CSE 373 Final Exam 17wi

Name: Solution Quiz Section: ____________
TA: ____________ Student ID #: ____________

- This test is closed book, closed notes, closed calculators, closed electronics.
- You have **110** minutes to complete this exam. You will receive a **10 point deduction** if you work before the exam starts, or keep working after the instructor calls for papers.
- If you write your answer on scratch paper, please clearly write your name on every sheet and write a note on the original sheet directing the grader to the scratch paper. **We are not responsible for lost scratch paper or for answers on scratch paper that are not seen by the grader due to poor marking.**
- Code/Pseudocode will be graded on proper behavior/output and not on style, unless otherwise noted.
- If you enter the room, you must turn in an exam and will not be permitted to leave without doing so.

Good luck and have fun!

#	Problem	Points	Score
1	Sorting Short Answer	15	
2	Sorting Code	10	
3	Union-Find and Kruskal's Algorithm	12	
4	Graphs Analysis and Pseudocode	18	
5	Dijkstra's Algorithm	11	
6	Minimum Spanning Trees	5	
7	Topological Sort	6	
8	Design Decisions	6	
9	Short Answer	7	
10	Psuedocode and Algorithms	10	
	Total	100	

Advice:

- Read questions carefully. Understand a question before you start writing.
- Write down assumptions, thoughts and intermediate steps so you can get partial credit. Clearly circle your final answer.
- The questions are not necessarily in order of difficulty. Skip around. Make sure you get to all of the questions.
- If you have questions, ask them. Any clarifications will be noted on the screen.

1) Sorting Short Answer (15 points)

a) **(4 points)** For each sorting algorithm below, give the asymptotic worst-case tight bound using "Big-O" notation. **You should give the tightest and simplest bound possible.** You do NOT need to explain your answer.

Merge Sort	Insertion Sort	Selection Sort	Heap Sort
O(n log(n))	$O(n^2)$	$O(n^2)$	O(n log(n))

b) **(2 points)** With a pivot rule of "always pick the element at index `lo`", give an example array of numbers, where the length is at least length 6, that gives quick sort the worst-case runtime. What is this quick sort worst-case runtime using "Big-O" notation?

1	2	3	4	5	6	7	8

OR

8	7	6	5	4	3	2	1

$O(n^2)$

c) **(1 point)** Briefly describe (in 1 sentence) a strategy discussed in lecture that makes quick sort pick a better pivot and avoid this worst-case runtime and get a better average runtime.

Median of three: pick the median value of input[lo], input [(hi +lo)/2) and input[hi – 1]

d) **(1 point)** Briefly explain, in 1-2 sentences, why we use a cut off for divide and conquer sorting algorithms like quick sort or merge sort.

We use a cut off to stop our recursive calls. The overhead for recursion can be more expensive for small inputs of N, than a simple N^2 sort.

e) **(2 points)** Can heap sort be done in place? If it can, briefly in 1-2 sentences, describe how. If it can't, briefly in 1-2 sentences, describe why not.

Yes. Every deleteMin() operation frees up one space in the array at the end of the heap values. You can insert the next min in that empty spot, and then reverse your input at the end (or use a max heap) to get sorted ascending order.

You are given the following unsorted input (an array of patients in an emergency room) where each element has a String name and an int representing the intensity of their wound. In a sorted ordering, an element should come first in if it has a **larger** int.

[(“Chloe”, 5), (“Matthew”, 7), (“Pascale”, 2), (“Hunter”, 5)]

f) **(1 point)** Provide a sorted ordering of this data that was performed by a stable sort:

[(“Matthew”, 7), (“Chloe”, 5), (“Hunter”, 5), (“Pascale”, 2)]

g) **(1 point)** Provide a sorted ordering of this data that was performed by a not stable sort:

[(“Matthew”, 7), (“Hunter”, 5), (“Chloe”, 5), (“Pascale”, 2)]

h) **(2 points)** Suppose we use radix sort to sort the numbers below, using a radix of 3 (digits are 0, 1 or 2). Show the state of the sort after each of the first two passes, not after the sorting is complete. If multiple numbers are in a bucket, put the numbers that “come first” closer to the top.

