Exam 1 topics - updated 9/28/18 @ 10pm.

* Sorting methods (selection sort, insertion sort, ~~merge sort~~) and their properties: runtime complexity, space complexity, adaptive?, stable?, advantages and disadvantages (when would you choose one over the other)
* Indirect sorting, searching with indirect sorting
* ~~Correctness of Insertion sort and selection sort.~~ Loop invariant for the inner and outer loops of insertion sort and selection sort.
* Binary search
* Asymptotic notation
  + Big-Oh, Theta, Omega and their relationship   
    Be able to prove them using the limit theorem.
  + Given code, pseudocode, or an algorithm description, be able to find the asymptotic time complexity WITH THE TABLE. - A significant number of points may be given for the table.   
    When showing the time complexity with the table method, YOU MUST SHOW THE i-TH TERM.
* Compute runtime
* Counting instructions
* Summations
* Lists
* Be able to write code on paper involving pointers, lists, nodes. Be able to compute the time complexity for your own code.   
  Codding requirements: global, external or static variables are NOT allowed in any code (in homework, quiz or exam) in this class. Using such variables will result in losing 50%-100% of the credit for that problem.   
  Draw the memory corresponding to the program state at each time (after each instruction) during the program execution. Use the **drawing conventions/style used in class**.
* Stacks, FIFO queues
* Using stacks for evaluation of arithmetical expressions: produce postfix expressions and evaluate postfix expressions.

 Exam 2 topics - updated 10/18/18

* Tree properties (these are implicit as part of solving recurrences and counting number of recursive calls)
* For all the topics below you should be able to write code and also solve on paper.
* Compute the time complexity for functions you write and for code provided to you.
* Recurrences
  + all three methods: Master theorem method, tree and table method, induction method.
  + Know (or be able to quickly deduce) the time complexity for common recurrences (given at the beginning and end of the slides)
* Recursion:
  + Know the difference between these concepts and how each is represented: recursive call, tree for recursive calls, frame stack for rec calls, tree for time complexity, space complexity.
  + Be able to draw the time complexity (recursion) tree and to compute the number of recursive calls,
  + Given a recurrence formula, write code that has that time complexity
  + Write a recursive function (i.e. be able to implement a recursive function as the solution to a given problem), **Show the execution of your program (e.g. the data after each recursive call or at the end of a loop)**
  + Trace a recursive function (e.g. "What does this program print?"),
  + Given code, be able to write the recursive formula for the **time complexity** and solve it.
  + Be able to provide reasonable upper and lower bounds for recurrence formulas that you are not able to solve.
* Dynamic programming:
  + Problems: weighted interval scheduling (a.k.a. Job Scheduling), Knapsack (unbounded, 0/1, fractional), LCS (Longest Common Subsequence), Longest Increasing Subsequence, Edit Distance, Stair Climbing (with and without gain), Matrix Traversal (with or without fish and/or obstacles)
  + Solutions: iterative (bottom-up), memoization, inefficient recursion.
  + Analyze time and space complexity for:
    - iterative solution - compute Theta
    - memoized solution - compute Theta
    - inefficient - some recurrences are too hard to solve in this case, but you should be able to give a good lower bound. For example, if the T(N) would be exponential, find show an exponential lower bound.
  + Write the cost/gain function (identify the dependence of the solution for the current problem on solutions for previous problems)
  + For small problems: compute the optimal cost/gain (the solution), recover the solution choices,
  + Brute-force search.
* Greedy techniques for:
  + Knapsack (for unbounded, bounded, 0/1, fractional), interval scheduling (for both weighted and un-weighted intervals), Huffman codes and tree.
  + What would be a Greedy solution for any of the studied dynamic programming problems? If that Greedy method does not produce an optimal solution, be able to give a counter example where it does not give the optimal solution.
  + Time and space complexity
* From exam 1 (lower priority): Asymptotic notation
  + Big-Oh, Theta, Omega and their relationship - definitions or proofs or will not be required, but you may be asked to give the time complexity of a piece of code.
  + Given code, pseudocode, or an algorithm description, be able to find the asymptotic time complexity
* NOT part of Exam 2: Quicksort, Mergesort

 Final exam topics - Updated 12/3/18

* Other notes: same cheat sheet as for exam 2 (provided by the instructor), calculator ok, possible multiple choice questions or questions with short answers (brief justification)
* The exam is NOT cumulative, but some topics from previous exams may be part of algorithms covered by this exam (e.g. some sorting method needed for sorting edges for Kruskal,...).
* Sorting: mergesort, quicksort (partition, randomized partition, median-of-three partition), count sort, bucket sort, radix sort.
* Heaps, Heapsort
  + Heap properties: shape, parent-child key relation, parent-child node-index mapping.
  + Build-Max-Heap - time complexity (not the proof, just the complexity)
  + Fix the heap when the key/priority of an item:
    - increased,
    - decreased
  + Insert in a heap
  + Delete/remove from a heap: both delete (meaning delete [max/min]) or delete x (meaning delete the specific element x)
  + Find top k largest elements using a heap show work or continue work in the format shown in slides.
  + Applicability: when/why would you use a heap instead of other options such as sorting the data or using it as is (unsorted and not in a heap).
  + Max-heap, Min-heap
  + Index heaps (Handles between a heap and another data structure)
  + Heapsort
    - algorithm,
    - time and space complexity
    - Properties: stable, adaptive (be able to give counter examples or proofs to support your answer)
* Trees: traversal, simple operations on trees, be able to read code and tell what it does and/or compute the time complexity. ~~be able to write code for easy methods (e.g. count number of nodes in a tree)~~
* Search trees
  + BST: insertion, left/right rotations, range of values in a subtree, valid search path in a BST, successor /predecessor   
    (NOT part of the exam: deletion, insertion at root),
  + 2-3-4 trees: insertion, search, lgN time complexity for insertion and search.
* Hash tables:
  + Hash functions (including for strings),
  + Separate chaining,
  + Open addressing (linear and quadratic probing, double-hashing),
  + Load factor,
  + Insertion, search, deletion
  + Time complexity: best, worst, average
* Graphs:
  + Directed/undirected,
  + Connected/strongly connected,
  + Traversals: BFS (with node colouring: white/gray/black), DFS (with node coloring, and start and finish time stamps),
  + Edge classification: tree edges (that discover white nodes) and backward edges (that indicate a cycle)
  + Detecting cycles,
  + Topological sorting,
  + Properties,
  + Time complexity (aggregate analysis)
* Minimum spanning trees:
  + Prim
  + Kruskal
  + Union-Find Algorithm (also called Disjoint Sets) is NOT part of the exam.- Updated 12/6/18 @5:02am
* For all the algorithms you should know the performance (best/worst/ expected, guarantees).
* Given piece of code or pseudocode be able to determine time complexity (including worst, best, average)
* Shortest paths (single source shortest path and all-pairs shortest paths) - ?? if covered in class ()