做事情被打断时的优先级处理

有次去医院，到医院的复印室让工作人员帮我复印一些资料。复印快完成的时候，一个人在窗口问这位工作人员一些事，她马上放下复印的工作，跑到电脑前去帮这个人查资料；不到半分钟，电话响了，她又拿起电话…
挂掉电话之后，再帮这个人查完资料，最后再帮我复印剩下的一页资料。而最后复印一页资料的时间，不超过20秒。

这位工作人员处理问题的方式让我感觉有些新鲜，她默认后面的事情优先级比前面一件事情更高，当在做事情A（复印）时，来了事情B（查资料）；先保存A的状态，放到代办事项的顶部，然后去处理B；当事情C（接电话）来了的时候，保存B的状态，放到代办事项的顶部，去做C。等事情C做完了之后，从待办事项的顶部拿出事情B，B做完了，继续从待办事项中拿出A，直至所有的待办事项完成。
这不是堆栈的工作方式吗？

读书时，在数据结构这门课里，提到了递归与非递归。当权衡使用递归还是非递归的方式时，如果更加关注程序的执行效率，希望程序运行更快，占用更少的内存，那么无疑会选择非递归；如果更关心程序的开发效率，希望更快完成开发，我的CPU很快，内存很大，至于运行速度慢点，我也不介意，那么肯定会采用递归。
在递归调用时，每调用一个函数，就会把的状态（包括局部变量等）压到堆栈里，直到这个函数运行完了才退出堆栈。
如果递归次数太多，太多的嵌套函数调用可能会导致堆栈溢出。
回到刚才医院工作人员的这件事，事情A（复印）与事情B（查资料）两件事情的优先级一样，应该采取队列的方式，当事情A处理完再处理B；当碰到事情C（接电话）时，因为事情C的优先级更高，此时应该打断事情B，做事情C。
否则，每碰到一件事情B就放下原来的事情A，站在当事人的角度，一方面降低工作效率（因为在做事情B之前要保存原来事情A的状态，做完事情B之后要获取原来事情A的状态），另一方面还要耗费很大的精力（保存状态需要堆栈存储数据），事情很多时也容易出错，还会增加焦虑感。站在第三方的角度，事情（如复印、查资料）耽搁了很久没有完成。也就是说，这是一个双输的结果。

结论：
当两件事情优先级相同时，采取队列先进先出的工作方式；当后面事情优先级更高时，采取堆栈的方式，先进后出。