CSE 373 Final Review

June 2, 2017

James Wang, Justin Sievers, Kimberly Bautista

Exam Info

- Tuesday, Jun 6
- 2:30 – 4:20 pm
- SMI 120

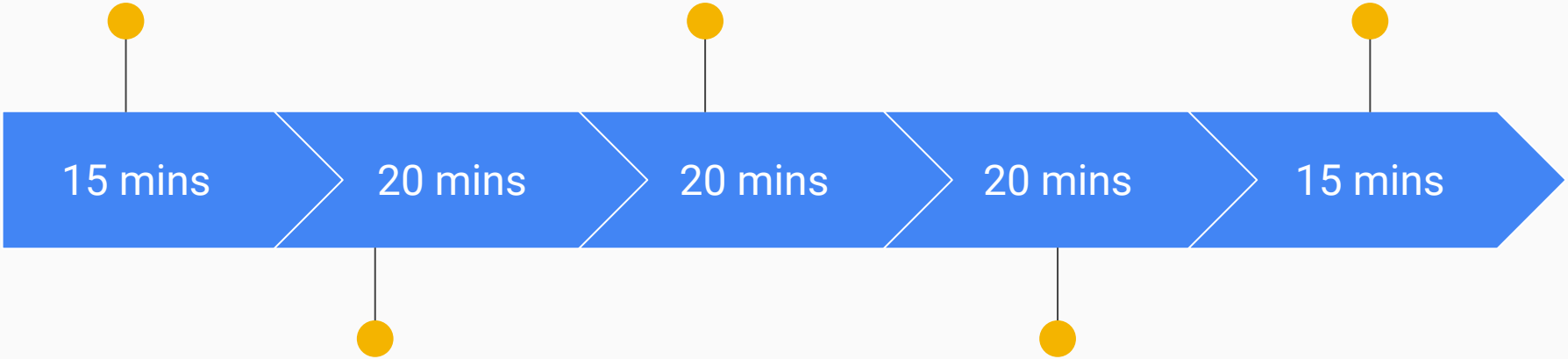
Exam Topics

- Stacks and Queues
- Heaps/Priority Queues
- Algorithm Analysis
- Dictionaries
- BSTs, AVL Trees
- Hashing
- Graphs, Union Find
- Sorting
- Algorithm Design

Review of Pre-Midterm Material: AVL/Hash Table

Sorting Algorithms

Algorithm Design



Graph Searching Algorithm and MST

Big-O and Recurrences

Pre-Midterm Material

AVL Tree

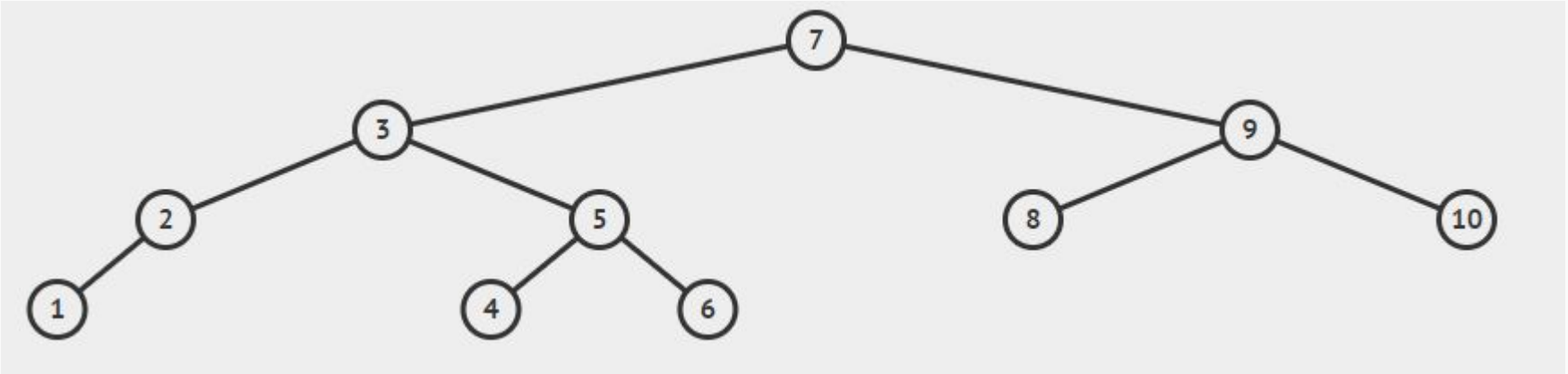
Quick Review:

- AVL is a BST that has additional constraints, aka AVL property
- AVL Property: For each node, the left and right subtrees height cannot differ by more than 1

Easy Challenge:

Insert [10,9,8,7,6,5,4,3,2,1] into an AVL and show the final tree!

AVL Solution



Hash Table

Quick Review:

- Hash Table: Buckets of elements resulting from a Hash Function
- Linear Probing, when the hashing bucket is full, move to next available bucket
- Quadratic Probing: When the hashing bucket is full, find next bucket quadratically
- Separate Chaining: Chain elements into same bucket

Easy Challenge:

1. Explain the importance of load factor when resolving collision using linear or quadratic probing?
2. What is the worst case when using separate chaining? How can this be resolved?

Hash Table Solution

1. Load factor is an easy way to determine the likelihood of clustering, resulting in slow runtime. Also, for quadratic probing, a load factor over .5 can result in failures to insert.

In terms of implementing a hash table, you can look at the current load factor and resize the array if it's higher than the ideal one

2. In the worst case, all of the elements will be in one bucket. If buckets are linked lists, then find will be O(n) and insert can be O(1). If buckets are AVL trees, find and insert are O(log(n)).

When this worst case happens, rehashing with a new hash function is extremely likely to fix the problem.