Numbers to sort (in their initial order):
222, 1, 10, 21, 201, 112, 200

After the first pass:

0	1	2
010 200	001 021 201	222 112

After the second pass:

0	1	2
200 001 201	010 112	021 222

i) **(1 point)** Is there a proven lower bound asymptotic runtime on comparison sorting? You do NOT need to explain.

Yes. It is $\Omega(N \log(N))$.

2) Sorting Code (10 points)
Write Java code for a method **`mergeSection`** that takes two `int[]`s and indices `lo` (inclusive) and `hi` (exclusive) **to perform the merge part of merge sort**. The `int[] input` parameter, between `lo` and `hi`, stores two sorted sections (each section is half of the range from `lo` to `hi`). The `int[] output` parameter is the output array to merge the sorted sections into. After merge is called, the `int[] output` should store the result from merging the two sorted sections from the first array together. As with merge discussed in lecture, if the range between `lo` and `hi` has an odd length, the smaller sorted section will be the first half of the range.

You may not make any extra data structures (has to be done with O(1) extra space). You may assume the array parameters are the same size and non-null. You may not make any other assumptions about the length of the array parameters. You may assume that `lo` and `hi` are valid indices and represent bounds on , but you may not make any assumptions about duplicate values in the `int[] input`.

Four Examples (underlined parts of the input array showing sorted sections to merge):

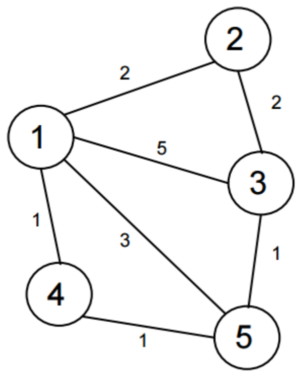
```
int[] input = {2, 4, 6, 1, 3, 5, 7};
int[] output = {0, 0, 0, 0, 0, 0, 0, 0, 0};
mergeSection(input, output, 0, list.length);
// after call, output: [1, 2, 3, 4, 5, 6, 7]

int[] input = { 1, 3, 2, 4, 7, 1, 3, 5};
int[] output = {0, 0, 0, 0, 0, 0, 0};
mergeSection(input, output, 0, 4);
// after call, output: [1, 2, 3, 4, 0, 0, 0, 0]

int[] input = {5, 6, 7, 4, 2, 3},
int output = {0, 0, 0, 0, 0, 0}
mergeSection(input, output, 3, 6);
// after call, output: [0, 0, 0, 2, 3, 4]

int[] input = {1},
int[] output = {0}
mergeSection(input, output, 0, 1);
```

One solution:

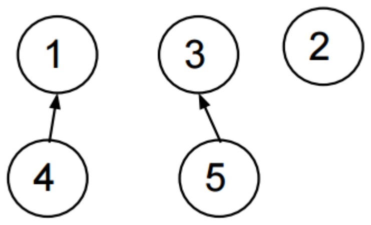
```
public static void mergeSection(int[] input, int[] output, int lo, int hi) {
    int i = lo;
    int mid = (lo + hi) / 2;
    int j = mid;
    int k = lo;
    while (i < mid && j < hi) {
      if (input[i] <= input[j]) {
        output[k] = input[i];
        i++;
        k++;
      } else {
        output[k] = input[j];
        j++;
        k++;
      }
    }
    while (i < mid) {
      output[k] = input[i];
      i++;
      k++;
    }
    while (j < hi) {
      output[k] = input[j];
      j++;
      k++;
    }
}
```

3) Union-Find and Kruskal's Algorithm for MST (12 points)
You're performing Kruskal's algorithm for finding a minimum spanning tree on the following undirected weighted graph. After considering 2 edges and adding them to your minimum spanning tree, you have the following up-trees to represent your disjoint sets:

Graph:



Up-Trees:



a) **(2 points)** Fill in the array representation of the up-trees, where up[x] == -1 indicates a root node and weight[x] represents the number of values in the disjoint set (unspecified for non-root nodes).

