) $n - 1$

(b) $4n + 1$

[✓ (c) $4n + 3$]

(d) $5n + 2$

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

h=1 7

n=2 11

h=3 15

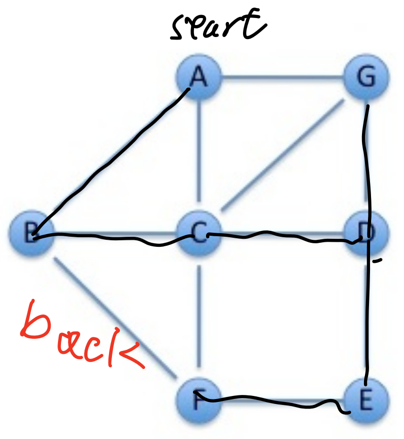
$5n+2-n+1$

$4n+3$

MC9 (2.5pts)

d.

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.

MC10 (2.5pts)

d.

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.

$n + m\lg n + O(m)$

$n + n^2 + m\lg n + O(m)$

MC11 (2.5pts)

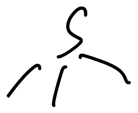
b. Suppose we implement an algorithm to determine whether or not a simple undirected graph is connected using a modification of a graph algorithm we saw in class. What is the tightest worst case asymptotic running time of the best algorithm to do this?

(a) $O(n^2)$

(b) $O(n + m)$ ✓

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

(d) $O(n)$



MC12 (2.5pts)

Consider two vertices x and y that are simultaneously on the queue during execution of BFS from vertex s in an undirected graph. Which of the following is/are true?

~~a.~~

I. The number of edges on the shortest path from s to x is at most one more than the number of edges on the shortest path from s to y.

II. The number of edges on the shortest path from s to x is at least one less than the number of edges on the shortest path from s to y. ✗

C. III. There is a path from x to y. ✗ ✓

(a) Only one of these statements is true. ✓

(b) Items I. and II. are true. ✗

(c) Items I. and III. are true.

(d) Items II. and III. are true. ✗

(e) All three statements are true. ✗

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;  x
    int * hi5 = &linkedin; x
    int * orkut = new int[4];                orkut[ ]
    int * friendster = orkut; x
*   int ** bebo = new int * [3];
    bebo[0] = &friendster[3];              bebo[1]
    bebo[1] = new int;
    bebo[2] = bebo[0];

    // add code to free each piece of dynamically
    // allocated memory exactly once
    delete[] orkut;    orkut = NULL;
    delete bebo[1];
    delete[] bebo ;

}
```

(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.

vector<BMP*> album ;

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

- when the function that create a instance class finishes.
- when the function that create a instance class on heap and then call delete to free the instance class.

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:
    OneDSearch();
    void insert(double X);
    double query (double y) const;
















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 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.

```
/* helper function to decide whether to change the current best node */
bool OneDSearch :: shouldReplace(const double& query,
                                 const double& currentBest,
                                 const double & potential)
{
    if ((currentBest -query)*(currentBest-query) >=
        (potential-query)*(potential-query)) return true;
    return false;
}
double OneDSearch::query(double y) const {
    double currentBest
    bool radius = -1;        // set -1 to represent has not find yet
    if( root != NULL) query(root, currentBest,y, radius);
    return currentBest;
} /* helper recursive function
void OneDSearch ::query (const KDTreeNode* root, double& currentBest,
                         const double& y, double& radius) const {
    // check stop condition
    if (root->left == NULL && root->right ==NULL ) {
        if (radius == -1 || shouldreplace( y, currentBest, root -> value)){
            radius = abs( y-root->value); // need #include <cmath> first
            currentBest = root->value;
        }
        return;
    }
```

(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...

$O(1)$ $O(\log n)$ [✓ $O(n)$] $O(n \log n)$ $O(n^2)$

```
// handle main point
if ( y <= root->left ) query(root->left , ...
                      ->value
       right                   right , ...