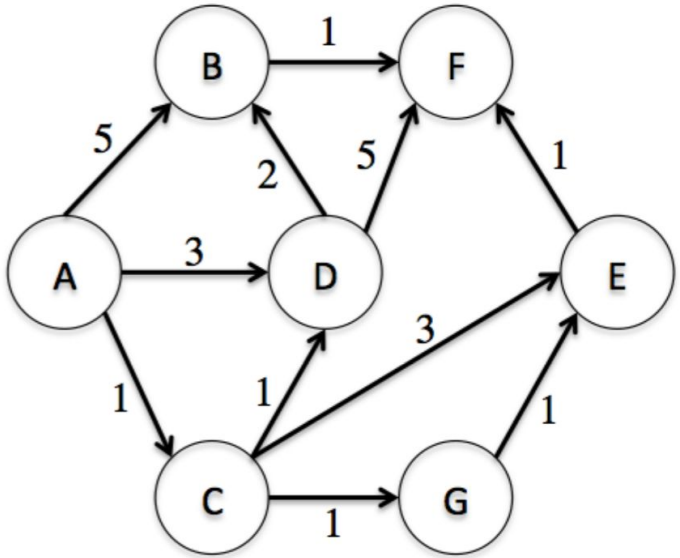
Graph Algorithms

Graphs

Quick Review

- Make sure you understand the difference between unweighted/weighted graph and directed/undirected graph
- Dijkstra's Algorithm: guarantee to find the shortest path given a graph with nonnegative weights.

Use Dijkstra's algorithm to provide the shortest path between nodes A and F. Show each of your steps. In your work, indicate in which order vertices are added to the known set.



Graphs Solution

Starting node -> A

	A	B	C	D	E	F	G
Visited?	x		x				
Shortest Path from A	0		1				
From?	A		A				

Graphs Solution

Starting node -> A

	A	B	C	D	E	F	G
Visited?	x		x				x
Shortest Path from A	0		1				2
From?	A		A				C

Graphs Solution

Starting node -> A

	A	B	C	D	E	F	G
Visited?	x		x	x			x
Shortest Path from A	0		1	2			2
From?	A		A	C			C

Graphs Solution

Starting node -> A

	A	B	C	D	E	F	G
Visited?	x		x	x	x		x
Shortest Path from A	0		1	2	3		2
From?	A		A	C	G		C

Graphs Solution

Starting node -> A

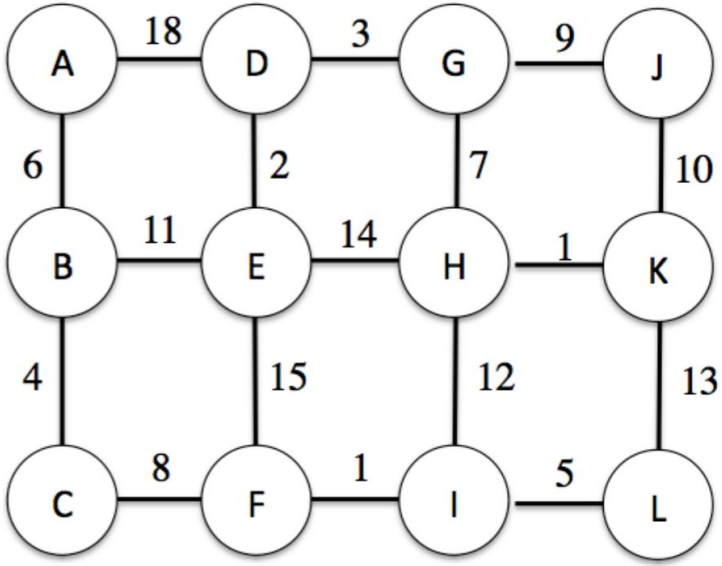
	A	B	C	D	E	F	G
Visited?	x		x	x	x	x	x
Shortest Path from A	0		1	2	3	4	2
From?	A		A	C	G	E	C

MST

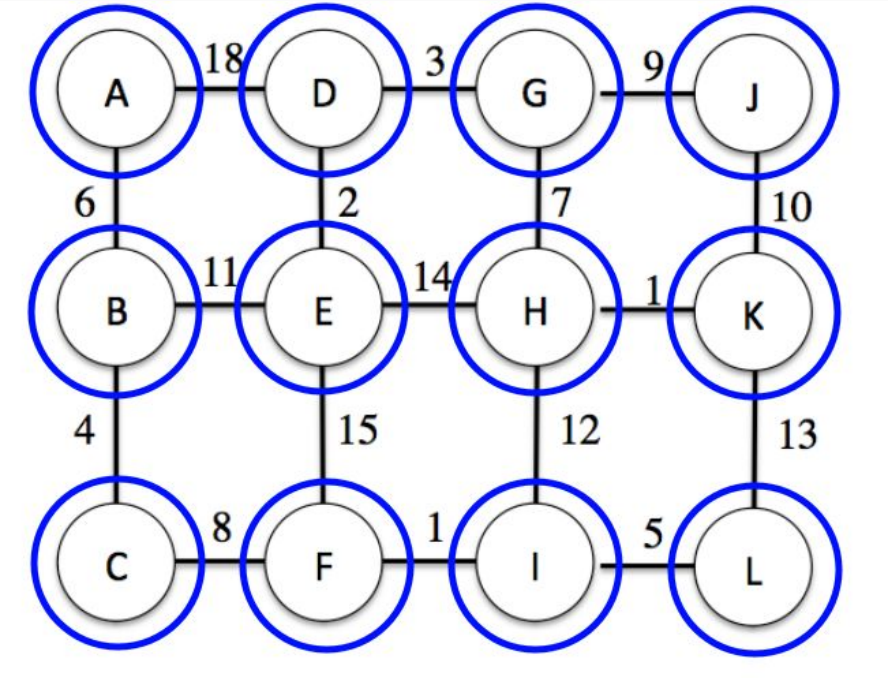
Quick Review

- Union-Find ADT: A data structure that keeps track of elements in disjoint sets.
- Prim's Algorithm: An algorithm that finds MSTs by modifying Dijkstra's to only account for the single edge cost, not the total path cost when connecting a node.
- Kruskal's Algorithm: An algorithm to find MSTs by traversing the edges and using the Union-Find ADT.