	1	2	3	4	5
up	-1	-1	-1	1	3
weight	2	1	2	-	-

b) **(4 points)** With up-trees represented as the up and weight arrays as described above, fill in the pseudocode for the find operation. Implement the algorithm using **the path compression optimization** described in lecture.

```
// Assuming x is an element, return the representative element for x
// Assume you have access to int[] up and int[] weight
int find(int x) {
   int parent = x;
   while (up[parent] != -1) {
      parent = up[parent];
   }
   while (up[x] != -1) {
      int oldParent = up[x];
      up[x] = parent;
      x = oldParent;
   }
   return parent;
}
```

c) **(3 points)** With up-trees represented as the up and weight arrays as described above, fill in the pseudocode for the union operation. Implement the union algorithm using the **union by weight optimization** described in lecture. If the pair of root nodes passed to your method are tied, resolve that tie by choosing the first parameter as the representative element.

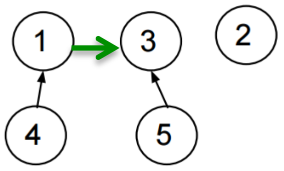
```
// Assuming x and y are root elements, perform a union of their sets
// Assume you have access to int[] up and int[] weight
void union(int x, int y) {
   if (x != y) {
      if (weight[x] < weight[y]) {
         weight[y] = weight[y] + weight[x];
         up[x] = y;
      } else {
         weight[x] = weight[y] + weight[x];
         up[y] = x;
      }
   }
}
```

d) **(1 point)** What is the next edge E that will be added to the MST by Kruskal's algorithm? Write the edge as *(u, v, n)*, where *u* is one of the nodes, *v* is the other node, and *n* is the weight.
(4, 5, 1)
e) **(1 point)** What series of union and find operations will you do for this edge E to add it to your MST? Write them below as *union(x, y)* and *find(x)*, where *x* and *y* are values of nodes.

```
int x = find(4);
int y = find(5);
if (x != y) {  // optional if statement, part of Kruskal's algorithm to determine if edge to add
    union(x, y);
}
```

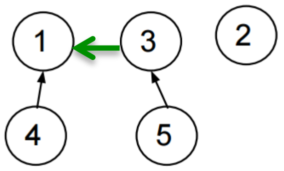
f) **(1 point)** Update the array representation and the diagram below after adding this edge E and updating the up-trees. There are two equally valid solutions. **Do not do any optimizations.**

	1	2	3	4	5
up	3	-1	-1	1	3
weight	-	1	4	-	-



OR

	1	2	3	4	5
up	-1	-1	1	1	3
weight	4	1	-	-	-



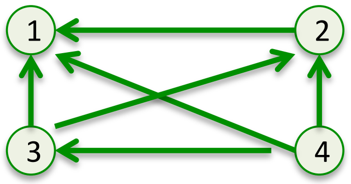
4) Graph Analysis and Pseudocode (18 points)

a) **(1 point)** What is the minimum number of edges possible in a connected undirected graph that does not have self edges? Give an exact answer, not in big-O notation, in terms of |V|. You do NOT need to explain.

|V| - 1

b) **(7 points)** Suppose a directed graph has k nodes, where each node corresponds to a number (1, 2, ..., k) and there is an edge from node i to node j if and only if i > j.

I. **(1 point)** Draw the graph (using circles and arrows) assuming k = 4.

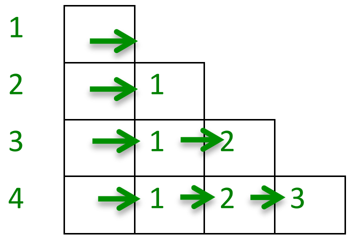


II. **(2 points)** Draw both an adjacency matrix representation and an adjacency list representation of the graph assuming k = 4. If unsure about notation, clearly label your drawings to convey what your notation means for the graph representation.

Adjacency Matrix:

	1	2	3	4
1	F	F	F	F
2	T	F	F	F
3	T	T	F	F
4	T	T	T	F

Adjacency List:



III. **(1 point)** In terms of k (if k is relevant), exactly how many edges are in the graph assuming an **arbitrary** value of k?

1 + 2 + 3 + 4 + ... + (k-1) or $\sum_{i=1}^{k-1} i$ or many other ways to write this

IV. **(1 point)** In terms of k (if k is relevant), exactly how many correct results for topological sort does this graph have assuming an **arbitrary** value of k?

