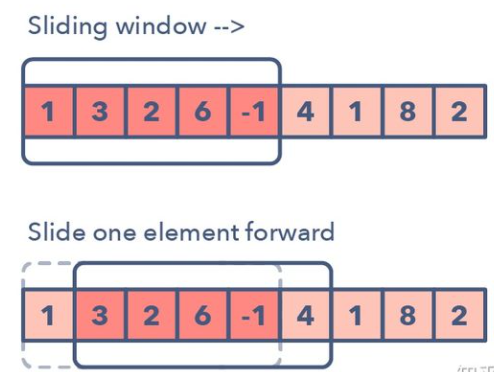
**算法的14种模式**

1. **滑动窗口**

滑动窗口类型的题目精彩是用来执行数组或是链表上某个区间上的操作。比如找最长的全为1的子数组长度。滑动窗口一般从第一个元素开始，一直往右边一个一个移动。



判断可能需要使用滑动窗口策略的方法：

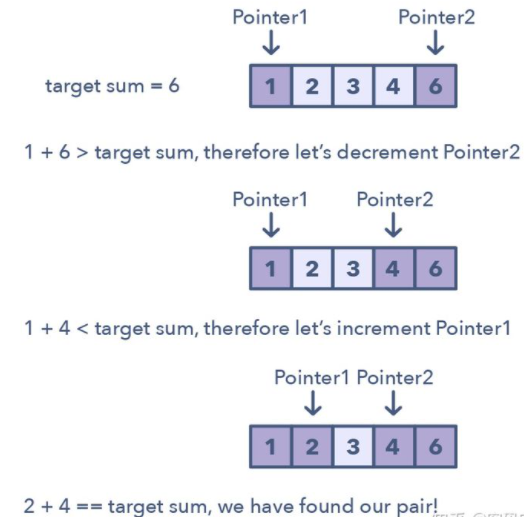
* 问题的输入是一些线性结构：链表，数组，字符串；
* 求最长/最短子字符串或是某些特定的长度要求；

题目：

* 窗口大小为K的最大子数组和；（简单）
* 拥有K个不同的字母的最长字串；（中等）
* 字符串的同字母异序词；（困难）

1. **双指针**

两个指针朝着左右方向移动（同向双指针和异向双指针），直到一个或两个都满足某种条件。通常用在排好序的数组或是链表中寻找对子。比如，去比较数组中每个元素和其他元素的关系时，需要用到双指针。双指针可以降低空间和时间复杂度。



判断使用双指针的方法：

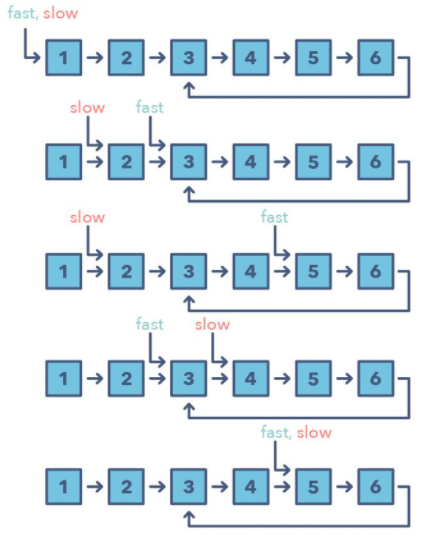
* 数组或链表是排好序的，需要找一些组合满足某种限制条件；
* 这种组合可能是一对数，三个数，或是一个子数组；

题目：

* 输出一个排好序的数组的平方数组（简单）；
* 3-Sum（中等）；
* 比较两个字符是否相等，字符中包括得有退格键（中等）；

1. **快慢指针**

两个指针在数组上的移动速度不一样。通过控制指针不同的移动速度（比如环形链表），这种算法证明了肯定会相遇的。快的指针会追上慢的一个。



上图显示了快慢指针最终在5相遇了。

判断使用快慢指针的方法：

* 问题需要处理环是的问题，比如环形链表和环形数组；
* 当需要知道链表的长度或是某个特别位置的信息时；
* 有些情形下，不应该使用双指针，比如我们在单链表上不能往回移动的时候。一个典型的需要用到快慢指针的是当你需要去判断一个链表是否是回文的时候。

