Name: _____Sample Solution____________________

Email address: ____________________________________

CSE 373 Autumn 2012: 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.

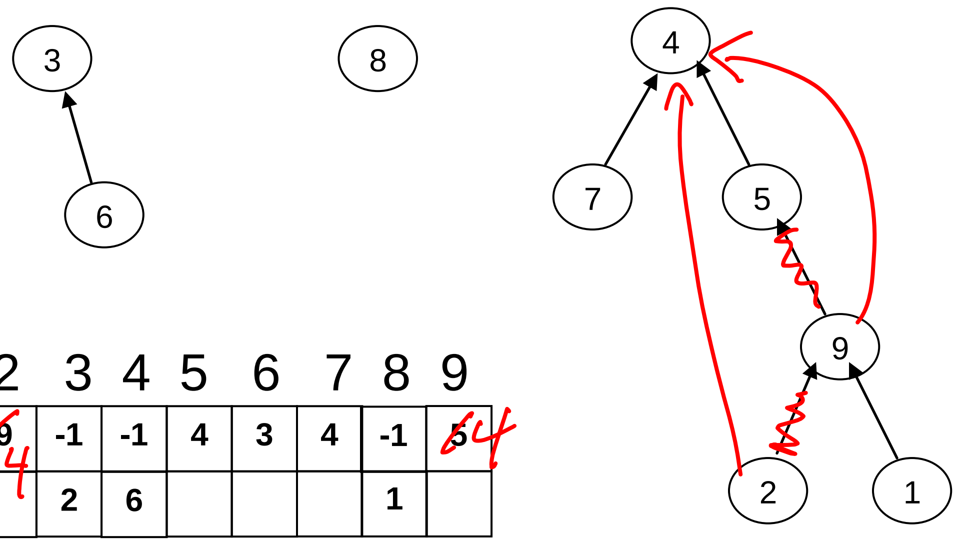
Note: For questions where you are drawing pictures, please circle your final answer for any credit.

Good Luck!

Total: 67 points. Time: 50 minutes.

Question	**Max Points**	**Score**
1	10	
2	13	
3	8	
4	7	
5	9	
6	8	
7	12	
Total	67	

1) [10 points total] Disjoint Sets: Given the array representation of up-trees discussed in class (where `weight[x]` = total number of values in set x, and `up[x]` == -1 indicates that node x is a root node), for example (`weight` is unspecified for non-root nodes):



	1	2	3	4	5	6	7	8	9
up	9	~~9~~ 4	-1	-1	4	3	4	-1	~~5~~ ~~6~~ 4
weight			2	6				1	

a) **[3 points]** Fill in the code below for **Union by Size** (weighted union). ***On a tie in size,*** the set referred to by the first parameter should be added to the set referred to by the second.

```
int up[N];
int weight[N];

void union(int x, int y) {
     // Assuming x and y are roots of two different sets,
     // fill in code to implement union by size (weight)
     // Break ties in size by adding the set x
     // to the set y.
   if ( weight[x] ≤ weight[y]) {
        up[x] = y;
        weight[y] = weight[x] + weight[y];
   } else
        up[y] = x;
      weight[x] = weight[x] + weight[y];
   }
}
```

1) (cont)

b) **[3 points]** Fill in code below for **find** (without path compression) below. DO NOT INCLUDE PATH COMPRESSION, you will lose points if you do.

```
int find(int x) {
     // Assuming x is an element, fill in code to
     // return the root of the set x belongs to.
```

```
while (up[x] != -1) {
    x = up[x];
}
return x;
}
```

↑

Iterative

```
if (up[x] == -1)
    return x;
else
    return find(up[x]);
}
```

↑

Recursive

c) **[2 points]** What is the worst case running time of a single **find** operation in your code above. That is, assume union by size is used but path compression is *not* used. N = total # of elements in all sets. (no explanation required)

O(log N)

d) **[2 points]** Update the **diagram** AND the **two arrays** on the previous page to reflect the effect of doing a find operation WITH path compression on the value 2. That is, do not use the find operation you implemented above, do a find operation that uses path compression.

