Csci335 Final topics (after midterm topics):

1. Lambda functions: [=] pass global variables by value (on final, it makes a lambda function constant - **False**) [=] actually makes the global variable passed into the lambda function to not change while inside the function even if we change the global variable outside of the lambda after we first pass it in.

[&] pass global variables by reference.

1. Threads – allowing parallelism t1.join() = wait for program to finish
2. Mutex – is a lock that locks the access to shared memory – global variables
3. Function pointers
4. Hashing:
5. Hash functions to process integers
6. Hash function string encoding to handle strings  
   Collision can only be minimized, cannot be removed

Collision Solution Techniques:

Load factor = the total items stored / the size of the table, smaller LF the better

1. Open address (closed hashing):

* Linear probing: find the next available spot
* Quadratic probing
* Double hashing: h(x) = x mod 11, h2(x) = 8 – (x mod), [h(x) + i\*h2(x)]%m, m is the size of the table

1. Separate chaining (open hashing)

* Linkedlist

Graphs: BFS DFS and dijsktra traversal

Code up lambda

Heap insert or delete something

1. Graphs:

Kruskal’s algorithm is a simple algorithm for finding a minimum spanning tree: in a spanning tree every node is connected to every other node, so the tree spans all of the nodes. Removal any single edge from this spanning tree causes the graph to be unconnected.  
  
A spanning tree is minimum is there’s no other spanning tree with smaller cost.

If the graph is unweighted, then the cost is just the number of edges (\_\_, \_\_\_, \_\_\_, ), if 3 edges, cost = 3

If has weighted edges, then its sum of the them.

For undirected graph and minimum spanning tree – same set of vertex, but a subset of edges. 反正就是要连上所有的点，但用最小的路，不需要direction，不需要loop back

Kruskal’s algorithm:  
1. Starts with a forest, with single standalone vertex, each vertex itself is a tree.   
2. Then loops for an edge with smallest weight and connects two dots together.

3. looks for another edge and connect with another dot.  
4. When all standalone dots are connected, it stops.   
5. It forms just one tree, does NOT form a loop.

To form a new tree from standalone dots, a union operation is applied, a disjoint set. A heap is used for picking the minimum weight edge at each step.   
  
Performance:  
This algorithm takes O(|E|) steps to build heap because the while loop iterates the worst case once for each edge in the graph. To build the heap because it must read in the edge set, which is |E|, cardinality of the set into the array first. Each deleteMin() is log|E| because it’s stored in a min heap, min is always returns, and it does an internal sorting, which is log|E|, after root is deleted, last element is swapped to the front and trickle down to compare with its children. logn

Note: |E| = O(|V^2|) because ONE unique edge connects a PAIR of vertices.

|E| = O(|V^2|)  
log(|E|) = log(|V|^2)  
so kruskal’s O(|E|log|E|)

**Topics not covered:**

**Topological sort**: is directed Acyclic graph, with arrows and weights, no looping back.  
thus performance is O|E|+|V|), each edge is examined at most once and each vertex is processed at most once.

Indegree: ->node<-, number of incoming edges.  
outdegree: <-node->, number of outgoing edges.

**Searching** is faster in AVL tress because its strictly balanced tree.

**Insertion && deletion** is faster in red-and-black tress because it requires fewer rotations