// check main node
```

// check other plane
if (y <= root->left->value && y+radius >= root -> value)
check right
if (. . .

4. **[Hashing – 15 points].** check left

(a) (6 points) Insert the following keys, k, into an initially empty linear probing hash table of size 7, using the following table of hash values ($h(k) = i\%7$, where k is the ith letter of the alphabet):

k	i	$h(i)$
P	16	2
R	18	4
O	15	1
B	2	2
I	9	2
N	14	0
G	7	0

0	1	2	3	4	5	6
N	O	P	B	R	I	G

(b) (9 points) Suppose a hash table of size m handles collisions by separate chaining, and that it employs periodic resizing. 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)

computation time

- (assumption 2 about the hash function)

SUHA

- (assumption 3 about the hash function)

determinic

- (assumption 1 about the resizing scheme)

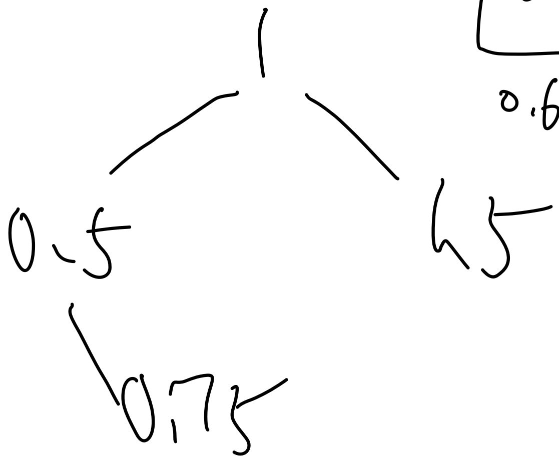
double the size

- (assumption 2 about the resizing scheme)

$\alpha > 0.6$

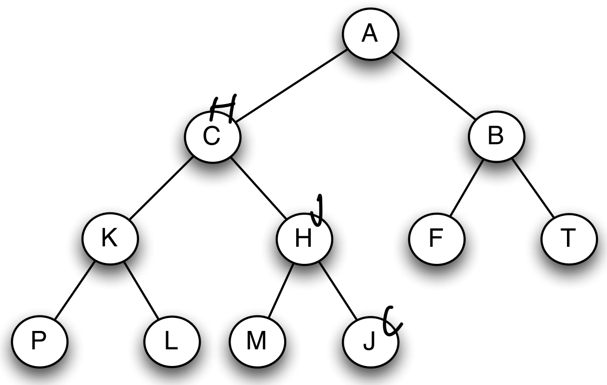
$\boxed{0.4}$

0.6



5. **[Binary Heaps – 15 points].**

(a) (5 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);
    ...
};
```

[illegible]

i. (5 points) Write function `changeKey`. Be sure to comment your code!

```
void Heap<kType>::changeKey(int index, const kType & newKey) {
    if (theHeap[index] == newKey) return;
    if (theHeap[index] > newKey) {
            theHeap[index] = newKey;
            heapifyUp[index];
    }
    if (                    <          )

                  down
}
```

优先级?

min Heap

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

$O(1)$ [✓ $O(\log n)$] $O(n)$ $O(n \log n)$ $O(n^2)$

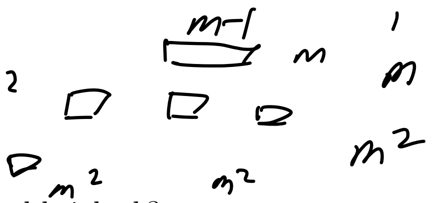
iii. (2 points) Is `changeKey` a public or private member function of the `Heap` class?

public

$h=0 \quad m-1$

$h=1 \quad m-1+m(m-1)$

$h=2 \quad m-1+m(m-1)+m^2(m-1)^2$



6. **[B-Trees – 10 points].**

(a) What is the maximum number of *nodes* in a B-Tree of order m and height h?

$n-$

$$\frac{1-m^{h+1}}{1-m}$$

(b) What is the maximum number of *keys* in a B-Tree of order m and height h?

~~$h=0 \quad m$~~

~~$h=1 \quad m+m^2$~~

~~$h=2 \quad m+m^2+m^4$~~

~~$= \begin{cases} m, & h=0 \\ m+\sum_{n=1}^{h} m^{2n} \end{cases}$~~

$$\sum_{n=0}^{h} m^n(m-1)^n$$

(c) What is the maximum number of disk seeks in a search for a particular key in a B-Tree of order m and height h?

h

7. [**Miscellaneous Data Structures – 15 points**].

(a) Dictionaries

i. (3 points) List 5 dictionary data structures we've studied:

BST
linked list
AVL
heap
KD-Tree

ii. (2 points) List 5 data structures we've studied that are NOT dictionaries:

stack
queue
array
graph

Dset

(b) PalTome

Suppose social network PalTome needs to support only 2 operations: a) declare A and B to be pals, and b) determine whether or not A and B are pals. At PalTome, if A and B are pals, then all of A's pals are also B's pals, and vice versa. (It's a friendly place!)

i. (5 points) Which of the following data structure(s) would you use to implement PalTome?