(2 and 9 now point to 4)

2) [13 points total] Hashing:

a) [6 points] 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.

Table 1: **Separate chaining**, (where each bucket points to an unsorted linked list)

Insert: 4, 21, 6, 28, 14, 12

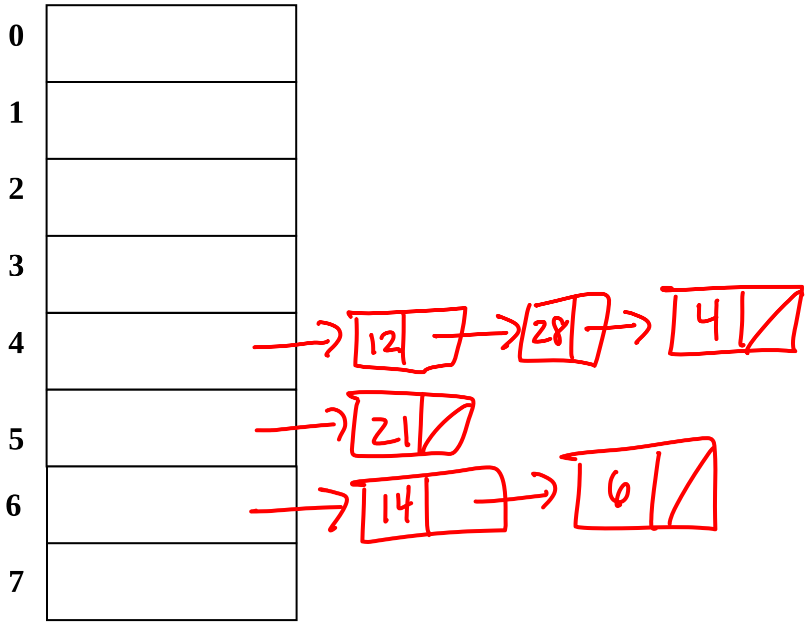


Table 2: **Quadratic Probing**

Insert: 3, 10, 4, 11

0		
1	11_2	
2		
3	3_0	10_0
4	10_1	4_0 11_0
5	4_1	11_1
6		

b) [2 points] Give the load factor for each table:

Load factor for Table 1:

6/8

Load factor for Table 2:

4/7

2) (cont)

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

(can just push on front of list)

i. performance of inserts will gradually degrade as more values are inserted

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

iii. be fine on the next insertion, but may fail to find a location on any insertions after that

[✓ iv. none of the above]

d) **[1 point]** Table 2 will (circle one):

i. performance of inserts will gradually degrade as more values are inserted

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

iii. be fine on the next insertion, but may fail to find a location on any insertions after that

iv. none of the above

iii. is actually true for this particular hash table, although in general the next insertion is not guaranteed. Both ii and iii will be counted as correct for this problem.

e) **[1 point]** In Table 1, if we randomly pick one of the values and delete it (using the method that makes the most sense): (circle one):

(less stuff to look through)

[✓ i. this may improve performance of find operations]

ii. this will not affect performance of find operations

iii. this may degrade performance of find operations

iv. this may cause a find on a different value to fail

v. none of the above

f) **[1 point]** In Table 2, if we randomly pick one of the values and remove it from the table, leaving that location empty: (circle one):

this is not a valid way of doing deletion.

i. this may improve performance of find operations

ii. this will not affect performance of find operations

iii. this may degrade performance of find operations

[✓ iv. this may cause a find on a different value to fail]

v. none of the above

g) **[1 point]** If we re-hash the values from Table 2 into a new table of size 17 (circle one):

~~i.~~ locations 7 through 17 of the new table will initially be empty

~~ii.~~ the values will all be clustered at the beginning of the new table

~~iii.~~ the values will all be clustered at the end of the new table

[✓ iv. we are guaranteed to be able to insert at least five more values into the new table]

