CS 3100 Quiz Day 2

This packet contains the quizzes for this quiz day. This **cover sheet** is here to provide instructions, and to cover the questions until the quiz begins. **do not remove this cover sheet** until your proctor instructs you to do so.

You will have the entire class period to complete these quizzes. Each quiz is two pages (front and back of one sheet of paper) worth of questions. Make sure to **write your name and computing id at the top of each individual quiz**.

When you are done, you will come to the front of the room and cut off the staple to this quiz booklet. Afterward, you will discard this cover sheet and submit each quiz separately in a different pile. The proctors will be available at the front of the room to clarify this if you have any questions.

This quiz is CLOSED text book, closed-notes, closed-calculator, closed-cell phone, closed-computer, closed-neighbor, etc. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly. Please sign the honor pledge below.

In theory, there is no difference between theory and practice.
But, in practice, there is.

THIS COVER SHEET WILL NOT BE SUBMITTED. DO NOT PUT WORK YOU WANT GRADED ON THIS PAGE

Quiz - Module 1: Basic Graphs

Name ______________________________

1. [8 points] Answer the following True/False questions regarding *graphs and their basic algorithms*.

Adjacency Matrices always use exactly $\Theta(\|V\|^2)$ space.	**True**	**False**
Adjacency Matrices can be used to find all incoming edges to a node, but it always takes $\|V\|$ iterations	**True**	**False**
Adjacency Lists save memory, and you can look up the existence of an edge in $\Theta(1)$ time.	**True**	**False**
When running *BFS*, it is possible for a node to change state from *black* to *gray*	**True**	**False**
Changing *BFS* to use a *stack* instead of a *queue* will turn the algorithm into *DFS*	**True**	**False**
It is possible for a graph to have multiple unique *Topological Sorts*	**True**	**False**
Our *Topological Sort* algorithm worked by running DFS, and then sorting the nodes by reverse *start time*	**True**	**False**
When traversing a graph's directed edges, once you leave *strongly connected component*, it is never possible to return	**True**	**False**

2. [6 points] List the runtimes for the following algorithms from class. Be as precise as you can.

Algorithm	**Implementation Notes**	**Runtime**
Breadth-First Search	N/A	
Depth-First Search	N/A	
Depth-First Search	Finding ALL possible paths to a destination nodes (e.g., board games)	
Topological Sort	DFS Method	

This question is about the proof of correctness for *Breadth-First Search*. Answer each question as clearly as you can.

3. [3 points] Lemma 1 stated that if $G = (V, E)$, $e = (u, v) \in V$, and $s \in V$, then it must be true that $\delta(s, v) \leq \delta(s, u) + 1$. Explain why this inequality holds in your own words.

4. [3 points] Later in our proof, we saw that $v.d \geq \delta(s, v)$. In your own words, explain why this inequality is useful.

NOTHING BELOW THIS POINT WILL BE GRADED

Quiz - Module 2: Advanced Graphs

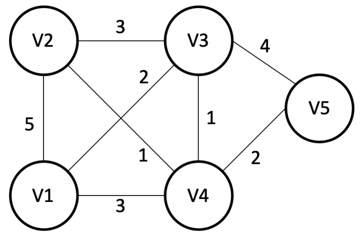
Name ______________________________

1. [8 points] Answer the following True/False questions regarding *graphs and their algorithms*.

Dijkstra's Algorithm selects the next node to become known by examining the least cost edge from any unknown node to any known node.	**True**	**False**
In *Dijkstra's Algorithm*, each node v stores one predecessor node (the node that comes before v on the optimal path to v)	**True**	**False**
When solving the *Flights (Min Layover Time)* problem in class, we decided to model the flights as nodes and layover waits as edges	**True**	**False**
In your homework, you solved the *Robot Problem*. We solved this by making a new state-space graph, but this graph had $\Theta(\lvert V\rvert^3)$ nodes relative to the original Graph.	**True**	**False**
Kruskal's Algorithm runs in $\Theta(E \log V)$ if the find-union data structure is optimized	**True**	**False**
When solving the *Wiring* homework problem, edges between outlets and switches could be removed because they could not possibly be in the *MST*	**True**	**False**
The number of edges in any *MST* of a connected graph G is exactly $\lvert V\rvert - 1$	**True**	**False**
If all of a Graph's edges have weight 1, then the graph has more than one *MST*	**True**	**False**

2. [3 points] Briefly explain, in your own words, what the primary advantage of an *indirect heap* is. More specifically, what operations are improved and how does the data structure accomplish this?

3. [3 points] Consider the graph in the box to the left. Draw the *MST* of the graph in the middle box. In the right box, list the *order in which the nodes become known* if *Prim's Algorithm* is executed starting at V_1

Original Graph	**MST**	**Order Nodes are Added**
 V1 V2 V3 V4 V5 3 4 2 5 1 1 2 3		

4. [6 points] Complete the implementation of the find-union's *union* method using *union by rank* below. Fill in the code in each of the blanks to accomplish this.

```
Union(x,y):

  Link(find(x), find(y))


Link(x,y):

  if x.rank > y.rank:

    ___.parent = ___                       //x or y

  else

    ___.parent = ___                       //x or y

    if x.rank ___ y.rank:                  //<, >, or ==

      ___.rank = ___.rank + 1              //x or y
```

NOTHING BELOW THIS POINT WILL BE GRADED

Quiz - Module 3: Divide and Conquer

Name ______________________________

1. [8 points] Answer the following True/False questions regarding.