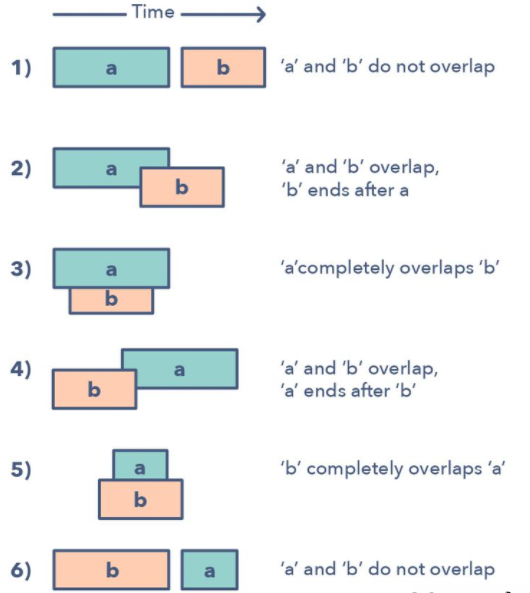
题目：

* 链表是否有环（简单）；
* 链表是否满足回文（中等）；
* 环状数组中检测环（困难）；

1. **区间合并**

用来处理有区间重叠的高效技术。需要判断是否有重叠，或合并区间。对于重叠，是这么工作的：

给两个区间，一个是a，一个是b，可以有6种情况。



判断是否使用区间合并：

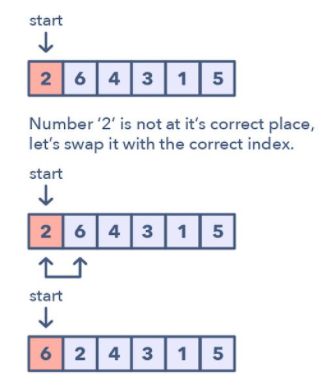
* 当需要产生一堆相互之间没有交集的区间的时候；
* 听到重叠区间的时候；

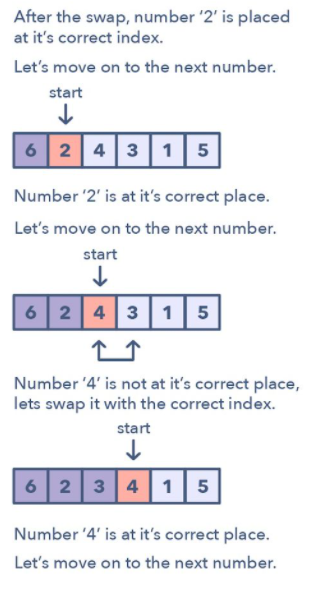
题目：

* 区间交集（中等）
* 最大化CPU负载（困难）

1. **循环排序**

用来处理数组中的数值限定在一定区间的问题。这种模式一个个遍历数组中的元素，如果当前这个数不在其应该在的位置的话，就把它和它应该在的那个位置上的数交换一下。





判别方法：

* 涉及到排序好的数组，而且数值一般满足于一定的区间；
* 在排好序/翻转过的数组中，寻找丢失的/重复的/最小的元素；

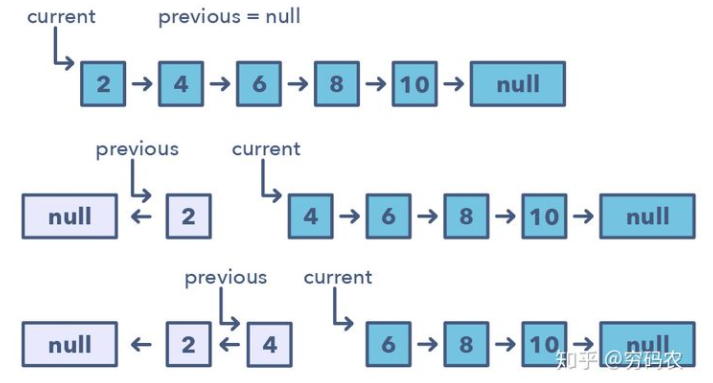
题目：

* 需要数组中没出现的数字（简单）；
* 寻找最小的没出现的正整数（中等）；

1. **原地反转链表**

题目需要去翻转链表中某一段的节点，通常都是原地翻转，就是重复使用已经建好的节点，不使用额外的空间。

这种模式每次翻转一个节点。一般需要用到多个变量，一个变量指向头节点（current），另一个（previous）指向刚刚处理完的那个节点。需要先将当前节点（current）指向前一个节点（previous），再移动到下一个。同时，需要将previous指向刚刚更新完的节点。



