**1. 排序算法**

* 1. **冒泡排序**

1.1.1 算法描述：

1）比较相邻元素，如果第一个比第二个大，交换

2）从开始到结尾的每对相邻元素均做1）中的工作，则最后一个为最大的数

3）对所有元素重复1-2步，除了最后一个，直到不再需要交换为止

1.1.2 稳定性：稳定

1.1.3 空间复杂度

1.1.4 时间复杂度：最优，平均，最坏

* 1. **插入排序**

1.2.1 算法描述：

1）把待排序的数组分成已排序和未排序两部分，初始的时候把第一个元素认为是已排好序的。

2）从第二个元素开始，在已排好序的子数组中寻找到该元素合适的位置并插入该位置。

3）重复上述过程直到最后一个元素被插入有序子数组中。

1.2.2 稳定性：稳定

1.2.3 空间复杂度

1.2.4 时间复杂度：最优，平均，最坏

* 1. **快速排序**

1.3.1 算法描述：

1）从数列中挑出一个元素，称为"基准"（pivot）

2）重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面。（相同的数可以到任何一边）。

3）递归地把小于基准值元素的子数列和大于基准值元素的子数列排序。

1.3.2 稳定性：不稳定

1.3.3. 空间复杂度：In-place ，非in-place 最坏

1.3.4 时间复杂度：最优，平均，最坏

* 1. **归并排序**

1.4.1 算法描述：

1)递归法：

i) 申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列

ii)设定两个指针，最初位置分别为两个已经排序序列的起始位置

iii)比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置

iv)重复步骤iii)直到某一指针到达序列尾

v) 将另一序列剩下的所有元素直接复制到合并序列尾

2)迭代法（原理如下（假设序列共有n个元素）：

i)将序列每相邻两个数字进行归并操作，形成ceil(n/2)个序列，排序后每个序列包含一/两个元素

ii)若此时序列数不是1个则将上述序列再次归并，形成ceil(n/4)个序列，每个序列包含四/三个元素

iii)重复步骤ii)，直到所有元素排序完毕，即序列数为1

1.4.2 稳定性：稳定

1.4.3 空间复杂度

1.4.4 时间复杂度：最优，平均， 最坏

* 1. **堆排序**

1.5.1 基本概念：

1）二叉堆：通过数组实现的二叉树：

父节点i的左子节点位置left(i) = 2i+1

父节点i的右子节点位置right(i) = 2i+2

子节点i的父节点位置parent(i) = floor((i-1)/2)

2）最大（小）堆：

最大（小）元素值出现在根节点的二叉堆

1.5.2 算法描述：

1）把最大堆堆顶的最大数取出(如何建立：筛选法、插入法)

2）将剩余的堆继续调整为最大堆，再次将堆顶的最大数取出

(如何实现：把最后一个元素a移到根结点，比较a和子元素b大小，如果b大于a则由b代替根结点，重复此过程直到没有子元素比根结点大为止)

3）重复步骤1)-2)，直到剩余数只有一个时结束

1.5.3 稳定性：不稳定

1.5.4 空间复杂度

1.5.5 时间复杂度：最优，平均，最坏

References:

https://zhuanlan.zhihu.com/p/42586566

https://zh.wikipedia.org/wiki/%E5%86%92%E6%B3%A1%E6%8E%92%E5%BA%8F

https://zh.wikipedia.org/wiki/%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F

https://zh.wikipedia.org/wiki/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F

https://zh.wikipedia.org/wiki/%E5%BD%92%E5%B9%B6%E6%8E%92%E5%BA%8F

https://zh.wikipedia.org/wiki/%E5%A0%86%E6%8E%92%E5%BA%8F

**2. 查找算法**

**2.1 顺序查找：**

2.1.2 适用范围：顺序存储或者链接存储的线性表

2.1.3 算法描述：按顺序检查每一个元素，知道找到所寻的元素为止

2.1.4 空间复杂度

2.1.5 时间复杂度：最优，平均，最差

**2.2 二分查找：**

2.2.1 适用范围：元素有序的数组

2.2.2 算法描述：

0) 给定目标值T和有序数组

1) L为0，R为n-1

2) 如果L>R，则查找以失败告终

3) 令m为floor((L+R)/2),

如果, 令L为m+1并回到步骤2);

