**一. 空间复杂度**

空间复杂度，是指除了原本的数据存储空间外，算法运行还需要额外的存储空间。不管是顺序栈还是链式栈，我们存储数据只需要一个大小为n的数组就够了。在入栈和出栈过程中，只需要一两个临时变量存储空间，所以空间复杂度是O(1)。

**二. 支持动态扩容的顺序栈**

尽管链式栈的大小不首先，但要存储next指针，内存消耗相对较多。基于数组实现一个可以支持动态扩容的栈，只需要底层依赖一个支持动态扩容的数组就可以了。

当栈满了以后，我们就申请一个更大的数组，将原来的数据搬移到新数组中。

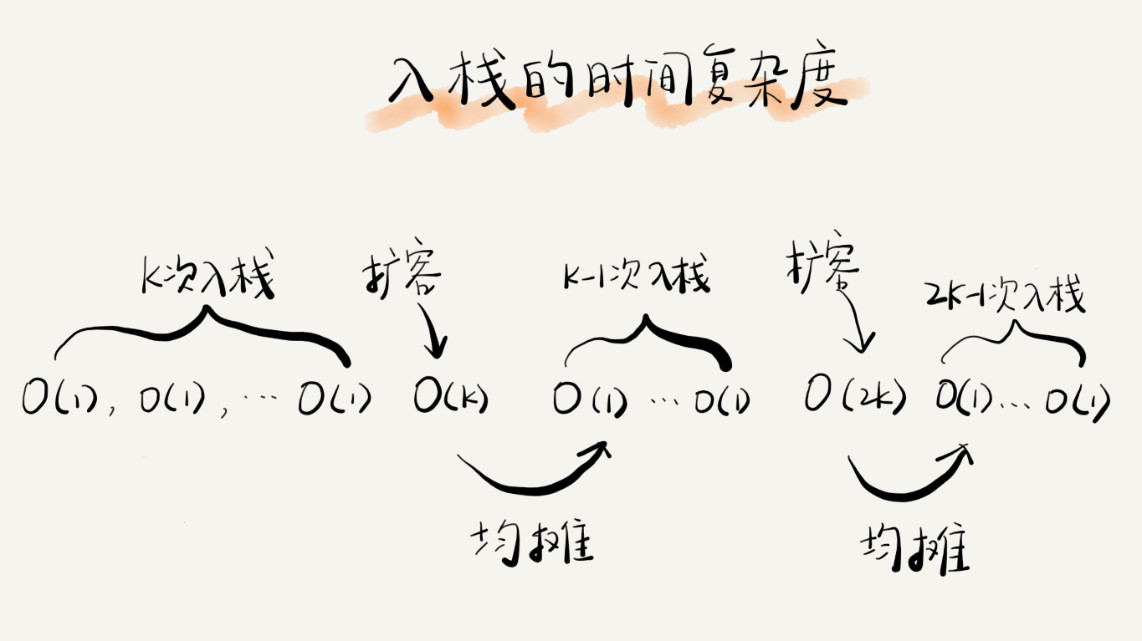
出栈O(1),当栈中有空闲空间时，入栈O(1),空间不够时就需要重新申请内存和数据转移，时间复杂度O(n)

摊还分析法：

为了分析的方便，我们需要事先做一些假设和定义：

* 栈空间不够时，我们重新申请一个是原来大小两倍的数组；
* 为了简化分析，假设只有入栈操作没有出栈操作；
* 定义不涉及内存搬移的入栈操作为 simple-push 操作，时间复杂度为 O(1)。

