## Quiz

0. Concept: hashtable BST DFS/BFS.

1. Two Sum: N^2, NlgN, N.

2. Three Sum: N^3, N^2lgN, N^2.

3. How to delete the maximum from an array.

4. Find the largest/smallest M among N.

**5. Analyze the comlexity of merge sort, quick sort, binary search, priority queue, BST.**

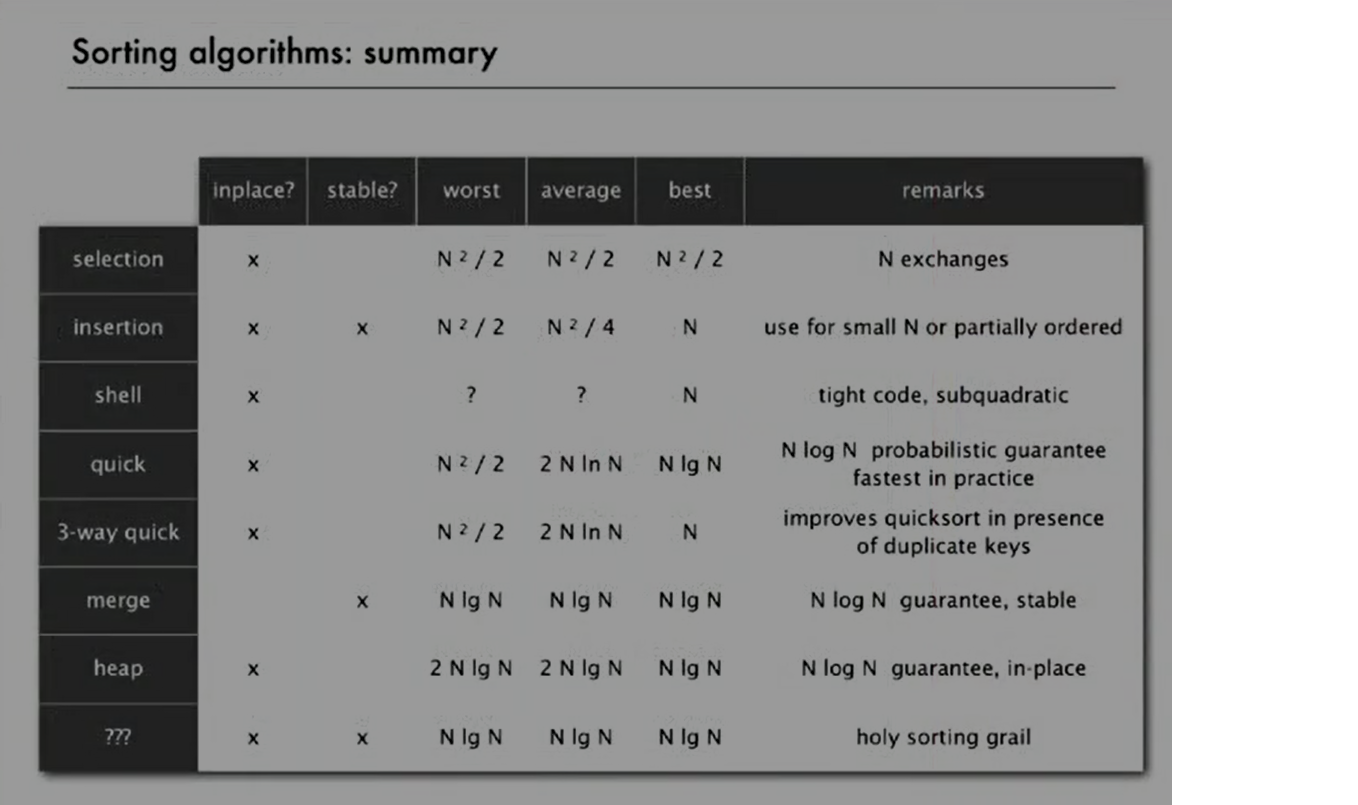
6. Three-way partition.