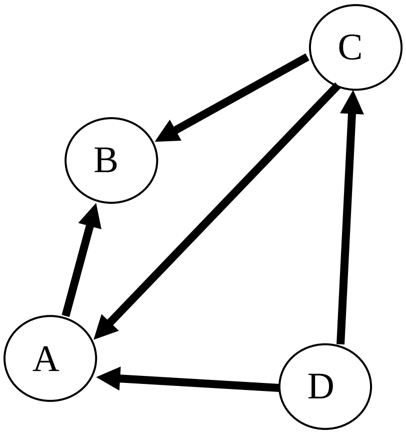
v. none of the above

$\frac{4}{17} + \underset{\text{more inserts}}{4} = \frac{8}{17}$ $\lambda < \frac{1}{2}$

So 5th insert will be possible.

3) [8 points total] Graphs

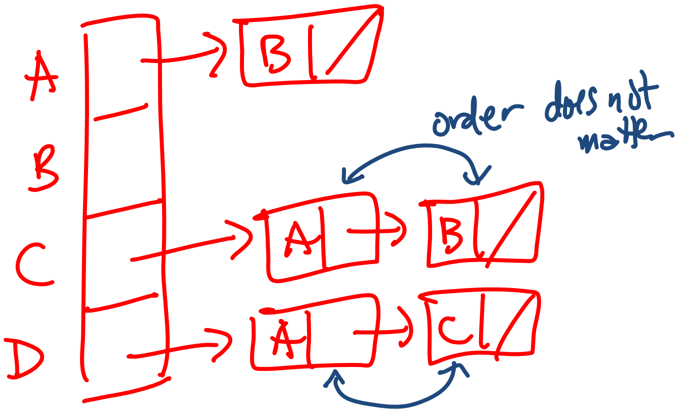
a) **[2 points]** Draw both the adjacency matrix and adjacency list representations of this graph. *For this problem, assume there are no implicit* *self loops* (e.g. an edge from A to A). That is, unless there is a self loop explicitly drawn in the graph, there should not be one in the representation.



Adjacency Matrix:

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

Adjacency List:



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]** Insert an edge in a graph that is stored in an adjacency matrix.

O(1)

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

O(V)

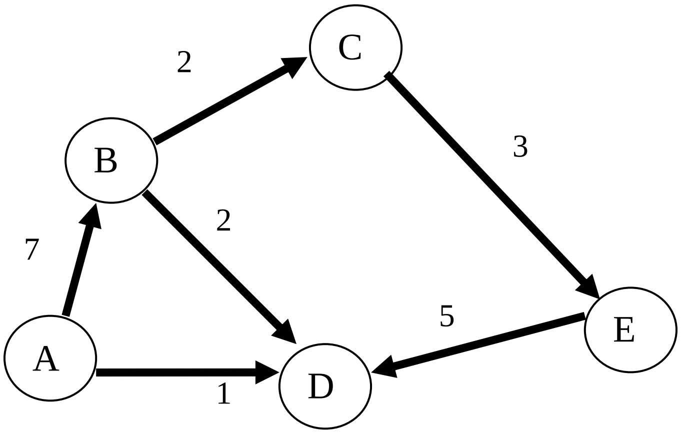
d) **[2 points]** Find the *out-degree* of a single vertex whose graph is stored in an adjacency list.

O(V)

(O(d) where d is the outdegree 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, E, D

(This is the only ordering.)

b) **[2 points]** What is the worst case big-O running time of optimized topological sort for a graph represented as an adjacency list? (note this refers to the *optimized* version presented in lecture where a queue is used) (use V and E rather than N in your answer) **No explanation is needed**.