判别方法：

* 翻转链表，不使用额外的空间；

题目：

* 翻转链表中的一段（中等）；
* 翻转每K个元素为一组的子链表段（中等）；

1. **树上的BFS**

这种模式基于宽度搜索，适用于需要遍历一棵树。借助于队列数据结构，从而能保证树的节点按照他们的层数打印出来。打印完当前层所有元素，才执行到下一层。

这种BFS模式是通过把根节点加到队列中，然后不断遍历直到队列为空。每一次循环中，会把头节点拿出来，然后对齐进行必要的操作。在删除每个节点的同时，其孩子节点都会被加到队列中。

判别方法：

* 需要按层操作的方式去遍历树；

题目：

* 二叉树层序遍历（简单）；
* 之字形遍历（中等）；

1. **树上的DFS**

DFS基于深度搜索来实现树的遍历。可以使用递归（显式栈）来记录遍历古城中访问过的父节点。

该模式的运行方法是从根节点开始，如果该节点不是叶子节点，需要做：

1. 需要区别我们是先处理根节点（前序），处理孩子节点再处理根节点（中序），还是处理完所有孩子再处理根节点（后序）。
2. 递归处理当前节点的左右孩子。

判别方法：

* 需要按前中后序的DFS方式遍历树；
* 该问题的解一般离叶子节点比较近；

题目：

* 树上所有路径上表示的数字的和（中等）；
* 树中所有能形成目标和的路径（中等）；

1. **双堆**

可以拿到一大把可以分成两队的数字。为了解决这个问题，感兴趣的是，怎么把数字分成两半？使得：小的数字放在一起，大的放在另一半。

该模式用到了两个堆。一个最小堆用来找最小元素，一个最大堆，拿到最大元素。这种模式将一半的元素放在最大堆中，这个就可以从这一堆中找到最大元素。同理，最小堆一样，O（1）时间找到他们中最小的元素。通过这样的方式，这一大堆元素的中位数就可以从两个堆的堆顶拿到数字，计算出来。