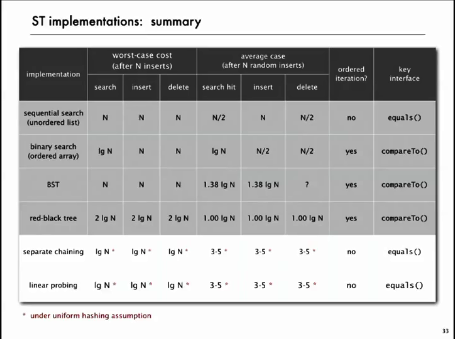
7. Shuffle an array.

8. Implement ST\_BST: insert, delete, range.

**9. String: find max repeated substring; max match of two strings; substring search.**

## Gragh





## Data Structure

* Stack
* Queue
* linked list
* array
* dictionary
* binary tree
* binary heap
* priority queue(what is the complexity?)
* Binary search tree
* Hash table
* undirected gragh
* directed gragh

## Design

* Implement calculation
* implement convex hull and shuffle
* bottom-up merge sort
* Give an array of N items, find the Kth largest: quick select
* Duplicate sort. 3-way partition.
* Implement priority sort using binary heap. Key function swim() and sink
* Heap sort: do not build max heap after each exchange.
* Implement dictionary using BST, especially the insert.
* Implement more function of BST
* Horizontal and vertical lines intersection number.
* is a gragh bipartite?
* implement dfs and bfs of a gragh?
* does a gragh contain a cycle?
* find a cycle that uses every edge exactly once.
* find a cycle that uses every vertex exactly once(NP complexity problem)
* topological sort
* strong component
* two MST algorithm: based on priority queue and union find
* four Shortest path algorithms.
* Parallel job scheduling problem(hard)
* string sort
* find longest repeated substring
* substring search: brute-force and 状态机 Boyer-Moore hash

## Conclusion

* There are some basic issues in algorithms, many different data structures are created to solve it.
* There are many data structures to store data: stack queue array linkedlist dictionary heap BST hashtable gragh
* Use case: add items to a cluster and remove it in an order, including stack, queue and priority queue.
* You could use ordered array, unorder linked list to save these data.
* Sorting use case: There is a key for an item. Basic operation is sort. You should sort all the items according to their keys.
* For sort problems, insertion sort, selection sort, merge sort, quick sort, heap sort. Data structures used include array and heap
* Searching use case: There are many key-value pairs. Basic operation is insert and search. You could store them in unordered array, ordered array, BST(ordered keys) or hash table(unordered keys). BST and hashtable are the best two implementation of dictionary. BST average complexity for insert and search is O(lgN), hashtable is O(1)
* Bottom line of search operation is O(N). But you can reduce it to O(lgN) by exclude some samples after each comparison. Index based search is O(1)
* Model your problem using data structure.
* There are many practical problems which can be projected to gragh problems.
* Shortest-paths is a broadly useful problem-solving model
* Use generic<T> or interface(Icompare) to make your design class more general.
* search and sort: it’s much easier to search for an item in a sorted array.

# 2016/05/04

## Union Find

* Fast find: idx[i] stores the component number.
* Fast union: idx[i] stores i’s parent.
  + 下好定义最重要！！
  + abstract common operations to function
* improvement:
  + balanced tree
  + path compression

## Practical experience

* exception: try-catch block, exception as a class, e.Message()
* Read from the console
* How to solve a problem?
  + Model the problem using mathematics.
  + Find a solution
  + Fast in speed? Save memory?
  + Improve iteratively.
* How to design a class?
  + Design the APIs first
  + Design API test code
  + Consider all the input case and corner case. Do input validation.
  + Implement the code。

# 2016/05/05

## Analysis of Algorithms

* 1 lgN N NlgN N^2 N^3 2^N
  + lgN: divide in half(binary search)
  + N: loop N^2: double loop N^3: triple loop
  + NlgN: divide and conquer(merge sort)
* 3-Sum problem: optimize based on sort and binary search. N^3🡪N^2lgN

## Stack and Queues

* **Data abstraction: separate interface and implementation.**
* Design a stack:
  + What APIs? Push() Pop() IsEmpty() Size()
  + Write client test code
  + think about possible input case and corner case.
  + preliminary implementation
  + make it better.
* You can implement stack/queue using linked list or resizing array. There are tradeoff between different designs.
* Stack application: Can always use a stack to remove recursion. implement calculation with () and +-\*/

## Practical experience

* How much memory needed for string variables?