$O(V + E)$

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

[✓ directed] [✓ weakly connected] undirected

complete [✓ acyclic] strongly connected

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

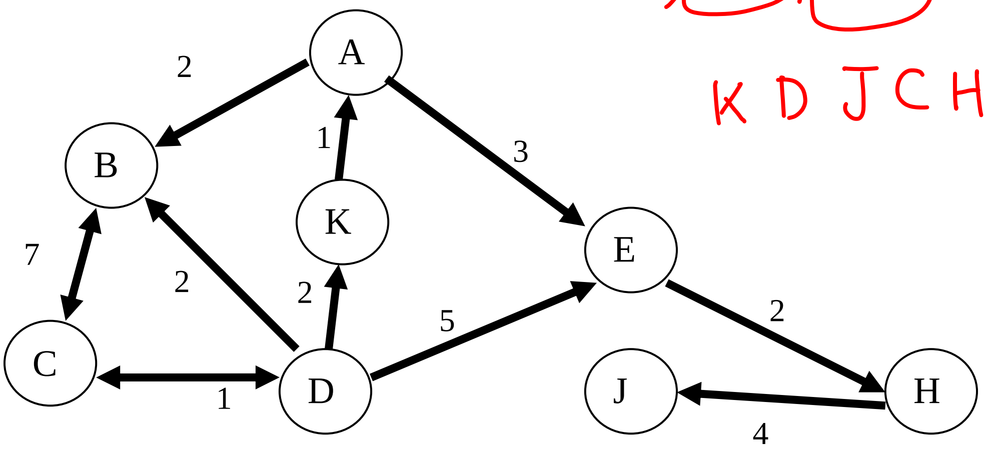
1

Queue

K J D H C E B A

K D J C H B E A

5) [9 points total] Traversals and More Graphs



a) **[3 points]** Given the graph above and using the algorithm described in lecture, list one plausible order that vertices in the graph above would be processed if a **breadth first traversal** is done starting at A.

A	B E	C H	D J	K
A	E B	H C	J D	K
	1 away	2 away	3 away	4 away

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.

b) **[2 points]** What is the maximum number of edges possible in a directed graph if self loops are allowed?

V^2

c) **[2 points]** What is the maximum degree of a vertex in a complete undirected graph that has self loops?

V

d) **[2 points]** What is the minimum number of edges possible in a connected undirected graph that does not have self loops?

$V - 1$

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

```
b = 2045;
i = 1
x[i] = i;
while (i < 1000) {
     a = y[300];
     b = z[i] + c;
     c = c + w[i+500] + y[i];
     i = i + 1;
}
```

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
i	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 pts) **Running Time Analysis:**

- Give the tightest possible big-O upper bound for the ***worst case*** running time for each operation listed below in terms of *N*. 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, including knowing the number of values currently stored. For array-based implementations, assume that the array is large enough – does not need to be grown.

- **You do *NOT* need to give an explanation.**

- Give your answer in big-O, **only fully simplified answers will receive full credit**.

a) A rehash operation in an open addressing hash table where quadratic probing is used to resolve collisions. Assume original tablesize = 3N (before re-hashing), new tablesize = N^3 and there are currently N items in the hash table.

If all hashed to same location # of collisions:
$0+1+2+3+ \ldots + N-1$

a) $O(N^2)$

b) Decrease (also known as DecreaseKey) in a priority queue containing N values implemented with a binary min heap, assuming you have direct access to the element you are decreasing.

cost of a percolate up

b) $O(\log N)$

c) Find in a separate chaining hash table where each bucket points to a binary search tree. Assume: tablesize N^2 and there are currently N items in the hash table.

all in same bucket, tree = a list

c) $O(N)$

d) Union by size in a disjoint set implemented with path compression (total number of elements = N). (Assuming passed roots/set names as parameters)

d) $O(1)$

e) Find the maximum value in hash table where linear probing is used to resolve collisions. Assume: tablesize = N^5 and there are currently N items in the hash table.

Examine all N^5 locations

e) $O(N^5)$

f) Given *three* arrays of unsorted integers, each one of size N^2, create a single binary min heap out of those values.

concatenate + call buildheap,
total size = $3N^2$

f) $O(N^2)$