判别方法：

* 这种模式在优先队列，计划安排问题中有奇效；
* 找一组数中的最大/最小/中位数；
* 二叉树；

题目：

* 找数字流中的中位数（中等）；

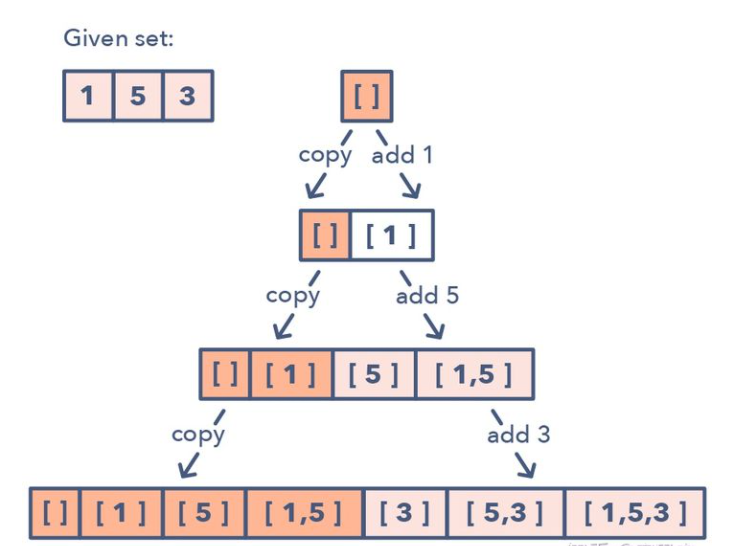
1. **子集**

涉及到排列组合问题，子集模式讲的是用BFS来处理这些问题；

给一组数字[1, 5, 3]

1. 我们从空集开始：[[]]；
2. 把第一个数1），加到之前已经存在的集合中：[[], [1]]；
3. 把第二个数5），加到之前的集合中得到：[[], [1], [5], [1,5]]；
4. 再加第三个数3），则有：[[], [1], [5], [1,5], [3], [1,3], [5,3], [1,5,3]]；

该模式的详细步骤如下：



判别方法：

* 需要寻找数字的组合或排列；

题目：

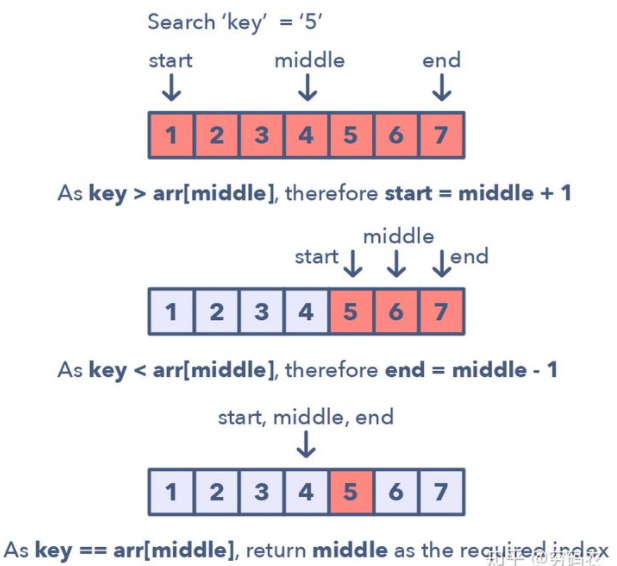
* 有重复元素的所有子集（简单）；
* 通过改变大小写，找到所有可能的字符串排列（中等）；

1. **变种二分法**

当需要解决的问题的输入是排好序的数组，链表或者排好序的矩阵，要求寻找某些特定元素。

对于一组满足上升排列的数据来说，这种模式的步骤是这样的：

1. 首先，算出左右端点的中点。最简单的方式是：middle=（start+end）/ 2。这种方式可能会出现整数越界。一般都是推荐使用：middle=start+（end-start）/ 2；
2. 如果要找的目标等于中点的值，返回中点的下标；
3. 如果不等的话，有两种移动方式：
4. 如果target<arr[middle]，将下一步搜索空间放到左边end = middle - 1；
5. 如果target>arr[middle]，将下一步搜索空间放到右边start = middle+1；



