Name: _____Sample Solution_____

Email address: _____

CSE 373 Autumn 2011: Midterm #2

(closed book, closed notes, NO calculators allowed)

Instructions: Read the directions for each question carefully before answering. We may give partial credit based on the work you **write down**, so if time permits, show your work! Use only the data structures and algorithms we have discussed in class or that were mentioned in the book so far.

Note: For questions where you are drawing pictures, please circle your final answer for any credit.

Good Luck!

Total: 65 points. Time: 50 minutes.

Question	Max Points	Score
1	12	
2	14	
3	6	
4	7	
5	6	
6	8	
7	12	
Total	65	

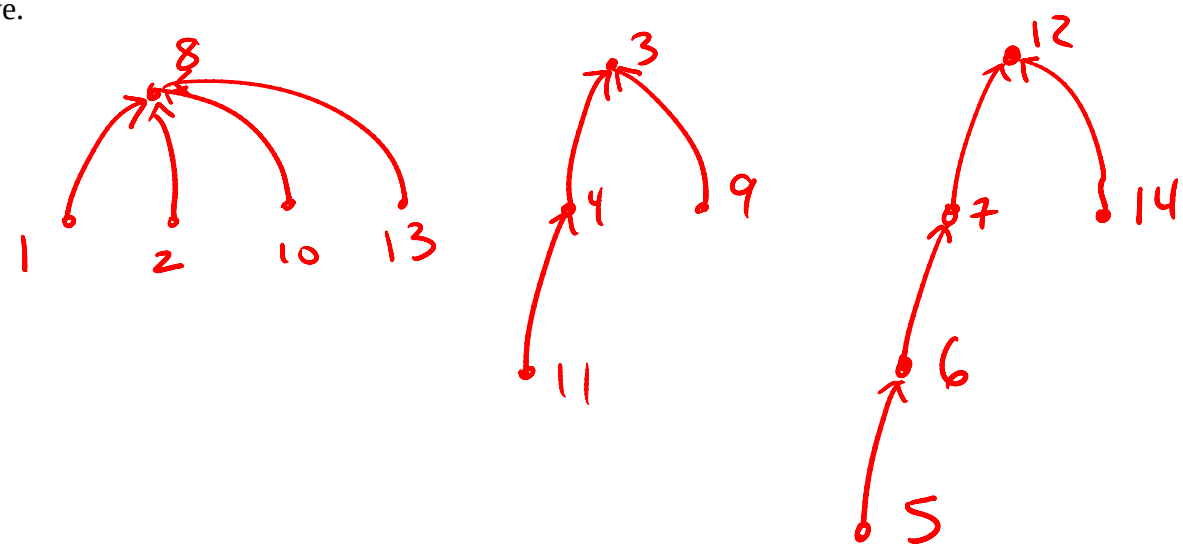
1) [12 points total] Disjoint Sets

The uptrees used to represent sets in the union-find algorithm can be stored in two *n*-element arrays. The **`up`** array stores the parent of each node (or -1 if the node has no parent). The **`weight`** array stores the number of items in a set (its weight) if the node is the root (representative node) of a set. (If a node is not a root the contents of its location in the **`weight`** array are undefined – we don't care what value it holds, it can be zero or any other number.)

The following shows a collection of sets containing the numbers 1 through 14, without the **`weight`** array filled in:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`up`	8	8	~~-1~~ 8	3	~~6~~ 12	~~7~~ 12	12	-1	3	8	4	-1	8	12
`weight`								9				5		

a) **[3 points] Draw** a picture of the uptrees represented by the data in the **`up`** array shown above.

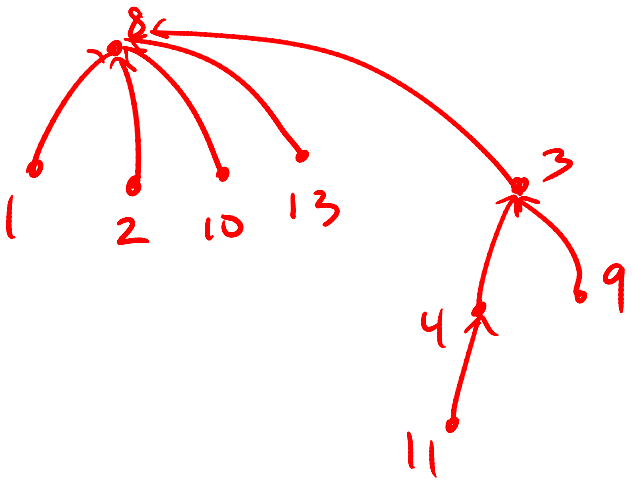


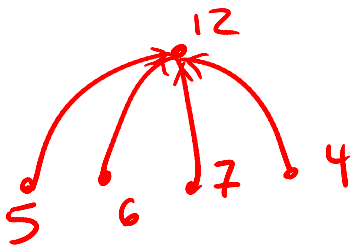
1) (cont)

b) **[3 points]** Now, **draw** a new set of uptrees to show the results of executing:

```
union(find(2), find(4));
find(5);
```

