Name: Solution
UW Net ID: ____________

CSE373 Summer 2015: Final
(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.

Questions are not necessarily written in order of difficulty; so don't spend too long on one question and feel free to skip around.

Read questions carefully and show your work!

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

You have 6 questions (many with multiple parts) and 60 minutes for the exam.

Good luck!

Question	Max Points	Score
1	9	
2	10	
3	10	
4	10	
5	10	
6	11	
Extra Credit	2	
Total	**60**	

1. (9 pts total) **Short answer:**
   a. (1 pt) If the upper bound on the runtime for a sequence of n operations on a data structure is O(n^5), what is the amortized runtime of each operation?

      O(n^4)

   b. (1 pt) What is the amortized runtime of adding an element to a queue implemented as an array with resizing?

      O(1)

   c. (1 pt) Why might it be faster to access two sequential items in an array than two sequential items in a linked list?

      spatial locality

   d. (1 pt) Would you use a queue or a stack to implement Breadth First Search of a graph?

      Queue

   e. (1 pt) Which performs better in the best case, insertion sort or selection sort?

      insertion sort

   f. (1 pt) Circle One. Being able to verify a solution to a problem in polynomial time is a necessary condition for that problem to be in:
      a. P
      b. NP
      [✓ c.] both
      d. neither
   g. (1 pt) Circle One. Being able to solve a problem in polynomial time is a necessary condition for that problem to be in:
      [✓ a.] P
      b. NP
      c. both
      d. neither
   h. (1 pt) What is the algorithm technique used in parallel by the Fork-Join model we went over in class: Greedy, Divide and Conquer, Dynamic Programming or Backtracking?

      D & C

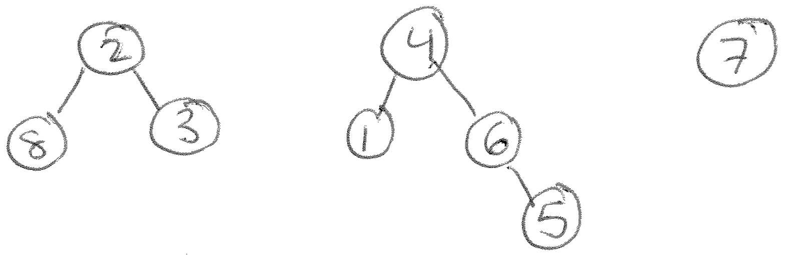
   i. (1 pt) Name one algorithm that uses a Greedy approach.

      Kruskal, Prim, Dijkstras

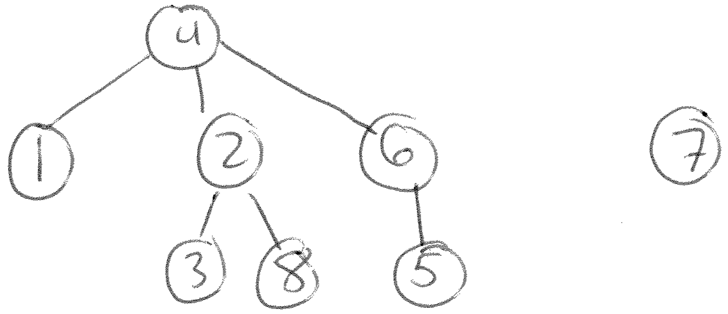
2. (10 pts total) **Disjoint Sets & Union Find:** Below is an array that represents a forest of up-trees for as an underlying data structure for the Union-Find ADT.

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

a. (3 pts) Draw the up-tree(s) represented by the array. Each node can be numbered by its corresponding index. Note that the indexing begins at 1 (not 0).

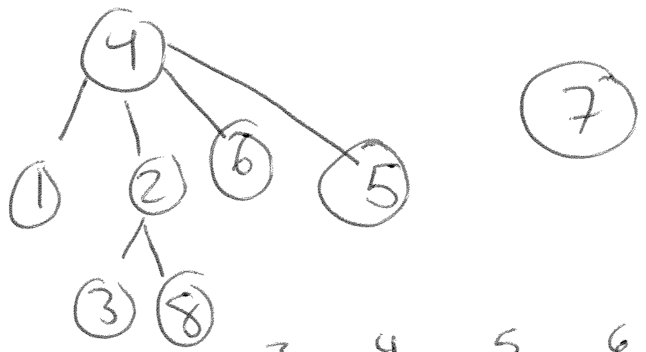


b. (2 pts) Perform **Union(4, 2)** using **union-by-size,** and write either the corresponding array or draw the up-tree representation.