1

V. **(1 point)** Is this graph (CIRCLE ONE):

(1) strongly connected [✓ **(2) weakly connected**] **(3)** unconnected

VI. **(1 point)** Is this graph (CIRCLE ONE):

[✓ **(1) dense**] **(2)** sparse

c) **(5 points)** Write the pseudodocode for a method BFS to print out an ordering of the graph vertices that can be reached in a breadth-first search given some starting `Vertex v`.

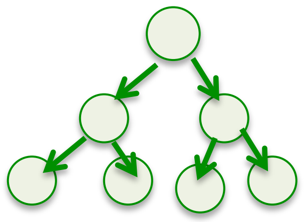
```
// For graphData, each Vertex a is mapped to a set containing
// Vertex b, if there is an edge (a, b) in the graph.
Map<Vertex, Set<Vertex>> graphData;

// Assume you have access to graphData and Vertex v is valid
void bfs(Vertex v) {
   Queue pending = new Queue();
   v.visited = true;
   pending.add(v);
   while (!pending.isEmpty()) {
      Vertex next = pending.remove();
      print(next);
      if (!v.visited) {
         next.visited = true;
         pending.add(next);
      }
   }
}
```

d) **(1 point)** What is the runtime of your pseudocode algorithm in terms of |E| and |V|?

O(|E|) tight bounds, O(|E| + |V|) okay

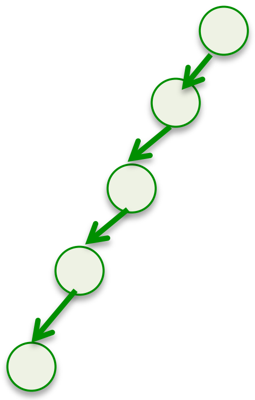
For the next two problems, recall that θ signifies an asymptotically tight bound.

e) **(2 points)** Draw a picture of a binary tree with N nodes (for your drawing arbitrarily make N = 5), where breadth-first search uses θ(N) auxiliary space. Briefly explain (in 1 sentence) why your tree makes BFS use θ(N) auxiliary space.



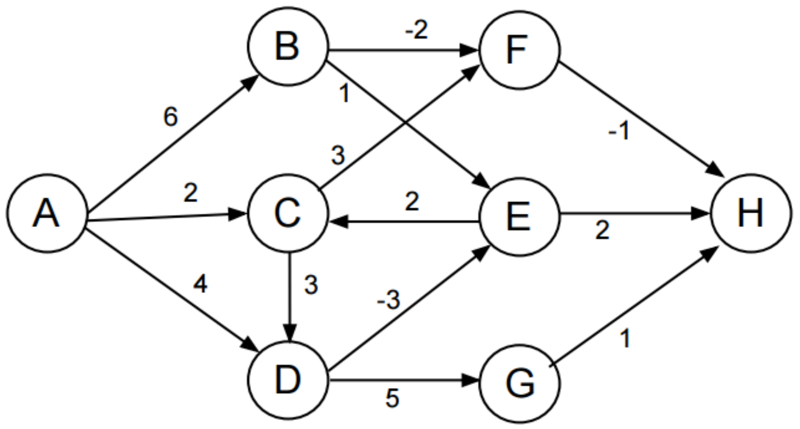
BFS uses a queue for a pending set, which will contain at maximum all the nodes at one level. The largest level is the bottom one, which contains a constant factor of N.

f) **(2 points)** Draw a picture of a binary tree with N nodes (for your drawing arbitrarily make N = 5), where depth-first search uses θ(1) auxiliary space. Briefly explain (in 1 sentence) why your tree makes DFS use θ(1) auxiliary space.



DFS (or BFS) uses a data structure of the pending nodes to process. At each level in this tree, we add the node to the pending set and then immediately pop it. That means we'll have at most 1 node in our auxiliary space.

5) Dijkstra's Algorithm (11 points)
Using the following graph:



a) **(5 points)** Use Dijkstra's algorithm with start node A to find the cost of the shortest path from A to each other node. For each node, **record the cost and the previous node in the path**. Keep track of the **order** you mark each node known for part (b).