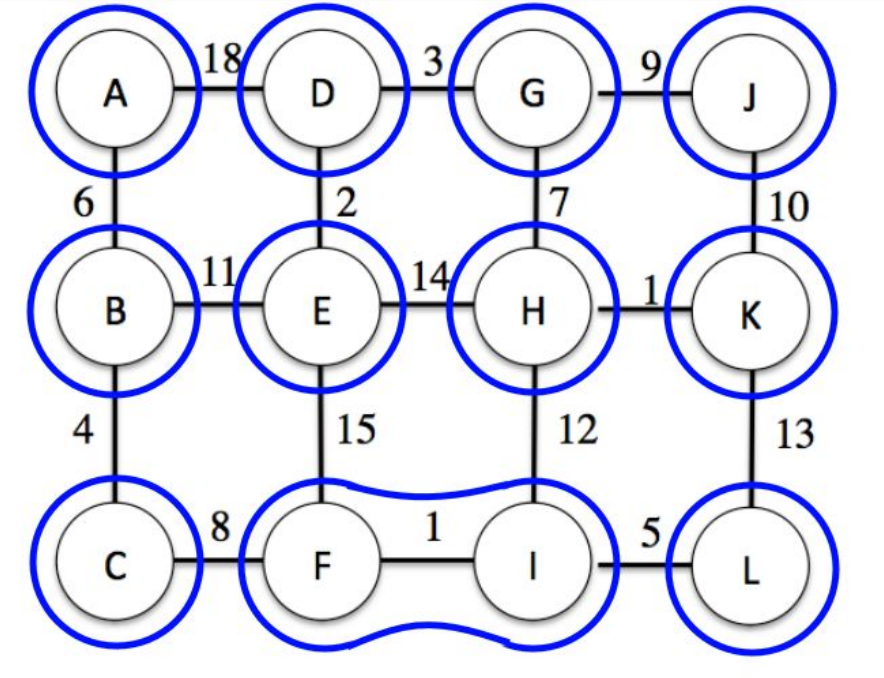
Find the minimum spanning tree of the following graph using Kruskal's algorithm. Provide your ordering of edges. If an edge cannot be added to the MST, provide the cycle that would be created if that edge were added.



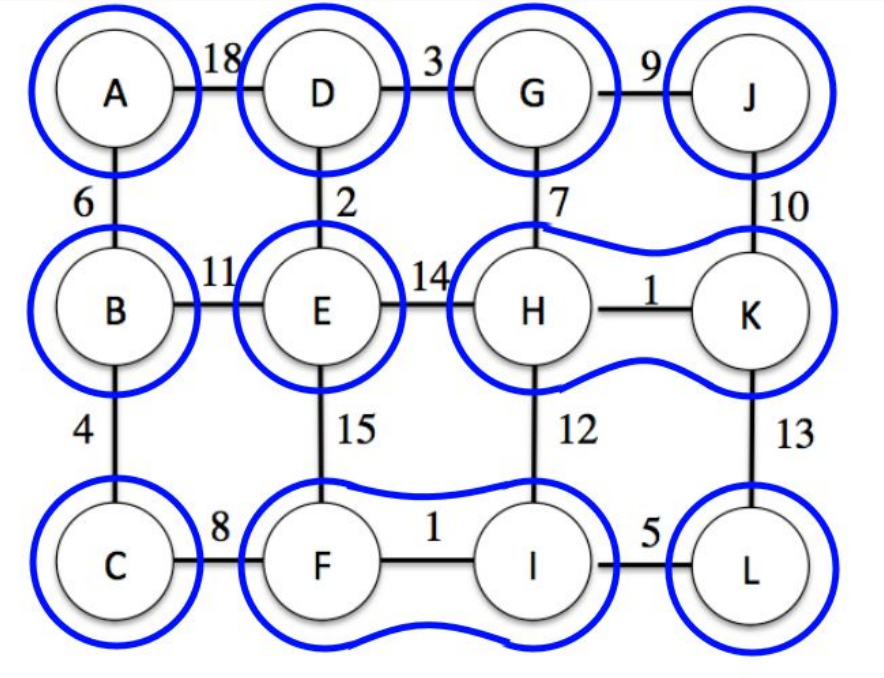
MST Solution



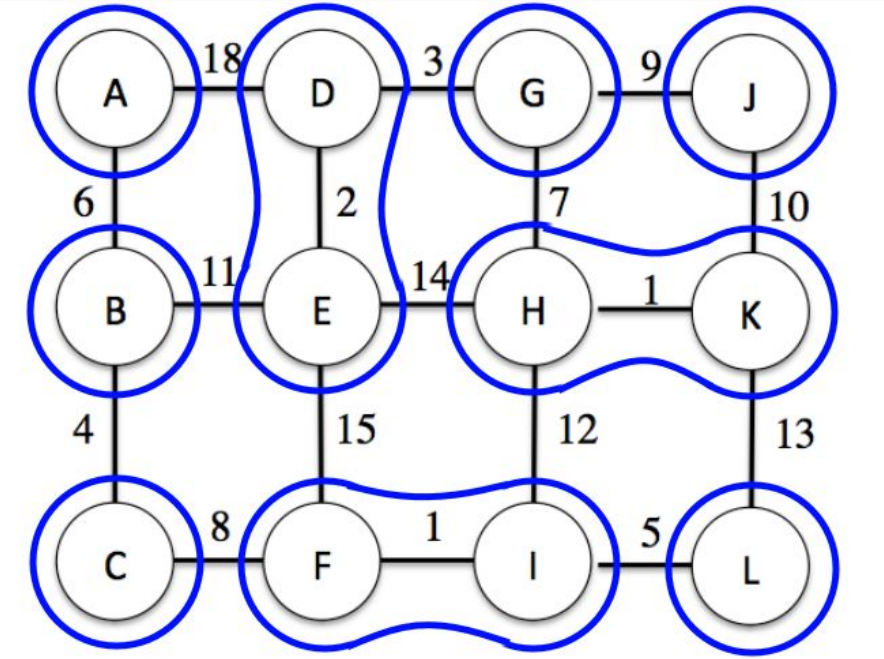
MST Solution (Cont.)