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

c. (2 pts) Based on your answer to part b, draw the tree or write the array after performing **Find(5)** using **path compression.**



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

d. (1 pt) What does **Find(1)** return?

4

e. (2 pts) In Kruskal's algorithm to find a minimum spanning tree of a graph, edges are evaluated in order of cost and added to the MST if they do not create a cycle. Briefly (in one sentence or less) describe how the union-find ADT be used in this algorithm.

It can be used to check if adding an edge would create a cycle in the MST

3. (10 pts total) **Hash Tables:**

a. (4 pts) Insert the following items in the hash table below. Use the hash function ***k%TableSize*** and use open addressing and quadratic probing to resolve collisions

Insert: 45, 32, 15, 25, 35

0	1	2	3	4	5	6	7	8	9
		32		35	45	15			25

b. (2 pts) What is the load factor of this hash table? In the worst case, how long would it take to insert the next item in this hash table?

$\lambda = 5/10$, $O(n)$, with the quadratic probing & $\lambda \geq 1/2$, you could get an infinite loop, but not with this hash table

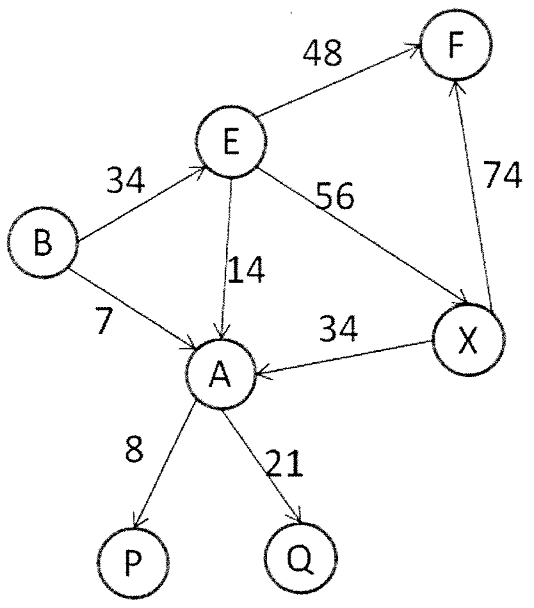
c. (2 pts) Why might deleting a number from the table without using lazy deletion cause a problem?

Would not be able to find other element that hash to the same index (or along the same probing path)

d. (2 pts) Name one way you could improve upon this hash table or hash function that would likely reduce collisions.

- prime table size
- increase table size
- better hash function

4. (10 pts total) **Graphs:** Use the directed, weighted graph below for all parts (a through e)



a. (3 pts) Create the adjacency matrix representation of the graph. Use -1 to represent "not an edge" and the weights for edges that exist.

	B	E	A	F	X	P	Q
B		34	7				
E			14	48	56		
A						8	21
F							
X			34	74			
P							
Q							

-1 elsewhere

b. (2 pts) Give one reason why an ***adjacency list*** might be a better representation for this graph than the ***adjacency matrix***.

This is a sparse graph, $|V| \approx |E|$, so a list takes up less space

c. (2 pts) Give one reason why you might still use an ***adjacency matrix*** rather than the ***adjacency list***.

better time to find/delete/insert an edge

d. (1 pt) How would the matrix change if the graph was undirected? (You don't have to draw the new matrix, just describe it).

it would be symmetric across the diagonal

e. (2 pts) If possible, perform a valid ***topological sort*** of the graph, and show the final output in order. If more than one order is possible, just show one.

BEX AFPQ
AFQP
FAQP
FAPQ
APFQ
APQF

5. (10 pts total) **Shortest Paths and Minimum Spanning Tree:** Given a weighted, undirected graph, Dijkstra's algorithm computes the shortest paths from a start node to all other nodes.