	Cost	Path
A	0	-
B	6	A
C	2	A
D	4	A
E	1	D
F	5	C
G	9	D
H	3	E

b) **(1 point)** In what order would Dijkstra's algorithm mark each node as known?

A, C, D, E, H, F, B, G

c) **(1 point)** Using Dijkstra's algorithm, what is the shortest cost path from node A to node G and what is it's cost?

A -> D -> G   cost of 9

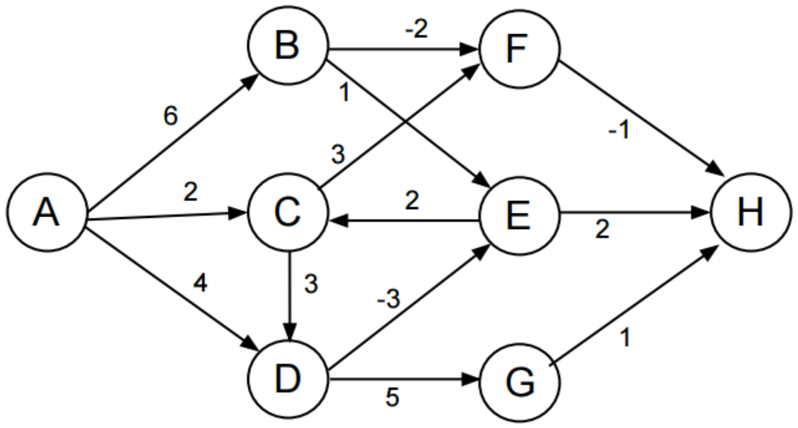
d) **(2 points)** Are any of the paths computed in part (a) incorrect? For each path that has an incorrect cost, write the correct lowest-cost path and explain why it was incorrect.

Yes, the cost to F is incorrect because of the negative edges. The shortest cost path found by Dijkstra's is A -> C -> F, with a cost of 5, but the actual shortest cost path is A -> B -> F with a cost of 4.

e) **(1 point)** Do self edges break Dijkstra's algorithm? Briefly explain, in 1-2 sentences, why or why not.

No. If the self edge is not negative, it does not affect Dijkstra's algorithm. Dijkstra's algorithm doesn't consider nodes that are already marked known, so self edges would never be considered.

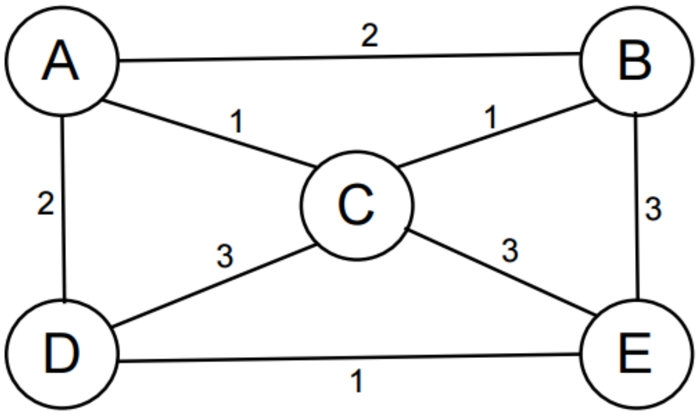
f) **(1 point)** The graph repeated for reference:



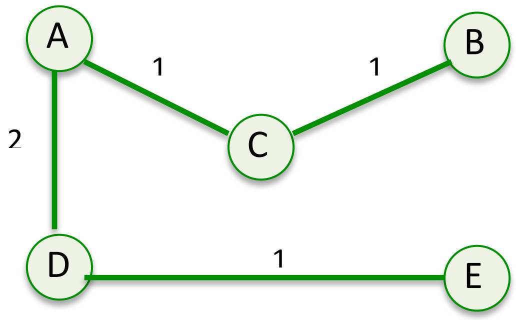
Is it possible to do a topological sort of the above graph? Briefly explain, in 1-2 sentences, why or why not.

No. This is not a DAG. There is a cycle from C -> D -> E.