Stack Binary Tree Hash Table [✓ Disjoint Sets] Heap Graph

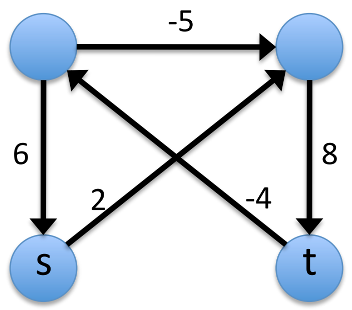
ii. (5 points) What is the worst case running time PalTome can guarantee on a sequence of m operations if there are n people on the site?

$O(m \cdot \log^* n)$

$O(m)$

8. **[Graphs – 15 points].**

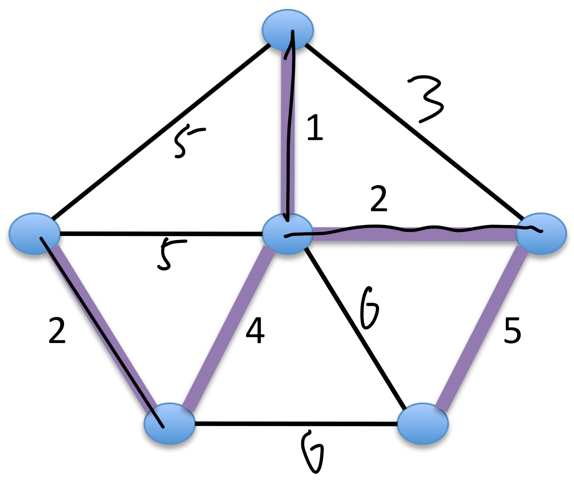
(a) (5 points) Consider the following weighted, directed graph:



Is there a shortest path from vertex s to vertex t? Why or why not?

No , as there exists negative circle

(b) (5 points) The edges highlighted on the graph below are a minimum spanning tree of this graph, and it is unique (there is no other spanning tree of this weight in the graph). Label each unlabeled edge with the smallest integer weight possible.



(c) (5 points) Suppose a graph is implemented using an adjacency list. Under what condition will Prim's algorithm be more efficient if it employs a priority queue implemented using a heap, rather than an unsorted array?

when it's often to update the elements' value

because after updating, we need to resort, and heap is more convinent

[illegible]

heap: O(nlogn + mlogn)

unsorted array: O(n²)

9. **[Algorithms – 20 points].**

Below is partial pseudocode adapted from an algorithm we discussed in class. *Assume it is a member function of a graph class.* Also assume that any instance of the graph class contains a collection of vertices V with $|V| = n$ and a collection of undirected edges E with $|E| = m$. Finally, for simplicity, you may assume that all graphs originally passed as a parameter to this function are connected, and all edges and vertices have been initialized to be "unexplored."

```
void graph::_____________________( graph & G, vertex s ) {

    G.setLabel(s) = VISITED;
    if (!existsVertex(s)) addVertex(s);  // only add new vertices

    for each vertex w in G.adjacentVertices(s):
            if (G.getLabel(w)== UNEXPLORED) {
                G.setLabel((s,w), DISCOVERY);
                addVertex(w);
                addEdge(s,w);

                _______________(G,w); //recursive call

            }
            else if (G.getLabel((s,w))==UNEXPLORED){
                 G.setLabel((s,w), BACK);
                 addEdge(s,w);
             }
}
```

(a) (4 points) What is a good name for the function described above, and what does it do?

DFS_helper

(b) (4 points) How many times is the addEdge function called?

~~the number of graph edges that can contribute to the path from the vertex s to other vertexs.~~

m

(c) (4 points) Which graph implementation gives the most efficient running time?

adjancy ~~matrix~~ list

(d) (4 points) Using the implementation in the previous part, what is the running time of the algorithm for general connected undirected graphs with n vertices and m edges? (Your answer should be in terms of n and/or m, and should be as accurate as possible.)

$O(n+m)$

(e) (4 points) Suppose that the degree of every vertex in the graph is no more than 10. What is the running time of this algorithm on graphs of this type? (Your answer should be in terms of n and/or m, and should be as accurate as possible.)

$O(n+10n) = O(11n)$
$= O(n)$

(scratch paper, page 1)