The following pseudocode is the efficient (uses a heap) version of Dijkstra's algorithm that we went over in class:

```
dijkstra(Graph G, Node start) {
     for each node x{
         x.cost=infinity
         x.known=false
    }
    start.cost = 0
    build-heap with all nodes
    while(heap is not empty) {
        b = deleteMin()
        b.known = true
        for each edge (b,a) in G{
            if(!a.known){
                if(~~b.cost~~ + weight((b,a)) < a.cost){
                    decreaseKey(a, ~~b.cost~~ + weight((b,a))
                    a.path = b
                }
            }
        }
    }
}
```

a. (2 pts) What does the cost of each node represent in Dijkstra's algorithm?

best known cost of path to reach that node from start node found so far

b. (2 pts) Why do we use a heap to find the next node to evaluate instead of iterating over all the vertices?

better asymptotic complexity for sparse graph to find next vertex to evaluate.

c. (4 pts) Prim's algorithm is very similar to Dijkstra's, but it outputs the Minimum Spanning Tree of a graph instead of finding the shortest paths. ***Change the pseudocode for Dijkstra's algorithm into Prim's algorithm.*** You may do this either by writing directly on the pseudocode above or by describing the changes below (e.g. Delete line 5, Replace line 6 with ...)

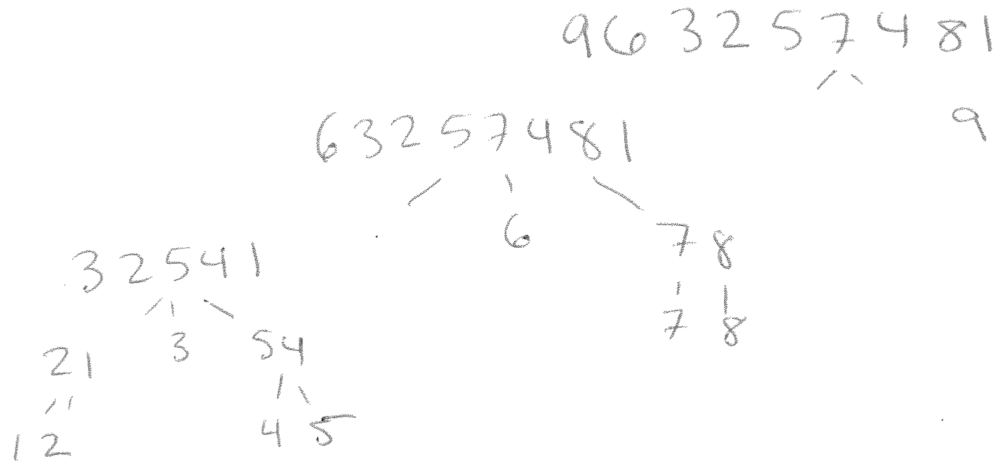
Note: you would generally pick a random node to start the MST with Prim's, instead we will use the start node passed in as an argument to our pseudocode.

see page 8

d. (2 pts) What does the cost of each node represent in Prim's algorithm?

The lowest cost edge to add node to set/cloud of known vertices

6. (11 pts total) **Sorting:**
   a. (3 pts) Sort the array [9 6 3 2 5 7 4 8 1] using ***QuickSort*** by drawing the recursion tree showing how it splits after each partition. Chose the first element of each array as the pivot. Assume that the partitioning step correctly partitions the elements, but within each partition the elements remain in the same order as they were in the step before.



   b. (2 pts) Why is picking the first element as a pivot a bad idea? What should you do instead?

Will take $O(n^2)$ on sorted/ reverse sorted input. Use random pivot or median of 3 instead

   c. (2 pts) How do the worst-case runtimes for QuickSort and MergeSort compare?

Quicksort is $O(n^2)$ vs Mergesort $O(n \log n)$

d. (4 pts) **Radix sort:** Sort the following elements [321, 322, 122, 92, 147, 1, 197] using a 10 bucket LSD (least significant digit) radix sort. Show the elements of each bucket after each pass.

0	1	2	3	4	5	6	7	8	9
	321 1	322 122 9					147 197		

0	1	2	3	4	5	6	7	8	9
1		321 322 122		147					92 197

0	1	2	3	4	5	6	7	8	9
1 92	122 147 197		321 322						

7. (2 pts) **Extra Credit:** What do you think, is P=NP?