1. Data Structures

Array:

* graph illustration
* consecutive memory
* fixed length
* constant time random access
* slow insertion/deletion
* why does raw array require the fixed length at initialization, instead of a variable length? Why std::vector doesn’t have such limitation?
* caching advantage

List:

* graph illustration
* spread-out in the memory
* variable length
* linear access time
* fast insert/deletion
* single linked list vs double linked list: implementation and performance comparison

Graph:

* graph illustration
* vertex and edge
* directional and un-directional
* generally for more complicated data modeling

Tree & Binary tree:

* graph illustration
* a special form of graph
* parent and children
* spread-out in the memory

Binary search tree:

* graph illustration
* spread-out in memory
* variable size
* fast search and insertion in general
* deletion: no child; one child; two children – replace the node with its in-order predecessor or in-order successor
* balanced tree vs unbalanced tree: performance implication
* Red black tree: a self-rebalancing BST; used in the implementation of STL std::map

|  |  |  |  |
| --- | --- | --- | --- |
|  | Array | List | BST |
| Random Access | O(1) | O(n) | N/A |
| Search | O(n) or O(log n) if sorted | O(n) | O(log n) |
| Insert | O(n) | O(1) | O(log n) |
| Delete | O(n) | O(1) | O(log n) |

1. Abstract Data Types

Queue:

* graph illustration
* first in first out (FIFO)
* implementation: array; circular array; double linked list

Stack:

* graph illustration
* last in first out (LIFO)
* implementation: array; singly linked list

HashMap:

* constant access time
* constant insertion time
* The elements are unordered
* implementation: hash function + array with linked list of key-value pairs
* Conceptually, shall we allow user to modify the value of a key?

TreeMap:

* The elements are ordered
* O(logn) for read and modification
* Implementation: balanced BST + compareTo()

1. STL containers

std::vector

* contiguous storage
* array-like behavior
* dynamic array under the hood
* on the heap

std::map

* key-value pairs
* red-black tree implementation
* on the heap

std::iterator

* various iterators for various containers
* independent of the container used; a more uniform/general interface for the container framework

A lot of other containers

* list, set, unordered\_map, queue, stack and etc.

1. Questions
   1. Merge sorted array (#88) [Array]

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

* 1. Merge two sorted lists (#21) [List]

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

* 1. Valid Parenthesis (#20) [Stack]

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

* 1. Two sum (#1) [map]

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution.

* 1. BST search and insert
  2. Binary tree traversal – depth first
     1. In order (#94): left, root, right
     2. Pre order (#144): root, left, right
     3. Post order (#145): left, right, root
  3. Binary tree traversal - breadth first (#102)
  4. Integer Break (#343)

Given a positive integer n, break it into the sum of at least two positive integers and maximize the product of those integers. Return the maximum product you can get.

For example, given n = 2, return 1 (2 = 1 + 1); given n = 10, return 36 (10 = 3 + 3 + 4).