6) Minimum Spanning Trees (5 points)
Consider the following undirected, weighted graph:



a) **(3 points)** Perform Prim's algorithm from the start node 'A' to find a minimum spanning tree. **Draw the final MST** and **list the edges** in the order they are added to the MST. Write an edge as *(u, v, n)*, where *u* is one of the nodes, *v* is the other node, and *n* is the weight.



(A, C, 1), (C, B, 1), (A, D, 2), (D, E, 1)

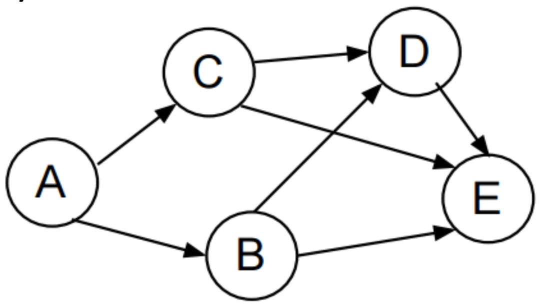
b) **(1 point)** Given a weighted, undirected graph with |V| nodes, assuming all weights are non-negative. If each edge has weight ≤ *w*, what can you say (in less than 1 sentence) about the cost of an MST?

The cost of the MST is <= w * (|V| - 1)

c) **(1 point)** Given a weighted, undirected graph with |V| nodes, if the cost of an MST is *c*, what can you say (in less than 1 sentence) about **the cost of the shortest path** returned by Dijkstra's algorithm when run with an arbitrary pair of vertices *(s, d)*, where *s* is the source and *d* is the destination?

The cost of the shortest path is <= c

7) Topological Sort (6 points)



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) **(3 points)** Given a `Vertex v` that has just been added to the topological ordering and a `Map<Vertex, Integer> inDegree` storing the state of the in-degrees for each node still to be processed in the topological sort algorithm (before adding v to the output ordering), perform the update step to inDegree to reflect that v has been processed.

```
// For graphData, each Vertex a is mapped to a set containing
// Vertex b, if there is an edge (a, b) in the graph.
Map<Vertex, Set<Vertex>> graphData;

// Update the state for the topological sort.
// Modify inDegree to accurately store the in-degree of each
// vertex after processing given Vertex v.
// Assume you have public access to graphData and parameters are valid
void updateInDegree(Vertex v, Map<Vertex, Integer> inDegree) {
   // your code here
   for (Vertex neighbor: graphData.get(v)) {
      if (inDegree.containsKey(neighbor)) {
         inDegree.put(neighbor, inDegree.get(neighbor) – 1);
      }
   }

}
```

c) **(1 point)** What is the runtime of your pseudocode algorithm in terms of |E| and |V|?
O(|V|)

8) Design Decisions (6 points)

For each of the following graph-based computational tasks, specify the type of graph most appropriate for the data in question in terms of **undirected or directed**, and **unweighted or weighted**.

In addition, choose the graph algorithm from the following list best suited to computing a solution:

- Breadth-First Search
- Depth-First Search
- Minimum Spanning Tree (e.g. Prim's or Kruskal's)
- Dijkstra's Algorithm
- Topological Sort

You do NOT need to explain your answers.

a) **(2 points)** You want to model a collection of Java files in a project. You know which files exist and which other Java files they depend on using. You want to determine the order that the files have to be compiled in before a given file f can be compiled and run.

Directed and unweighted graph. Topological Sort.

b) **(2 points)** You want to model how a tweet gets re-tweeted by followers on Twitter. You have data on who the users of Twitter are and all the followers of each user. Given a source, a tweet can be re-tweeted by any follower of that source. If a tweet gets arbitrarily re-tweeted k times, you want to know which users could have seen the tweet.

Directed and unweighted graph. BFS or DFS to a level / depth of k.

c) **(2 points)** You have locations and the distances between them. You want to know the shortest cost path from one source location to all other locations.

Undirected and weighted. Dijkstra's Algorithm.

9) Short Answer (7 points)
a) **(1 point)** For data that is mostly in sorted order, what is the best sorting algorithm to use? "Best" is defined as optimized for runtime and space complexity. You do NOT need to explain your answer.

Insertion sort

b) **(1 point)** For data that has a small range of values possible in the domain (less than 100 options), what is the best sorting algorithm to use? "Best" is defined as optimized for runtime and space complexity. You do NOT need to explain your answer.