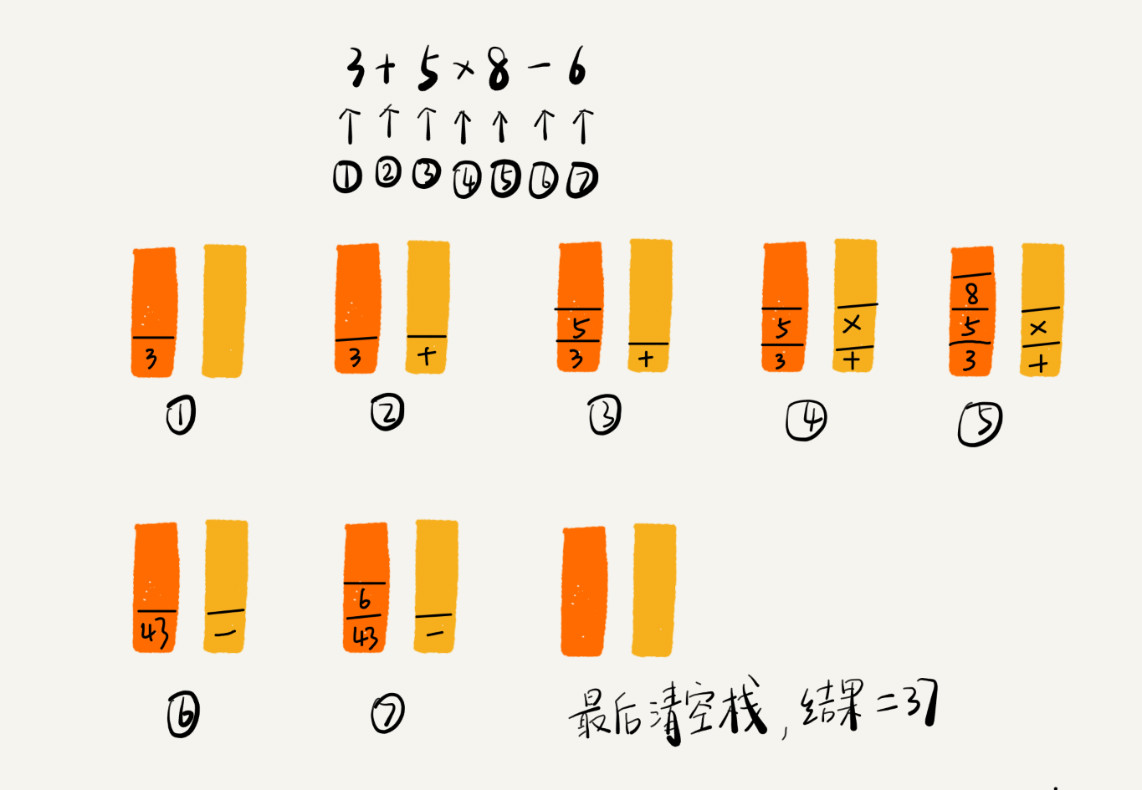
如果当前栈大小为 K，并且已满，当再有新的数据要入栈时，就需要重新申请 2 倍大小的内存，并且做 K 个数据的搬移操作，然后再入栈。但是，接下来的 K-1 次入栈操作，我们都不需要再重新申请内存和搬移数据，所以这 K-1 次入栈操作都只需要一个 simple-push 操作就可以完成。为了让你更加直观地理解这个过程，我画了一张图。



你应该可以看出来，这 K 次入栈操作，总共涉及了 K 个数据的搬移，以及 K 次 simple-push 操作。将 K 个数据搬移均摊到 K 次入栈操作，那每个入栈操作只需要一个数据搬移和一个 simple-push 操作。以此类推，入栈操作的均摊时间复杂度就为 O(1)。

通过这个例子的实战分析，也印证了前面讲到的，均摊时间复杂度一般都等于最好情况时间复杂度。因为在大部分情况下，入栈操作的时间复杂度 O 都是 O(1)，只有在个别时刻才会退化为 O(n)，所以把耗时多的入栈操作的时间均摊到其他入栈操作上，平均情况下的耗时就接近 O(1)。

**三. 栈在表达式求值中的应用**



实际上，编译器就是通过两个栈来实现的。其中一个保存操作数的栈，另一个是保存运算符的栈。我们从左向右遍历表达式，当遇到数字，我们就直接压入操作数栈；当遇到运算符，就与运算符栈的栈顶元素进行比较。

如果比运算符栈顶元素的优先级高，就将当前运算符压入栈；如果比运算符栈顶元素的优先级低或者相同，从运算符栈中取栈顶运算符，从操作数栈的栈顶取 2 个操作数，然后进行计算，再把计算完的结果压入操作数栈，继续比较。

**三. 阻塞队列和并发队列**

阻塞队列：队空，从头取数据会被阻塞；队满，从尾添数据会被阻塞

并发队列：线程安全的队列。多线程同时间仅一个存或取

**三. 队列在线程池等有限资源池中的应用**

线程池没有空闲线程，新的任务又请求线程资源时，线程池该如何处理：

1. 非阻塞，直接拒绝

2. 阻塞，将请求排队，运用队列。

基于链表实现的队列支持无限排队的无界队列(unbounded queue)，但是可能会导致过多的请求排队等待，请求处理的响应时间过长。所以，针对响应时间比较敏感的系统，基于链表实现的无限排队的线程池是不合适的。

而基于数组实现的有界队列（bounded queue），队列的大小有限，所以线程池中排队的请求超过队列大小时，接下来的请求就会被拒绝，这种方式对响应时间敏感的系统来说，就相对更加合理。不过，设置一个合理的队列大小，也是非常有讲究的。队列太大导致等待的请求太多，队列太小会导致无法充分利用系统资源、发挥最大性能。

实际上，对于大部分资源有限的场景，当没有空闲资源时，基本上都可以通过“队列”这种数据结构来实现请求排队。