字符串中的字符有不同编码方式，比如UTF-8，UTF-16，每个字符可能占1-3字节，结束字符’\0’

* StreamReader class: to read from a stream.
* Template class and template method <T>
* Implement IEnumerable in collection class: you should implement a method named GetEnumertor whose return type is IEnumerator. Then clients can use foreach to iterate a collection.
* Impelement Icomparable<T>: you should implement CompareTo() method
* How compiler implement a function: use stack
  + When a function is called, push the local environment along with return address into the stack.
  + When return from the function, pop return address and local environment from the stack.

# 2016/05/06

## Elementary Sort

* Selection sort and insertion sort.
* Application in Shuffle and Convex Hull

# 2016/05/07Z

## Merge Sort

* Rule the game first: sort items according to keys
* The parameter should be IComparable[] for generic. How you compare the element should be generic.
* Divide and conquer
* Analysis an algorithm: speed and memory.
* Improvement: insertion sort for tiny arr. If already sorted, no merge
* Bottom-up merge sort: recursion to iteration
* Stability:
  + insertion ok, but also depends on your code.
  + Selection sort is not ok for exchange.
  + Merge sort is ok, but depends on your code.

## Quick Sort

* Shuffle before sort to guarantee performance.
* Compare with merge sort, no extra space is needed. Faster than merge sort because of less data movement. But in the worst case, the complexity is N^2/2.
* Improvement: use insertion sort for tiny array. 3-way partition.
* Application: selection, duplicate sort

## Priority Queue

* Design API and basic implementation
* Find the largest M items in a stream of N items
* Basic implementation: ordered array VS unorder array. Take O(N) to insert or delet. The goal is to find O(lgN) for both insert and delete.
* Binary heap implement
* Event-driven simulation: bouncing balls;
* Write client test first to get the idea of what going to do.
* Keep a priority queue to predict the future possible collision. The nearest collision is the one to happen. After two particles collision, add the future possible collision involving these two particles into the queue.

## Sorting Application

* Sorting various type of data
* multiple keys

## Dictionary(Symbol Table in Java)

* Core idea of searching

Symbol table is a data structure of key-value pairs which support two operations: insert key value pair and ***search*** a value according to the given key.

The basic implementation include two types: unordered linked list and ordered array. Searching in an unordered list is sequential search, which is O(N) complexity. Searching in an ordered list is binary search, which is O(lgN) complexity. Index-based Search is O(1) complexity.

Ordered array method still requires O(N) for insert and delete a key-value pair. The senior method is to use hash table(unordered) or BST(ordered). The above two methods are O(lgN) complexity for all kinds of operations on symbol table.

* How to design dictionary?
  + First, non-primitive data structure is a class. You should design a class.
  + Design APIs first. Design client test code then.
  + Analysis the basic properties and features.
  + Primininary implementation.
* Basic implementation
  + Maintain an (unordered) linked list of key-value pairs.
    - For search and insert, should scan all key-value pairs.
    - For search and insert, O(N) complexity
  + Maintain an ordered array of key-value pairs
    - Then search could be binary search. O(lgN)
    - Then for insert, need to shift all greater keys over. O(N)
    - For ordered dictionary, a lot more API could be developed.
  + Maintain a binary search tree
    - Search and insert average complexity is O(lgN)
    - Tree shape depends on order of insertion. The best occation is to keep balanced binary tree.
  + Maintain a red-black BST
  + Maintain a hash table

# 2016/05/08

## Binary Search Tree

* Ordered operation
  + Max and Min
  + Floor and celling
  + Subtree count
  + Select
  + Search and insert
  + Rank: how many keys < k? Easy recursive method(3cased)
  + Inorder traversal
* The key is how to keep balanced search tree to get the best performance. Methos as listed bellow.