Regardless of how the trees from part a) were constructed, here assume that `find` uses **path compression** and that `union` uses **union-by-size (aka union by weight)**. In case of ties in size, always make the higher numbered root point to the lower numbered one. Unioning a set with itself does nothing.





c) **[2 points]** Update the **`up`** and **`weight`** arrays at the top of the previous page to reflect the picture after part b). That is, fill in the contents of the **`weight`** array and update the contents of the **`up`** array.

d) **[2 points]** What is the worst case big-O running time of a single union operation if union by size (aka union by weight) and path compression are used (assuming you are always passed roots as parameters)? N = total # of elements in all sets. **(no explanation required)**

O(1)

e) **[2 points]** What is the worst case big-O running time of a single find operation without path compression if when unioning, the higher numbered root points to the lower numbered one (union by size is NOT used)? N = total # of elements in all sets**. (no explanation required)**

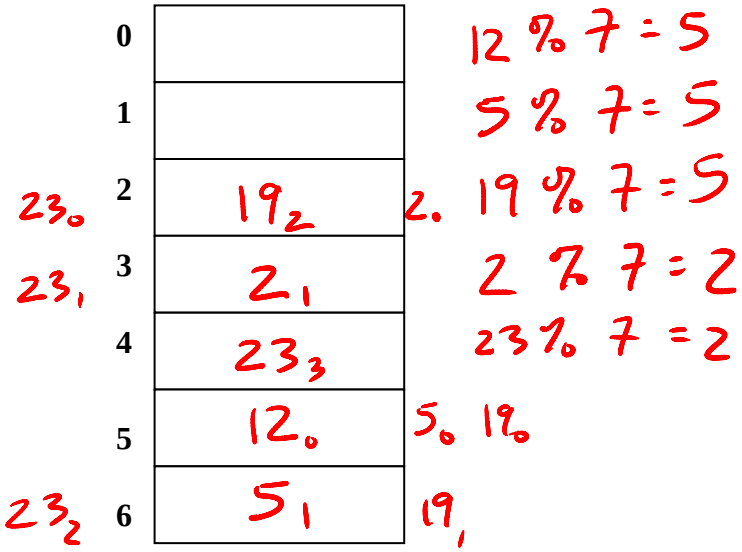
O(N)

2) [14 points total] Hashing:

[4 points each] Draw the contents of the two hash tables below after inserting the values shown. Show your work for partial credit. *If an insertion fails, please indicate which values fail and attempt to insert any remaining values*. The hash function used is H(k) = k mod tablesize.

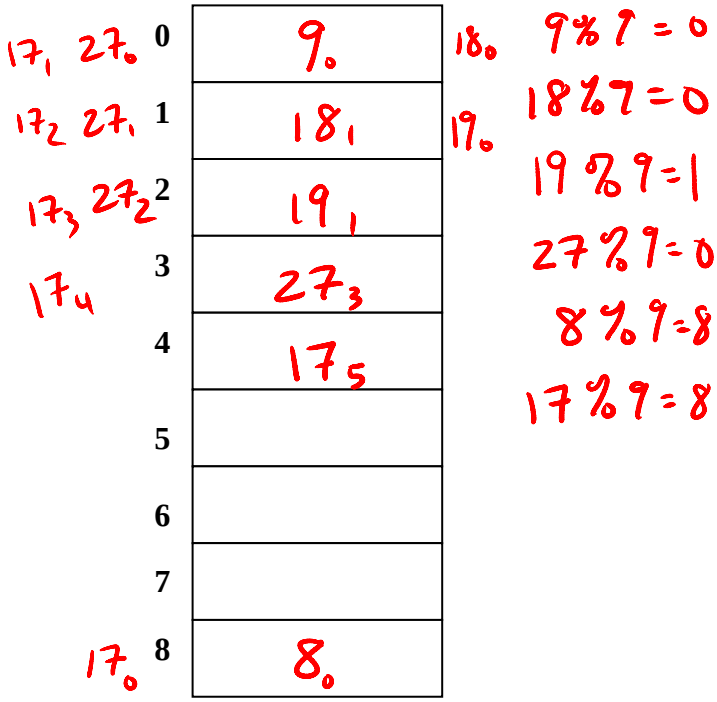
a) Quadratic probing

12, 5, 19, 2, 23



b) Linear Probing

9, 18, 19, 27, 8, 17



2) (cont)