Bucket Sort

c) **(2 points)** We discussed how data structures sometimes use lazy deletion for performance and correctness. For hash structures using open addressing (linear probing, quadratic probing, double hashing) collision resolution methods, is lazy deletion necessary? If it is necessary, explain in one sentence or with a brief example. If it isn't necessary, explain why not.

Yes, it is necessary. If you are using a probing technique, you need to mark an index as used so you know that there might be values after it in the future. For example, with linear probing and the following table of size 4 with hash function of (value + i) % tablesize.

0	4
1	8
2	12
3	

insert(4)
insert(8)
insert(12)
delete(8)
find(12)

If you actually deleted 8, then find(12) would return false without lazy deletion.

d) **(3 points)**
Fill in this table with the worst-case asymptotic, tight bound, running time of each operation when using the data structure listed, using the "Big-O" notation. For insertions, assume the data structure has enough room (do not include any resizing costs).

	insert(e)	**find(e)**
Unsorted Array	O(1)	O(N)
Sorted Array	O(N)	O(logN)
Min Heap in an Array	O(logN)	O(N)

10) Problem Solving and Algorithms (10 points)
Given two input arrays of integers, write pseudocode to determine if one array is an exact shuffle of the other array. A shuffle is defined as containing exactly the same elements in potentially different locations. You may write java code or pseudocode. Your pseudocode should resemble real code structure and syntax.

You will solve this question 2 times.

- First, optimize your solution to have an efficient runtime. You do not have to consider space complexity for this solution.
- Second, optimize for minimum auxiliary space. You do not have to consider runtime complexity for this solution.

In either case, you may modify the input lists if that helps you. You may assume that no parameters are null and you may assume that neither of the two input arrays are empty. You may make no other assumptions about the lengths of the lists or the elements in the lists.

Examples:
```
int[] list1 = {1, 2, 3, 4, 5};
int[] list2 = {2, 3, 1, 5, 4};
isShuffle(list1, list2);  // returns true

int[] list1 = {1, 2, 2};
int[] list2 = {2, 2, 1};
isShuffle(list1, list2);  // returns true


int[] list1 = {1, 2};
int[] list2 = {2, 3, 1, 5, 4};
isShuffle(list1, list2);  // returns false


int[] list1 = {1, 3, 2, 4};
int[] list2 = {2, 3};
isShuffle(list1, list2);  // returns false
```

WRITE YOUR ANSWERS ON THE NEXT PAGE

a) (5 points) Solution 1 (optimize for runtime):

```
boolean runtimeOptimizedIsShuffle(int[] list1, int[] list2)  {
   Map<Integer, Integer> counts = new HashMap<Integer, Integer>();
   for (int value : list1) {
      if (!counts.containsKey(value)) {
         counts.put(value, 0);
      }
      counts.put(value, counts.get(value) + 1);
   }
   for (int value : list2) {
      if (!counts.containsKey(value)) {
         return false;
      }
      counts.put(value, counts.get(value) – 1);
   }
   for (int value : counts.keySet()) {
      if (counts.get(value) != 0) {
         return false;
      }
   }
   return true;
}
```

O(N) space and O(N) runtime

b) (5 points) Solution 2 (optimize for space):

```
boolean spaceOptimizedIsShuffle(int[] list1, int[] list2) {
   Arrays.sort(list1);
   Arrays.sort(list2);
   return Arrays.equals(list1, list2);
}
```

O(1) space and O(N log(N)) runtime

An alternate cute solution that doesn't modify the lists:

```
boolean spaceOptimizedIsShuffle(int[] list1, int[] list2) {
   for (int domainValue : list1) {
      int count1 = 0;
      for (int i = 0; i < list1.length; i++) {
         if (list1[i] == domainValue) {
            count1++;
         }
      }
      int count2 = 0;
      for (int i = 0; i < list2.length; i++) {
         if (list2[i] == domainValue) {
            count2++;
         }
      }
      if (count1 != count2) {
         return false;
      }
   }
   return list1.length == list2.length;
}
```

O(1) space and O(N^2) runtime

You're done! Yay!