## 2-3 Search Trees

* Perfect balance, guarantee log performance for search and insert
* Search
* Insertion
  + Replace 2 node with 3node
  + Or move middle key into parent. If root is 4 node, split it into three 2nodes

## Left-leaning red-black tree

* Convert 2-3 tree to BST, so comes the rb tree. Make BST more balanced.
* The color of the node is determined by the link from its parent.
* Same code handles all cases. Because by analyzing the different cases, you can reduce them to simple and base cases.

## B-trees

* Generic 2-3 threes. More keys could be stored in one node.

## Hash Table

* Balanced search tree is effective data structure for key-value pair insert, search, delete and so on. If no ordered operation is needed, hash table is another effective way.
* Save items in a key-indexed table
* Classic space-time tradeoff
  + If no space limitation: use key as index
  + If no time limitation: hash everything to the same index and do sequential search.
* Hash function: method for computing array index from key.
  + Modular hashing
  + Uniform hashing assumption: bins and balls
* Collisions:
  + Two different keys hashing to same index
  + Metrics: can search item through index, which is O(1).
  + Key and value type should be object because you can not declare generic array.
  + How to deal with collisions effectively: algorithm and data structure
  + Method 1: Separate chaining symbol table
    - Improvement: hash to two positions, insert key in shorter of the two chains
  + Method 2: linear probing
    - When a new key collides, find next empty slot and put it there.
    - Array size M must be greater than number of key-value pair.
    - When the array is almost full, the performance would decrease because you should move a lot to find a empty slot.
    - Improvement: skip a variable amount instead of 1.
    - Improvement: hash to two positions.
  + Hash tables VS balanced search trees
    - Hash tables: no effective alternatives for unordered keys. Simpler to code except good hash function.
    - Balanced search trees: support for ordered opreations.stronger performance guarantee.

## Application for BST

* 1-d range count: size(lokey, hikey) = rank(hi) – rank(lo). Rank of a key is the number of nodes with smaller keys.
* 1-d range search
* Line segment intersection
* 2-d range search: 2 keys for an item.
* Interval search tree

## Practical experience

* How to use Array.Sort(T[] arr, Comparison<T>), Array.Sort(T[] arr, IComparable<T> )
* For design, always design the API first.
* Override Equals() method. Parameter should be Object. There should be type cast inside the method.

# 2016/05/09

## Appclication for BST 2

* hashset: a collection of distinct keys, no value. Implementation: remove value of dictionary implementation.
* dictionary client
* File index
* matrix-vector multiplication
  + use hashtable(because order is not important, so ht is better than BST) to represent sparse vector, index-non\_zero\_value is key-value pair.

## Undirected Gragh

* a set of vertices connected pairwise by edges.
* problems: path shortest-path cycle euler-tour connectivity planarity, etc
* Gragh representation: use integers to represent vertex(because you can convert between names and integers uising Sysmbol table)
* Gragh API:
  + Gragh(int V)
  + AddEdge(int v, int w)
  + Adj(int v)
  + Degree(int v)
  + E()
  + V()
* Basic representation of Gragh
  + matain a list of edges(linked list or array)
  + matain a 2-D V-by-V boolean array
  + matain vertex-indexed array of lists

## DFS

* Maze exploration: record where you have gone. When to the end of one road, step back and try anther unvisited road
* put unvisited vertices on a stack
* Design Pattern: Decouple gragh data type from gragh processing.
* API: class Paths{ Paths(Gragh g, int s)} //find all possible paths in G from point s
* usage: reachability path-finding topological-sort directed-cycle-detection.
* DFS is basis for solving difficult diagragh problems

## BFS

* put unvisited vertices on a queue.
* BFS can solve shortest path problem like fewest number of hops in a communication network.
* application: web crawler: DFS will go too far for this problem. Node close to the root has more connection to the root.

