1. Basic Concept:
   (1) ADT
      a. Concept: encapsulate data & operations on the data, and hide them from users.
      b. What ADTs did we learn in this class? Set, Bag, Stack, Queue, Priority Queue, Tree (General tree, binary tree, BST, AVL, heap), and Graph.
   (2) Big O
      a. Basic rules: drop all but the fastest-growing term; drop any constant coefficient on that remaining term
      b. Be able to analyze big O run time
2. Searching
   (1) Sequential search
      a. How does it work
      b. big O analysis
   (2) Binary search
      a. How does it work
      b. What is the pre-condition? A sorted sequence is required
      c. Big O analysis
   (3) <u>Hash</u>
      a. Concept: key to address mapping process
      b. Concept of Hash table; hash function; perfect hash function; collision; load factor
      c. Hash function
         i. Perfect hash function– direct hashing & subtraction hashing
         ii. Other - <u>modulo arithmetic</u> (a prime number is often required)
      d. Collision resolution strategy - probing
         i. Linear probing
            1. General form: h(k,i) = (h(k)+c*i) % m. When given certain h(k) and c, know how to insert an element into the correct position.
            2. Problem – primary clustering
         ii. Quadratic probing
            1. General form h(k,i) = (h(k)+c1* i + c2 *i^2)%m. understand how it works.
            2. Problem – secondary clustering.

            iii. Double hashing – general form h(k,i) = (h1(k) + h2(k)*i)%m.
        e. Hash operations: insertion, searching, min/max, deletion; understand how to conduct these operations; Big O run time analysis

3. List: implementation – array & linked list
    (1) Linear list – each element has only one predecessor and one successor
        a. Stack – LIFO.
            i. Given an example, know whether it's a queue or a stack.
            ii. When using an array to implement a stack, how to conduct push and pop? Big O?
        b. Queue – FIFO
            i. Given an example, know whether it's a queue or a stack
            ii. When using an array to implement a queue, how to conduct enqueue and dequeue? Two different implementations: always use index 0 to place the first element; use a pointer to indicate the first element.
            iii. Big O for enqueue and dequeue when use different array implementations.
        c. Implementation: <u>linked-list</u>
            i. Differences between linked list and array
            ii. Basic types of linked list: singly linked list, doubly linked list, circularly linked list, doubly linked circular list
            iii. Basic operations: create a list, insertion, deletion, search, traversal, destroy a list
            iv. Basic coding and big O analysis for different operations on sorted/unsorted linked list; on linked list with head pointer only or head pointer & tail pointer.
    (2) Non-linear list – each element can have multiple successors
        a. Tree – each element can have multiple successors but only one predecessor
            i. Concepts:
                1. Degree, in-degree, out-degree
                2. Root node, leaf node, internal node
                3. Parent, children, <u>sibling, ancestor, Descendent</u>
                4. Level, height
                5. Subtree – note that a single node is also a subtree.
                6. Balance factor (only exists for binary trees): it can be positive, negative or zero.
                7. Balanced tree; complete tree

ii. Logical architecture: understand the concept for general tree, binary tree, BST, AVL and Heap as well as their relationship

- General tree
    - Binary tree
        - BST – nodes are sorted (left < root < right); each subtree is also a BST.
            - AVL – a balanced BST
            - Not balanced BST
        - Heap – root is the largest node (for max-heap); each subtree is also a heap.
        - Others…
    - Trees in which one node can have more than two children

iii. General Tree Operations: (how to conduct operations in preorder & postorder)
Note that, these operations are all done in a recursive way.

1. Traverse a tree: e.g. Print out all the nodes. What orders? Both preorder & postorder
2. Count tree node: what order? Doesn't matter
3. Count tree height
4. Destroy a tree: what order? Only postorder.
5. Note that, the general tree is not sorted, so we have to traverse the whole tree to search a specific node or the min/max node. It's very similar to traversal operation. Therefore, we did not explicitly mention search and Max/min.

iv. Binary Search Tree Operations: (Basic coding & big O analysis & illustration of the insertion & deletion process)

1. Traverse a tree: similar to general tree case. Preorder & postorder & inorder. can only be done recursively.
2. Count tree height: similar to general tree case.
3. Search a tree: can be done both recursively and iteratively
4. Max/Min: can be done both recursively and iteratively
5. Insert a node
    a. BST – search + add; All inserts take place at a leaf or at a leaflike node
    b. AVL – search + add + rebalance. (understand how to conduct rebalance and be able to illustrate.)
6. Delete a node – the deleted node has no children; one child; or two children.
    a. BST – search + remove
    b. AVL – search + remove + rebalance

v. <u>Heap operations</u> (bigO analysis & illustration of insertion and deletion process)
    1. Insert a node – add + reheap; the new node can only be inserted at a fixed position.
    2. Delete a node – delete + reheap; we only care about delete the root node.
    3. Know how to store a heap in an array without any pointers.

b. Graph – each element can have multiple successors and multiple predecessors
    i. Concepts: directed graph; undirected graph; vertex; edges; cycle; acyclic; dag; planar; clique;
    ii. Relationship between vertex and edges – given n vertices, maximum number of edges (directed/undirected graph)?
    iii. <u>Graph representation</u>:
        1. Operations: Given a graph, know how to represent it with adjacency matrix & adjacency list. Given an adjacency matrix or adjacency list, know how to draw the graph.
        2. Understand when to use adjacency matrix and when to use adjacency list.
            a. When you care more about a given vertex's neighbors – adjacency list
            b. When you care more about whether an edge exists or not – adjacency matrix
    iv. <u>Graph traversal</u> – <u>Worklist algorithm</u>
        1. Be able to do depth-first traversal – stack based worklist
        2. Be able to do width-first traversal – queue based worklist

4. <u>Sorting</u> – understand the sorting category; be able to illustrate the sorting process; big O and stability analysis.
    (1) Selection based sorting
        a. Selection sort (basic coding)
        b. Heap sort
    (2) Insertion based sorting
        a. Insertion sort (basic coding)
        b. Shell sort
    (3) Exchange based sorting
        a. Bubble sort (basic coding)
        b. Quick sort

	Best case	Average case	Worst case	Stability
Selection sort	O(n^2)	O(n^2)	O(n^2)	Depends
Heap sort	O(nlogn)	O(nlogn)	O(nlogn)	Not stable
Insertion sort	O(n)	O(n^2)	O(n^2)	Stable
Shell sort	O(nlogn)	Quadratic	Quadratic	Not stable
Bubble sort	O(n)	O(n^2)	O(n^2)	Stable
Quick sort	O(nlogn)	O(nlogn)	O(n^2)	Not stable

SET	**Unsorted Array**	**Sorted Array**	**Hash Table**	**Unsorted Linked List**	**Sorted Linked List**	**BST**	**AVL tree**
Search	O(n)	O(log n)	O(n) (expected O(1))	O(n)	O(n)	O(h) (Log(n) <= h <= n)	log(n)
Add	O(n) + O(1) = O(n)	O(log n)+O(n) = O(n)	O(n) (expected O(1))	O(n) + O(1) = O(n)	O(n) + O(1) = O(n)	O(h)+ O(1) = O(h)	log(n)
Remove	O(n) + O(1) = O(n)	O(log n)+O(n) = O(n)	O(n) (expected O(1))	O(n) + O(1) = O(n)	O(n) + O(1) = O(n)	O(h)+ O(1) = O(h)	log(n)
Min/Max	O(n)	O(1)	O(**m**)	O(n)	O(1) (assuming fast access to tail)	O(h) (Log(n) <= h <= n)	log(n)