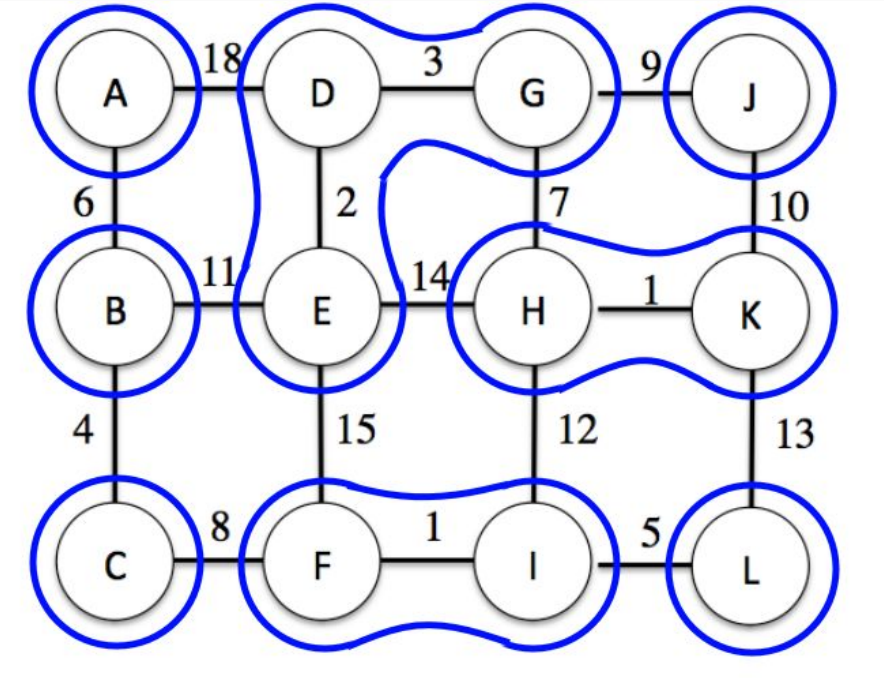
MST Solution (Cont.)



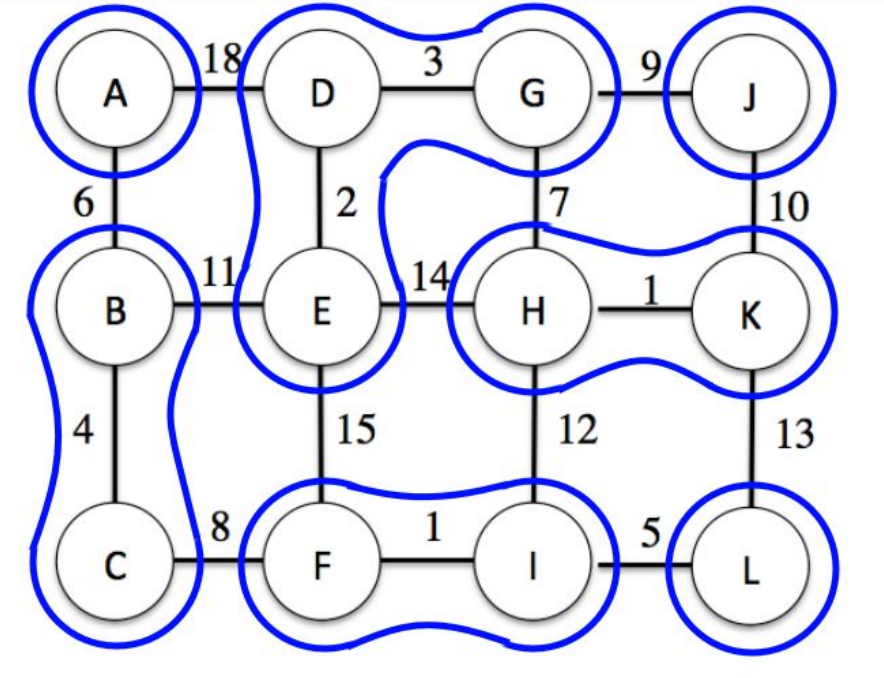
MST Solution (Cont.)



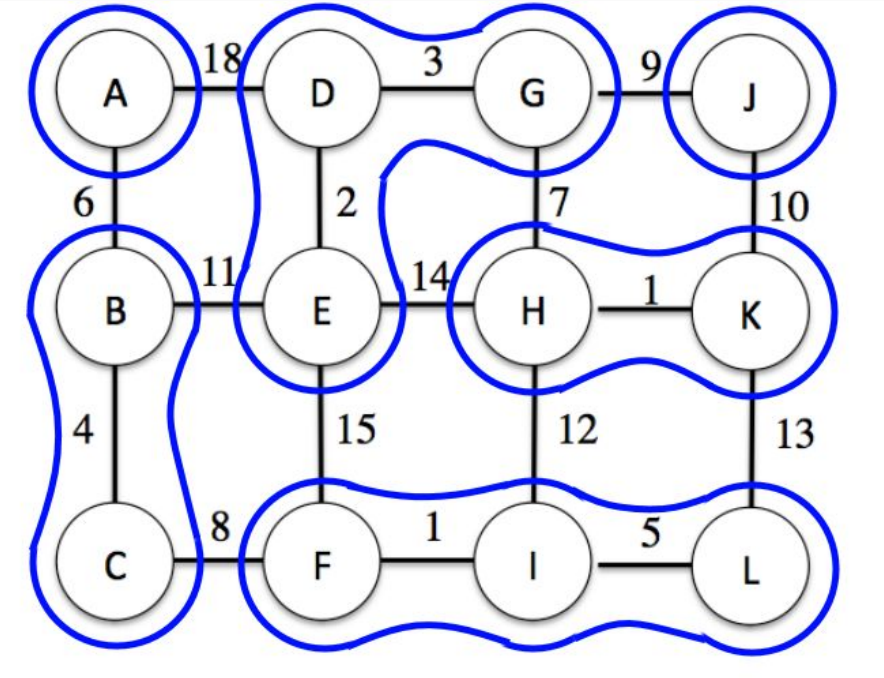
MST Solution (Cont.)