如果, 令R为m+1并回到步骤2)

4) 当，查找结束；回传值m。

2.2.3空间复杂度

2.2.4时间复杂度：最优，平均，最坏

**2.3 插值查找**

2.3.1 适用范围：表长较大，关键字分布均匀

2.3.2 算法描述：（二分查找的改进）

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

2.3.3 空间复杂度

2.3.4 时间复杂度：最优，平均，最差

* 1. **斐波那契查找**

2.4.1适用范围：適用於已排序好的序列，且運行於不擅長處理乘、除法的處理器上

2.4.2 算法描述：（二分查找的改进）

要求开始表中记录的个数为某个斐波那契数小1，即n=F(k)-1；开始将k值与第F(k-1)位置的记录进行比较(及m=L+F(k-1)-1)，比较结果分为三种：

1）相等，则mid位置的元素即为所求；

2）大于，则L=mid+1，k-=2；

说明：L=m+1说明待查找的元素在[m+1,R]范围内，k-=2 说明范围[m+1,R]内的元素个数为n-F(k-1)=F(k)-1-F(k-1)=F(k)-F(k-1)-1=F(k-2)-1个，所以可以递归的应用斐波那契查找。

3）小于，则R=m-1，k-=1；

说明：L=m+1说明待查找的元素在[L,m-1]范围内，k-=1 说明范围[L,m-1]内的元素个数为F(k-1)-1个，所以可以递归的应用斐波那契查找。

2.4.3 空间复杂度

2.4.4 时间复杂度：最佳, 平均, 最差

**2.5 树表查找**

**2.5.1 二叉查找树：**

先对待查找的数据进行生成树，确保树的左分支的值小于右分支的值，然后和每个节点的父节点比较大小，查找最适合的范围。

空间复杂度：

时间复杂度：最优，平均，最差

**2.5.2 红黑树：**

空间复杂度：？？

时间复杂度：平均

**2.5.3 B树和B+树：**

空间复杂度：？？

时间复杂度：平均

**2.6 哈希查找**

2.6.1 哈希表：按照关键字为每一个元素"分类"，然后将这个元素存储在相应"类"所对应的地方，这样形成的表称为哈希表

2.6.2 算法描述：

0）设哈希表为HST[0~(M-1)]，哈希函数取H(key)，解决冲突的方法为R(x)；

1) 给定元素的key值，计算哈希地址 Di=H(key): 若HST为空，则查找失败；

若HST=k，则查找成功；否则，执行step2（处理冲突）。

2） 重复计算处理冲突的下一个存储地址 Dk=R(Dk-1)，直到HST[Dk]为空，或HST[Dk]=k为止。若HST[Dk]=k，则查找成功，否则查找失败。

2.6.3 空间复杂度

2.6.4 时间复杂度：最优，平均，最差

Reference:

https://www.cnblogs.com/maybe2030/p/4715035.html#\_label4

**3.常见数据结构**

**3.1 线性表：**

3.1.1 线性表是最常用且最简单的一种数据结构，它是n个数据元素的有限序列。

3.1.2 线性表的实现：

1）一种是使用数组存储线性表的元素，即用一组连续的存储单元依次存储线性表的数据元素；

2）另一种是使用链表存储线性表的元素，即用一组任意的存储单元存储线性表的数据元素。

**3.2 数组：**

数组是一种大小固定的数据结构，连续的内存空间和数据类型。数组实现的线性表优点在于可以通过下标来访问或者修改元素，比较高效，主要缺点在于插入和删除的花费开销较大，比如当在第一个位置前插入一个元素，那么首先要把所有的元素往后移动一个位置。

**3.3 链表：**

3.3.1 概念：

链表是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。链表由一系列节点组成，这些节点不必在内存中相连。每个节点由数据部分Data和链部分Next，Next指向下一个节点，这样当添加或者删除时，只需要改变相关节点的Next的指向，效率很高。