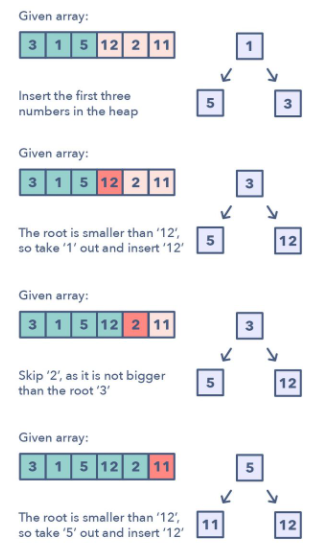
题目：

* 顺序未知的二分（简单）；
* 无界排序数组的二分（中等）；

1. **最大的前K个元素**

求解最大/最小/最频繁的K个元素的题。用来记录前K类型的最佳数据结构是堆，这种模式借助堆来解决很多种前K个数值的问题。

1. 将前K个元素插入到最小堆或最大堆。
2. 遍历剩下的还没访问的元素，如果当前的元素比堆顶元素大，就把堆顶元素先删掉，再加当前元素进去。



这种模式下，不需要排序数组，因为堆具有良好的局部有序性。

判别方法：

* 求最大/最小/最频繁的前K个元素；
* 通过排序找一个特定的数；

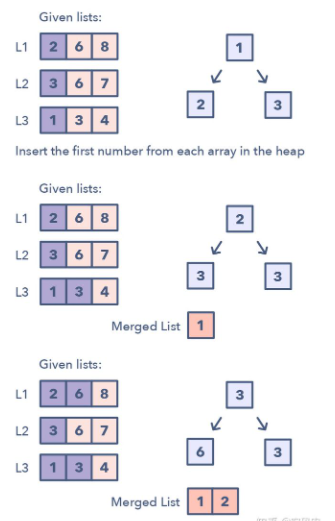
题目：

* 前K大的数（简单）；
* 前K个最常出现的数字（中等）；

1. **K-路归并**

K路归并可以解决设计到多组排好序的数组的问题。

将每个数组中最小的一个元素加入到最小堆中，得到全局最小值。当我们拿到这个全局最小值之后，再从该元素所在的数组里取出其后面紧挨着的元素，加入堆。如此重复直到处理完所有元素。



执行过程：

1. 把每个数组的第一个元素都加入到最小堆中；
2. 取出堆顶元素，将该元素放入排好序的结果集合里面；
3. 将刚取出的元素所在的数组里面的下一个元素加入堆；
4. 重复2，3，直到处理完所有数字；

判别方法：

* 该问题的输入是排好序的数组，链表或矩阵；
* 合并多个排好序的集合，或需要找这些结合中最小的元素；

题目：

* 合并K个排好序的链表（中等）；
* K对数和最大（困难）；

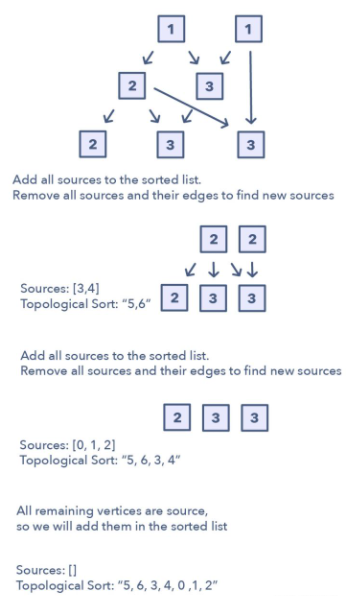
1. **拓扑排序**

拓扑排序用来寻找一种线性的顺序，这些元素之间具有依赖性。

执行过程：

1. 初始化
2. 借助于HashMap将图保存成邻接表形式；
3. 找到所有的起点，用HashMap来帮助记录每个节点的度；
4. 创建图，找到每个节点的入度
5. 利用输入，把图建好，然后遍历一下图，将入读信息记录再HashMap中；
6. 找所有的起点
7. 所有入度为0的节点，都是有效的起点，将他们都加入到一个队列中；
8. 排序
9. 对每个起点，执行一下操作：
10. 把它加到结果的顺序中；
11. 将其在图中的孩子节点取到；
12. 将其孩子的入度减少1；
13. 如果孩子的入度为0，则改孩子节点为起点，将其加入队列中；

b）重复a）过程，直到起点队列为空；



判别方法：

待解决的问题需要处理无环图；

需要以一种有序的秩序更新输入元素；

需要处理的输入遵循某种特定的顺序；

题目：

任务执行顺序安排（中等）；

树的最小高等（困难）；