c) **[1 point]** What is the **load factor** for table a)? 5/7

d) **[1 point]** What is the **load factor** for table b)? 6/9

e) **[1 point]** Table a) will (circle one):

i. gradually degrade in performance as more values are inserted

ii. possibly fail to find a location on the next insertion

iii. none of the above

f) **[1 point]** Table b) will (circle one):

i. gradually degrade in performance as more values are inserted

ii. possibly fail to find a location on the next insertion

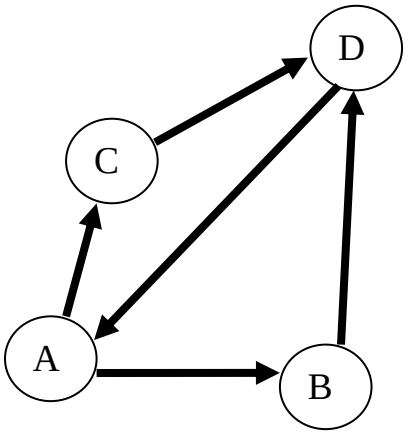
iii. none of the above

g) **[2 points]** Given an open addressing hash table where quadratic probing is used to resolve collisions, what would be the worst case big-O runtime of rehashing the values if we assume that the original tablesize = N^2 (before re-hashing) and there are currently N items in the hash table? **(no explanation required)**

$O(N^2)$

3) [6 points total] Graphs

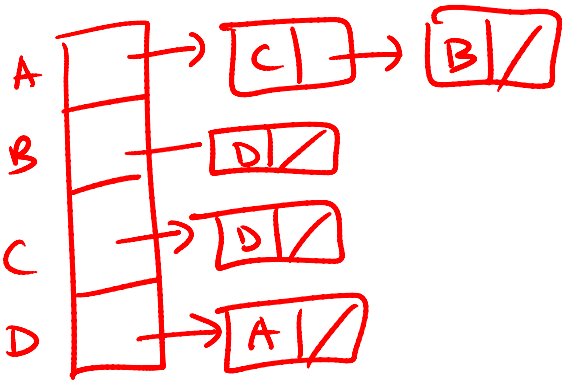
a) **[2 points]** Draw both the adjacency matrix and adjacency list representations of this graph. *For this problem, assume there are no implicit <u>self loops</u>* (e.g. an edge from A to A). That is, unless there is a self loop explicitly drawn in the graph, there should not be one in the representation.



<u>Adjacency Matrix</u>:

	A	B	C	D
A	F	T	T	F
B	F	F	F	T
C	F	F	F	T
D	T	F	F	F

<u>Adjacency List</u>:



What is the worst case big-O running time of the following operations (use V and E rather than N in your answers). **No explanation is required**.

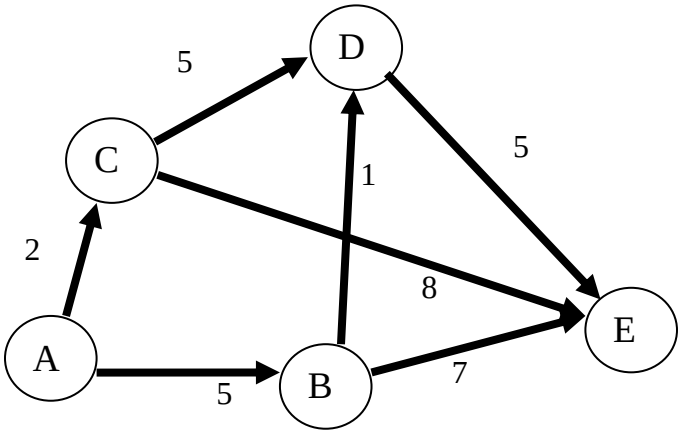
b) **[2 points]** Find the *in-degree* of a single vertex whose graph is stored in an adjacency matrix.

O(V)

c) **[2 points]** Find the *out-degree* of a single vertex whose graph is stored in an adjacency list.

O(V) (O(d) where d is the out degree also accepted)

4) [7 points total] Graphs
Use the following graph for the questions *on this page*:



a) **[2 points]** If possible, list ***two*** valid topological orderings of the nodes in the graph above. If there is only one valid topological ordering, list that one ordering. If there is no valid topological ordering, state why one does not exist.