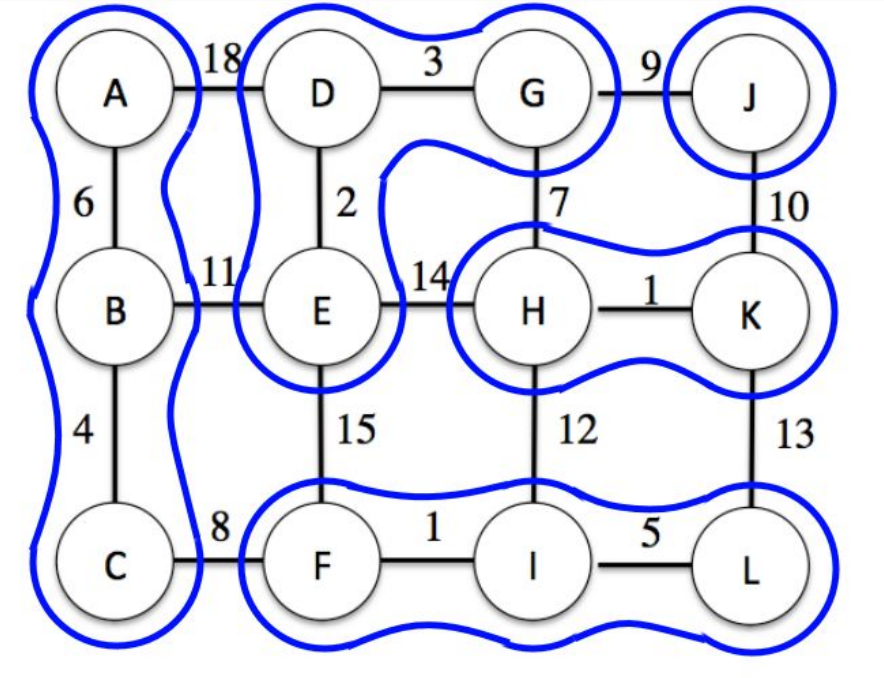
MST Solution (Cont.)



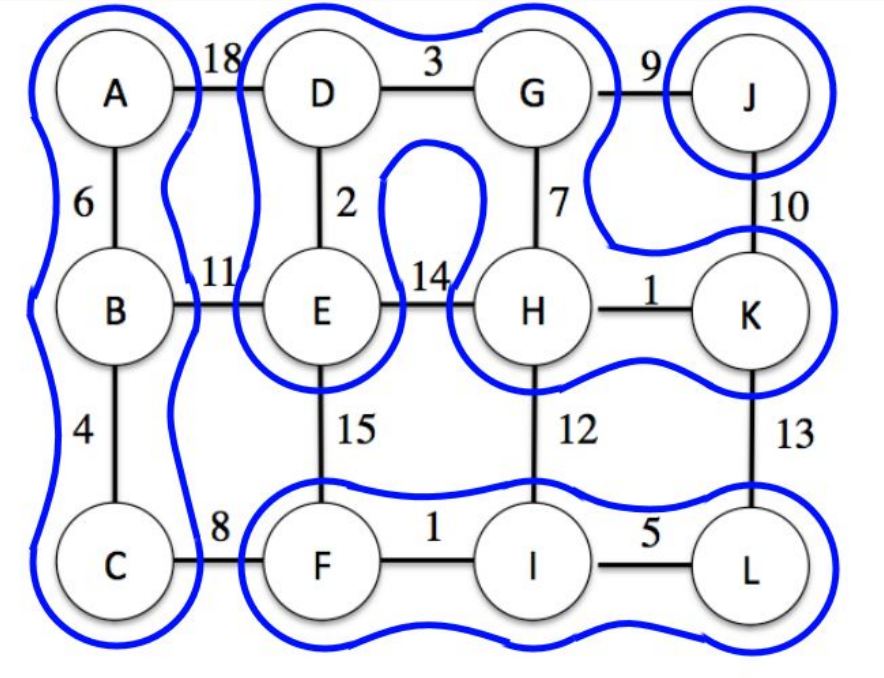
MST Solution (Cont.)