## Connected Component

* Connective component is a maximal set of connected vertices.

## Connected Gragh

* Set of vertices connected pairwise by directed edges.
* path, shortest path, topological sort, strong connectivity, transitive closure, page rank

# 2016/05/10

## Digragh

* Implementation is very similar to undirected gragh.

## *Topological sort And Cycle Detection*

* Directed acyclic gragh(DAG)
* goal: redraw DAG so all edges point upwards.
* DFS and BFS topological sorting
* application: precedence scheduling, cyclic inherent

## *Strong component*

## *Minimum spanning tree*

* Given: undirected connected gragh G with positive edge weights
* Def: A spanning tree of G is a subgragh T that is connected and acyclic.
* (subgragh: contain of vertices and partial edges.)
* Goal: find a min weight spanning tree
* Greedy algorithm:
  + assume edge weight is distinct and G is connected. So there exsits one and only one MST.
  + A cut in a gragh is a partition of its vertices into two nonempty sets.
  + A acrossing edge connects a vertex in one set with a vertex in the other.
  + Cut property: given any cut, the crossing edge of min weight is in the MST
  + Greedy MST algorithm: give a cut, find the minimum edge, try another until acrossing edges of any cut contains a finding edge.
    - Create an Edge data abstraction
    - Kruskal’s algorithm: order edges in asceding order of weight, add edges to T unless doing so would create a cycle. use MinPQ and Union Find
    - Prim’s algorithm: lazy and eager

# 2016/05/12

## Shortest path in edge weighted digragh

## ( Broadly Useful Problem-Solving Model )

* From a source point s, distTo[w]<=disTo[v]+e.weight(); The shortedst path to w comes from the shortest path to w’s neighbor
* Generic shortest-paths algorithm: Initialize DistTo[] and relax edge until optimal conditions satisfy.
* Dijkstra’s algorithm(non-negtive)
* Topological order(non-cycle, could be negative weight): all DAG(Directed acyclic Gragh) has topological order.
* A SPT exists iff there is no negative cycles. The weights of all edges in a cycle add to a negative number.
* Bellman-Ford algorithm: can have negtive weight and cycle, but no negative cycle. Relax all edges V times.
* Improve of BF algorithm: maintain queue of vertices whose distTo[] changed.
* Application:
  + image resize.
    - direction: from a pixel to 3 downward neighbors;
    - weight: energy function of 8 neighbor pixels;
    - delete shortest path from up to down. For every row, one pixel will be deleted.
  + longest path: negative all the weight and find shortest path
  + Arbitrage Detection.

## Maximum Flow

* MinCut Problem: the capacity of a cut is the sum of capacities(weight) of all edges go from A to B, application: to separate two objects using the least cost
* MaxFlow Problem: application-how to deliver a max flow to the destination.
* Ford\_Fulkerson algorithm:

How to combine many datas of the same type

* + Like linked list.
  + Or array(include binary heap)
  + Or tree (point to several other datas)

# 2016/05/13

## Maximum Flow

## (Broadly Useful Problem-Solving Model )

* Max-flow is equal to min-cut. You can compute a min-cut from a max-flow
* Implementation
  + Flow edge data type: class FlowEdge
  + Augmenting path in original network is equivalent to directed path in residual network

## String

* String: sequence of characters(immutable)
* Key-Indexed Count:
  + Sort algorithms should support all the objects that implement ICompare<>
  + Assumption： keys are integers between 0 and R-1, can use key as index
  + Count frequency and compute accumulate. Then you can know from which place one item should start.
* MSD
  + implement string sort
* application
  + find longest repeated substring
  + find key word in a string

## 2016/05/16

* Substring search: find a pattern of length M in a text of length N
  + M is small while N is huge
  + Brute-force
  + Deterministic finite state automation(DFA) 状态机
    - how to build DFA from pattern: simulate pat[1, ..,j-1] on DFA
  + Boyer-Moore
    - carefully design how much a skip when find a mismatch
  + Rabin-Karp Fingerprint search