3.3.2 链表的实现：

1）循环单链表：主要是链表的最后一个节点指向第一个节点，整体构成一个链环。

2）双向链表：主要是节点中包含两个指针部分，一个指向前驱元，一个指向后继元，JDK中LinkedList集合类的实现就是双向链表。

3）循环双向链表：最后一个节点指向第一个节点的双向链表。

**3.4栈与队列：**

它们是比较特殊的线性表，因为对于栈来说，访问、插入和删除元素只能在栈顶进行，对于队列来说，元素只能从队列尾插入，从队列头访问和删除。

3.4.1栈：

栈是限制插入和删除只能在一个位置上进行的表，该位置是表的末端，叫作栈顶，对栈的基本操作有push(进栈)和pop(出栈)，前者相当于插入，后者相当于删除最后一个元素。栈有时又叫作LIFO(Last In First Out)表，即后进先出。应用场景：函数调用栈、表达式求值、括号匹配。

3.4.2队列**：**

队列是一种特殊的线性表，特殊之处在于它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作，和栈一样，队列是一种操作受限制的线性表。进行插入操作的端称为队尾，进行删除操作的端称为队头。应用场景：阻塞队列：生产者和消费者模型；并发队列：利用CAS实现高效的并发队列。

* 1. **树与二叉树**

3.5.1 树：由n（n>=1）个有限结点组成一个具有层次关系的集合。

1）每个结点有零个或多个子结点；

2）没有父结点的结点称为根结点；

3）每一个非根结点有且只有一个父结点 。

3.5.2 二叉树：它的结点至多有2棵子树，二叉树子树有左右之分，次序不能颠倒

1）二叉树第i层至多有2^(i-1)个结点，深度为k的二叉树至多有2^k-1个结点

2）满二叉树：深度为k且有2^k-1个结点的二叉树

3）完全二叉树：深度为k，有个结点的二叉树，当且仅当每一个结点都与深度为k的满二叉树中序号为1至n的结点对应时，称之为完全二叉树。

4）遍历方法：

i)先序遍历:若二叉树为空，则空操作，否则先访问根节点，再先序遍历左子树，最后先序遍历右子树。

ii)中序遍历:若二叉树为空，则空操作，否则先中序遍历左子树，再访问根节点，最后中序遍历右子树。

iii)后序遍历:若二叉树为空，则空操作，否则先后序遍历左子树节点，再后序遍历右子树，最后访问根节点。

3.5.3 平衡树：它的左子树和右子树都是平衡二叉树，且左子树和右子树的深度之差的绝对值不超过1。

3.5.4 B树和B+树：

1）B树：

B树是一种平衡的多路查找树（其每一个结点的孩子数可以多于两个，且每一个结点处可以存储多个元素），结点最大的孩子数目称为B树的阶

2）B+树：

在B树的基础上做了改进，在B+树中，出现在分支结点中的元素会被当作它们在该分支结点位置的中序后继者（叶子结点）中再次列出，且每一个叶子结点都会保存一个指向后一叶子结点的指针。

**3.6图：**

图是一种较线性表和树更为复杂的数据结构，在图形结构中，结点之间的关系可以是任意的，图中任意两个数据元素之间都可能相关。

**3.7堆：**

堆就是用数组实现的二叉树，所有它没有使用父指针或者子指针。堆根据“堆属性”来排序，“堆属性”决定了树中节点的位置。堆分为两种：最大堆和最小堆，两者的差别在于节点的排序方式。在最大堆中，父节点的值比每一个子节点的值都要大。在最小堆中，父节点的值比每一个子节点的值都要小。这就是所谓的“堆属性”，并且这个属性对堆中的每一个节点都成立。

**3.8 散列表：**

用一个与集合规模差不多大的数组来存储这个集合，将数据元素的关键字映射到数组的下标，这个映射称为“散列函数”，数组称为“散列表”。查找时，根据被查找的关键字找到存储数据元素的地址，从而获取数据元素。

References:

https://zhuanlan.zhihu.com/p/93928546