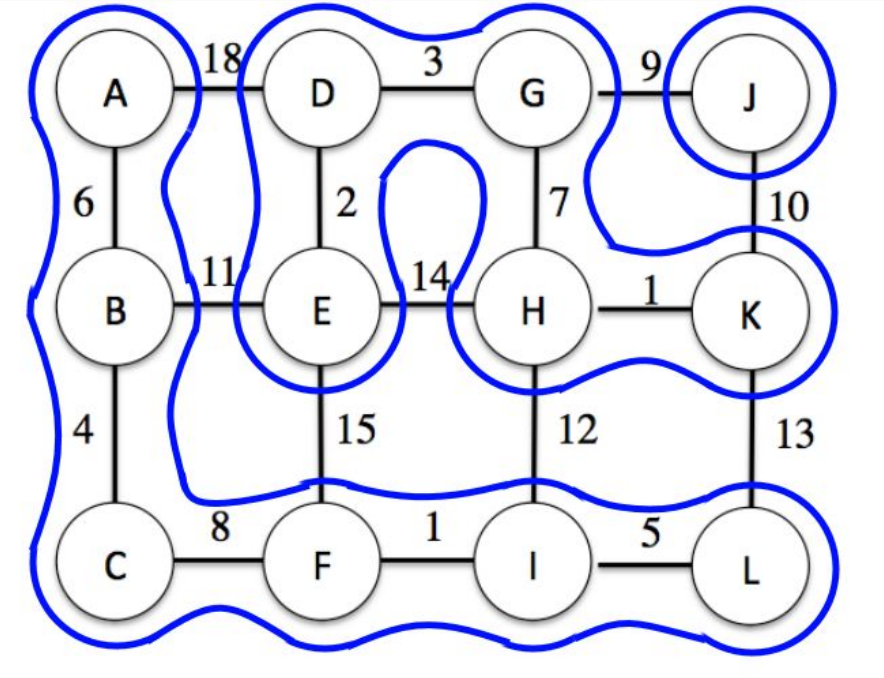
MST Solution (Cont.)



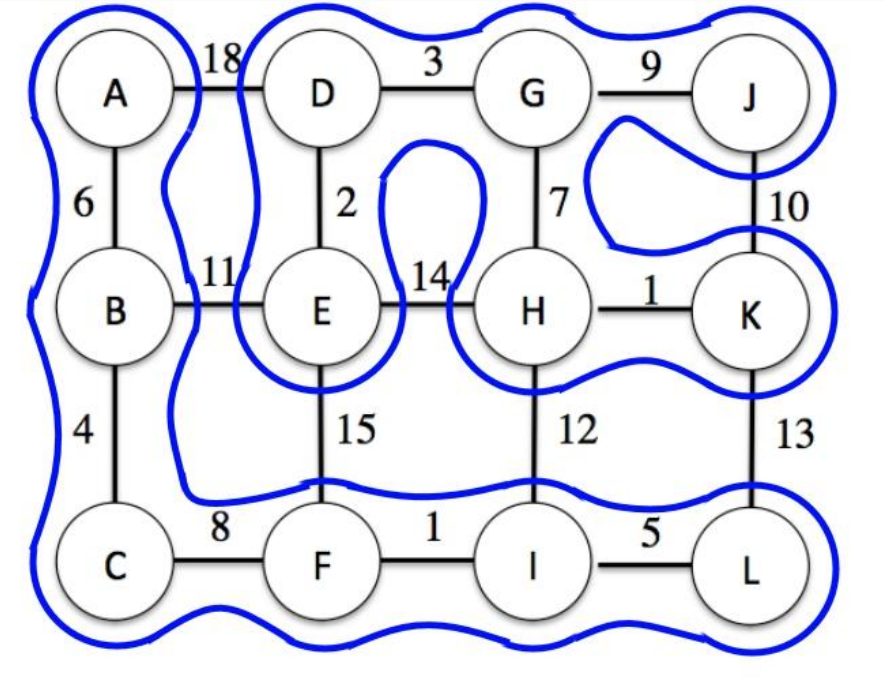
MST Solution (Cont.)



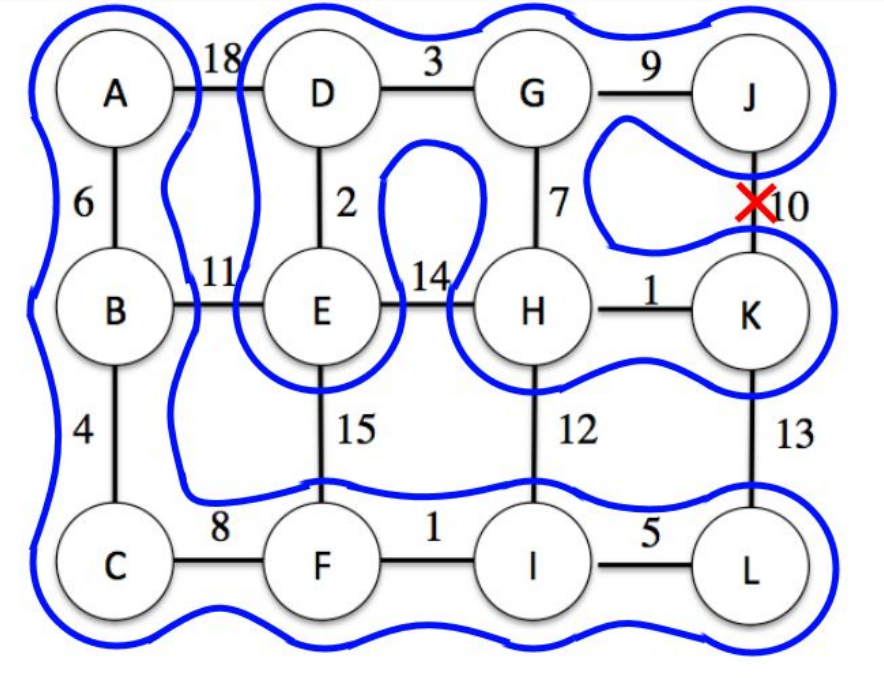
MST Solution (Cont.)