In *closest pair of points*, it is possible for all of the points to fall within the *strip*	**True**	**False**
When solving *closest pair of points*, we check the next seven points in the strip, but specifically the next seven on the opposite side of the divide.	**True**	**False**
One way to sort the list of points by y in our *closest pair of points* algorithm is to use mergesort's merge algorithm as the recursion returns up.	**True**	**False**
Quickselect invokes *median of medians* to find a good pivot, and *median of medians* also invokes *quickselect* to find the median of the medians.	**True**	**False**
Invoking *quickselect* from *quicksort* to find a good pivot gaurantees the pivot will be in the middle 40 percent of the list, but it may not be the exact median.	**True**	**False**
The runtime of *quickselect* is $\Theta(n)$	**True**	**False**
When solving *maximum subarray*, it is gauranteed that the three solutions (left, right, and across the divide) are mutually exclusive (they do not share any array elements)	**True**	**False**
Our *maximum subarray* solution ran in $\Theta(n^2)$ time, because our combine step was quite slow.	**True**	**False**

2. [6 points] Use the master theorem to establish the runtime of the following three recurrence relations. You MUST list the case that you used as well as the big-theta runtime. The master theorem is listed on the back of this quiz.

Recurrence	Case Used	Θ Runtime
$T(n) = 2 * T(\frac{n}{4}) + 1$		
$T(n) = 2 * T(\frac{n}{4}) + n^2$		
$T(n) = 2 * T(\frac{n}{4}) + \sqrt{n}$		

This problem will ask you to develop a divide-and-conquer algorithm in steps. **Problem statement**: You are given an array $A[]$ containing n unique integers that contains a *peak* (e.g., $3, 7, 9, 2, 1$). In other words, the elements increase monotonically until reaching the peak, and then decrease monotonically afterwards (Note that the peak may be the first or last element). Return the peak value in $\Theta(\log n)$ time.

3. [2 points] First, describe the base case (when $n = 1$). What do you return?

4. [2 points] Next, describe how you *divide* the problem into sub-problem(s). Also describe how/why you might return immediately at this step.

5. [2 points] Last, describe the recursive call(s) that you make and how the result(s) is/are used to get the solution to the overall problem.

NOTHING BELOW THIS POINT WILL BE GRADED

MASTER THEOREM: FOR YOUR REFERENCE

For a recurrence of form $T(n) = aT(\frac{n}{b}) + f(n)$, let $k = \log_b a$

- Case 1: if $f(n) = O(n^{k-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^k)$
- Case 2: if $f(n) = \Theta(n^k)$, then $T(n) = \Theta(n^k \log n)$
- Case 3: if $f(n) = \Omega(n^{k+\varepsilon})$ for some constant $\varepsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Quiz - Module 4: Greedy Algorithms

Name ______________________________

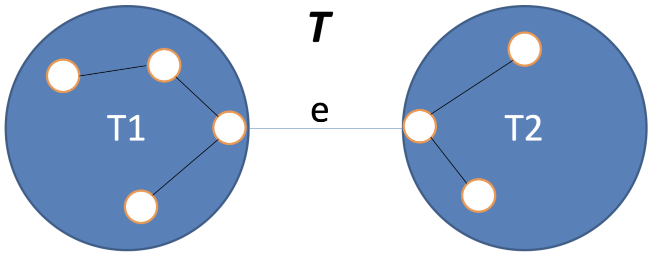
1. [8 points] Answer the following True/False questions regarding.

A problem that has *optimal substructure* always has a correct *greedy algorithm*.	**True**	**False**
optimal substructure exists when small optimal solutions of size n can be extended to an optimal solution of size $n + 1$.	**True**	**False**
Our greedy solution to the *Fractional Knapsack Problem* will always fill the knapsack completely full if it can (there is never a reason to leave extra space in the knapsack).	**True**	**False**
Our greedy algorithm for *Coin Change* always selected the highest value coin that was possible to issue.	**True**	**False**
Our greedy algorithm for *Coin Change* still works if the coin values are 10, 7, and 1 cents respectively.	**True**	**False**
For *activity selection*, we used a greedy choice of earliest finish time. Latest start time seems similar, but does NOT work because that activity could end at any time in the distant future.	**True**	**False**
Our greedy algorithm for *activity selection* runs in $\Theta(n \log n)$ time because we need to sort the input.	**True**	**False**
The *Greedy Choice* for the *bridge crossing problem* involved getting the *two slowest* walkers across the bridge as quickly as possible.	**True**	**False**

2. [3 points] Explain what it means for an algorithm to have the *Greedy Choice Property*. Explain in the context of the *Coin Change Problem*: What was the greedy choice and what does it mean that that this choice has the *Greedy Choice Property*.

3. [3 points] Provide a counter-example to the following claim: *The greedy algorithm of selecting the item with the highest value to weight ratio still works when applied to the discrete knapsack problem*. Your counter-example should use three items. Make sure to list their weights, values, and ratios of weight-to-value.

4. [6 points] Prove that the *Minimum-Spanning Tree Problem* has *optimal substructure*. Specifically, suppose you have a graph G (input) and a spanning tree T. Now suppose that T contains an edge e ($e \in T$). We can think of T as being made up of three parts: e, and the two smaller spanning trees that e connects together, T_1 and T_2.



Prove the following claim: If T is a minimum-spanning tree for G, then T_1 must be a minimum spanning tree for that portion of the graph. *Hint: Use a proof by contradiction like we did in class. If T_1 is NOT optimal, then what must be true?*

NOTHING BELOW THIS POINT WILL BE GRADED