A, B, C, D, E
A, C, B, D, E

b) **[2 points]** What is the worst case big-O running time of topological sort for a graph represented as an adjacency list implemented with a queue? (use V and E rather than N in your answer) **No explanation is needed**.

O(V+E) (O(E) also accepted)

c) **[2 points]** This graph is: (Circle all that are true):

complete	strongly connected	**directed**
undirected	**acyclic**	**weakly connected**

d) **[1 point]** What is the in-degree of node D?

2

5) [6 points] Graphs – Give an ***exact answer*** ***(NOT an answer in big-O)*** to each question below in terms of V (the number of vertices in the graph). Do NOT use E in your answers. **No explanation needed**.

a) What is the maximum number of edges possible in a directed graph if self loops are **not** allowed?

$V^2 - V$ or $V(V-1)$

b) What is the minimum number of edges possible in a connected undirected graph that does **not** have self loops?

$V - 1$

c) What is the minimum degree of a vertex in a complete undirected graph that has self loops?

V

6) [8 points] Memory Hierarchy & Locality: Examine the code example below:

```
a = 3;
b = 100;
c = 68;
d = 2;
w[10] = (50 * c) + 12;
for (i = 1; i < 100; i++) {
     a = a * 7;
     j = x[i] + x[i+1];
     k = y[1] + c;
     e += d + z[i] + 10;
}
```

Considering only their use in the code segment above, for each of the following variables, indicate below what type of locality (if any) is demonstrated. Please circle ***all that apply*** (you may circle more than one item for each variable):

a	**spatial locality**	**temporal locality**	**no locality**
b	**spatial locality**	**temporal locality**	**no locality**
c	**spatial locality**	**temporal locality**	**no locality**
d	**spatial locality**	**temporal locality**	**no locality**
w	**spatial locality**	**temporal locality**	**no locality**
x	**spatial locality**	**temporal locality**	**no locality**
y	**spatial locality**	**temporal locality**	**no locality**
z	**spatial locality**	**temporal locality**	**no locality**

7) [12 points total] Running Time Analysis:

- **Describe the most time-efficient way to implement the operations listed below**. Assume no duplicate values and that you can implement the operation as a member function of the class – with access to the underlying data structure.
- Then, give the tightest possible upper bound for the ***worst case*** running time for each operation in terms of *N*. ****For any credit, you must explain *why* it gets this worst case running time.** You must choose your answer from the following (not listed in any particular order), each of which could be re-used (could be the answer for more than one of a) -d)).

  $O(N^2)$, $O(N^{½})$, $O(N \log N)$, $O(N)$, $O(N^2 \log N)$, $O(N^5)$, $O(2^N)$, $O(N^3)$, $O(\log N)$, $O(1)$, $O(N^4)$, $O(N^N)$, $O(N^6)$, $O(N (\log N)^2)$, $O(N^2 (\log N)^2)$

a) Merging two binary min-heaps (both implemented using an array) each containing N elements into a single **binary min heap**. **Explanation:**

a) $O(N)$

First concatenate the two heaps into one array, this involves copying each array into the new array 2N (O(N)), then run Floyd's buildheap on the new array O(N). Floyd's buildheap takes O(N) because the total distance that elements must be percolated down is O(N) in the worst case.

b) Finding an element in a **hash table** containing N elements where separate chaining is used and each bucket points to an unsorted linked list. The table size = N. **Explanation**:

b) $O(N)$

Worst case is when all N elements hash to the same bucket. Finding an element then involves doing the hash function to get to the bucket O(1), then searching thru all N element to find the one you are looking for. (In the worst case it is the last one in the list or it is not there.)

c) Finding what the maximum value is in a hash table currently containing N elements, the hash table is of tablesize N^2. The hash table uses open addressing and double hashing to resolve collisions. **Explanation:**

c) $O(N^2)$

Since hash tables are not sorted, basically have to search the entire table. Since the table is size N^2, this search takes time $O(N^2)$. There is not really a worst case because you have to search through all values before you know which one is the maximum.

d) IncreaseKey(k, v) on a **binary min heap** containing N elements. Assume you have a reference to the key k. v is the amount that k should be increased. **Explanation:**

d) $O(\log N)$

Increasing the value takes O(1) time. Then k may need to be percolated down further in the heap. In the worst case you are increasing the current minimum in the heap to be so large that it has to move to the bottom row in the heap. This takes O(log N) since there are max O(log N) levels in the heap.