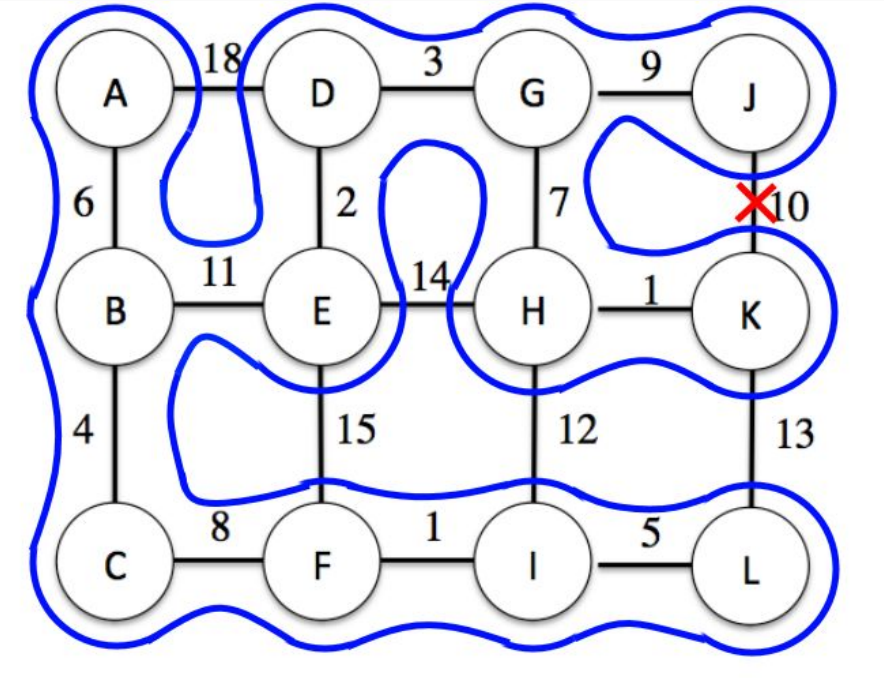
MST Solution (Cont.)



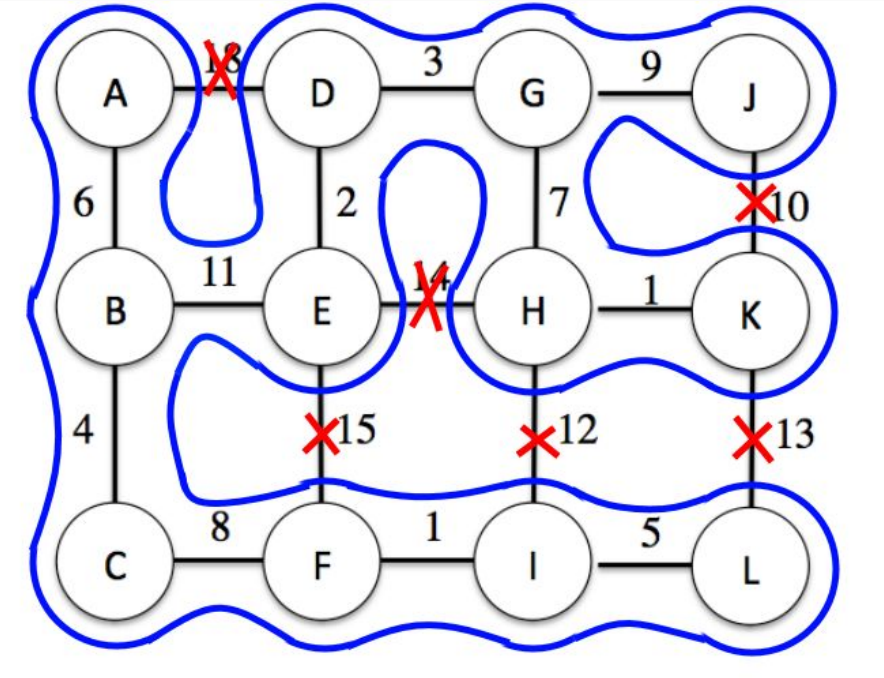
MST Solution (Cont.)



MST Solution (Cont.)



MST Solution (Cont.)



MST Solution (Cont.)

Edges used in MST (in the order they were added):
{F, I}, {H, K}, {D, E}, {D, G}, {B, C}, {I, L}, {A, B}, {G, H}, {C, F}, {G, J}, {B, E}

Along the way, {J, K} would have created a cycle that goes J->K->H->G->J, so it was not added.
Any edges examined after {B, E} would have also created cycles.

Sorting Algorithms

Comparison Sorts

Quick Review

- Quick Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Heap Sort

Concept Check:

- Given a list of 10 numbers, say if all I want to do is to get the top 3 sorted elements, which algorithm can achieve this in the most effective manner?

Comparison Sort Solutions

Solution 1:
Heap sort would be an appropriate sorting algorithm! Since as you place everything into a heap, you'll be able to just take out the first k element, thus achieving what the problem is asking!

Solution 2:

Selection sort would be also be another appropriate answer here, since as mentioned in lecture, it is an interruptible sorting algorithm. Since after you do the first k-pass of selection sort, the first k elements would be the sorted elements that you want

Non-comparison-based Sorts

Quick Review:

- Radix Sort
- Bucket Sort

Simple Problem:

Use Radix Sort to sort the following problem.
You only need to do the first two passes

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

Non-comparison-based Sort Solution

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

Big-O and Recurrences

Analysis Review

Quick review:

-O(k) is an upper bound

-Ω(k) is a lower bound

-Θ(k) is O(k) and Ω(k): a “tight” bound.

-Recurrences are recursive definitions of functions, usually of their runtime

Sample problem: Find a tight big O bound on the following recurrences.

1. T(n) = {

1	if n <= 1
T(n-2) + 100	otherwise

}

2. T(n) = {

1	if n <= 1
3T(n/2) + 1	otherwise

}

Recurrence Solution

1. Unrolling this recurrence:

$T(n) = 100 + T(n-2)$
$= 100 + 100 + T(n-4)$
$= 100 + 100 + 100 + T(n-6)$
$= \ldots$
$= O(100(n/2) + 1) = O(n)$

2. Unrolling this recurrence:

$T(n) = 1 + 3T(n/2)$
$= 1 + 3(1 + 3T(n/4))$
$= 1 + 3 + 9T(n/4)$
$= 1 + 3 + 9 + 27T(n/4)$
$= \sum_{i=0}^{\log(n)} 3^i = \frac{1}{2}\left(3^{\log(n)+1} - 1\right)$
$= O(3^{\log(n)})$
$= O(n^{\log(3)})$

Note that we can show that $k^{\log(n)} = n^{\log(k)}$ for all constants k by taking the log of both sides.

Common Summations to Solve Recurrences

$$\sum_{i=0}^{n} i = 0 + 1 + 2 + \ldots + (n-1) + n = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{n} x^i = x^0 + x^1 + x^2 + \ldots + x^{n-1} + x^n = \frac{1-x^{n+1}}{1-x}$$

Method Analysis

Find a big O bound on the method shown:

```
public void whee(int n) {
    for (int i = 1; i < n; i *=2) {
        for (int j = 1; j < n; j*= 3) {
            System.out.println("WHEE!");
        }
    }
    for (int k = n/2; k < n; k++) {
        System.out.println("WOAH!");
    }
}
```

Method Analysis Solution

The two first nested loops are each log(n), and the second outer loop will have runtime n/2.

Thus, the algorithm is O(log(n)log(n) + n/2) = O(n).

```
void whee(int n) {
    for (int i = 1; i < n; i *=2) {
        for (int j = 1; j < n; j*= 3) {
            System.out.println("WHEE!");
        }
    }
    for (int k = n/2; k < n; k++) {
        System.out.println("WOAH!");
    }
}
```

Algorithm Design

Algorithm Design Solution #1 (No number theory required)

How to approach this problem?

- Look at what's given to you in the problem description
  - "...if two numbers are both prime and not equal to each other, then they have no common factors."
  - "...if a number k is composite (not-prime) then it must have one factor that is at most √k. "

Algorithm Design Solution #1 (No number theory required)

How to approach this problem?

- Look at what's given to you in the problem description
  - "...if two numbers are both prime and not equal to each other, then they have no common factors."
  - "...if a number k is composite (not-prime) then it must have one factor that is at most √k. "

Algorithm Design Solution #1 (No number theory required)

Pseudocode

Suppose without loss of generality (WLOG) that i <= j → min{√i, √j} =√i

```
For (each integer k = 2 to k<= √i) {
    If ( i % k == 0) { // if k is a factor of i
        // check if k or its complementary factor (made up this word) are factors of j
        If (j % k == 0 || j % (i/k) == 0)
            Return true
    }
}
Return false // if there's no common divisor of i and j
```

Algorithm Design Solution #2 (Number theory required)

Idea: Find the greatest common divisor (GCD) of i and j

The Euclidean Algorithm:

- This algorithm will return to you the GCD which is a common factor of i and j, so you can return true once you find this
- How it works?
  - Keep subtracting the smaller number from the bigger number until both numbers i, j are equal to each other or one of them equals 1

Euclidean Algorithm

Example: Let i = 42, and j = 105. Let's say we already know that GCD(i,j) = 21

Now let's test it out against the Euclidean Algorithm:

i	j	smaller?
42	105	i is smaller so subtract i from j
42	63	i is again smaller, so j = j - i
42	21	j is smaller, so i = i - j (Note: We don't stop subtracting until i == j or either one of them gets to 1)
21	21	Done! i == j so the GCD is 21

Euclidean Algorithm

Example: Let i = 42, and j = 105. Let's say we already know that GCD(i,j) = 21

Now let's test it out against the Euclidean Algorithm:

i	j	smaller?
42	105	i is smaller so subtract i from j
42	63	i is again smaller, so j = j - i
42	21	j is smaller, so i = i - j (Note: We don't stop subtracting until i == j or either one of them gets to 1)
21	21	Done! i == j so the GCD is 21

Euclidean Algorithm

Example: Let i = 42, and j = 105. Let's say we already know that GCD(i,j) = 21

Now let's test it out against the Euclidean Algorithm:

i	j	smaller?
42	105	i is smaller so subtract i from j
42	63	i is again smaller, so j = j - i
42	21	j is smaller, so i = i - j (Note: We don't stop subtracting until i == j or either one of them gets to 1)
21	21	Done! i == j so the GCD is 21

Euclidean Algorithm

Example: Let i = 42, and j = 105. Let's say we already know that GCD(i,j) = 21

Now let's test it out against the Euclidean Algorithm:

i	j	smaller?
42	105	i is smaller so subtract i from j
42	63	i is again smaller, so j = j - i
42	21	j is smaller, so i = i - j (Note: We don't stop subtracting until i == j or either one of them gets to 1)
21	21	Done! i == j so the GCD is 21

Algorithm Design Solution #2 (Number theory required)

Pseudocode

At each iteration, subtract the smaller number from the bigger number until both numbers i, j are equal to each other or one of them equals 1 (This is the Euclidean Algorithm)

- If i = 1 or j = 1 // This means that there's no GCD
  Return False

- If i == j // This means that there's a GCD
  Return true

Motivation Slide - “Relax, you are here to learn!”

Before we end this review session, here’s the motivation slide!

You might not be aware, but here are the things that you’ve accomplished this quarter!

1. Learned about a variety of data structures that are commonly used in everyday life
2. Learned how to fully analyze a piece of code written by someone else
3. Learned about effective sorting algorithms
4. Started your journey to become an advanced programmer :D

As long you try your best and put in effort, you should be proud of your results!

Grades are merely a number, it does not define you as a person.

Last Minute Advice

- Make sure you bring your ID, in the event that we decide to check ID!
- If you get stuck on a problem, skip it and come back later.
- For problems that allow partial credit, you should show ALL your work!
- Look through all the problems at the beginning of the exams and do the easy ones first before tackling the harder problems
- If you find the exam extremely difficult, don’t panic! This exam is meant to be HARD, so the people around you problem feel the same! So you should calm down and do your best!

Additional Questions ???
If so, please come up to the front!
Thanks for coming! Good Luck :D