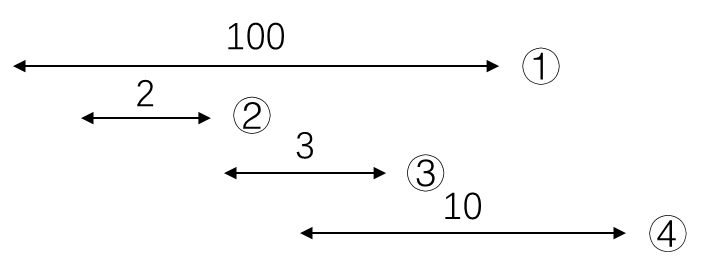
1. **动态规划的活动选择问题为什么要按照结束时间对活动排序？**

答：我们分别对下面的例子按照开始时间和结束时间排序，然后执行同样的算法，看看会发生什么：

比如有下面四个任务



编号为1、2、3、4，权重分别为100，2、3、10；

按照开始时间排序为：1、2、3、4，按照结束时间排序为2、3、1、4

按照开始时间计算P值为：P(1)=0, P(2)=0, P(3)=2, P(4)=2

按照结束时间计算P值为：P(1)=0, P(2)=1, P(3)=0, P(4)=1

注意上面的下标表示排序后的活动序号，而不是本身的活动编号。

之后我们计算M，对于开始时间排序：

M(1)=max{100,0}=100, M(2)=max{2+0, M(1)}=100, M(3)=max{3+M(2), M(2)}=102, M(4)={10+M(2), M(3)}=110

显然根据开始时间排序得到的M(4)是错误的。

对于结束时间排序：

M(1)=max{2,0}=2, M(2)=max{3+M(1), M(1)}=5, M(3)=max{100, M(2)}=100, M(4)=max{10+M(1), M(3)}=100

结果正确。

开始时间排序的错误结果本质上是因为，P值的定义要求前面的活动不能与当前活动冲突。具体地，对于活动j来说，P(j)指向的活动的最优解中，不能包含与j冲突的活动。而如果按照开始时间排序，可以看到1号活动排在了2号活动之前，而2号活动的最优解中也包含1号活动，所以当我们在从后往前推时，比如3号活动，与2号活动不冲突，所以它的解可以包含2号活动的解，但其实2号活动的解是跟3号活动冲突的，所以出现了错误。

**对于i<j, 如果按照结束时间排序，则如果活动j与活动i不冲突，则一定与活动i之前的活动也不冲突，也就是说与活动i的最优解不冲突，而按照开始时间排序，则如果活动j与活动i不冲突，则不一定与i之前的活动也不冲突。**

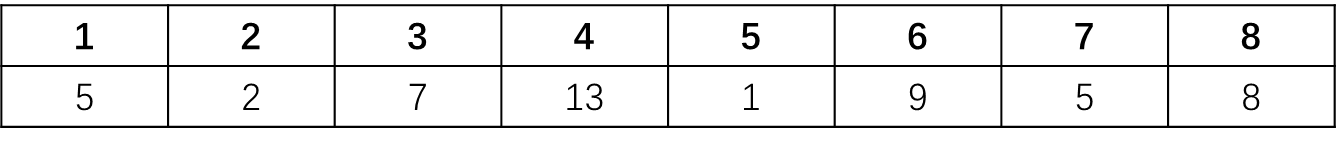
那么如果题目给出的就是按照开始时间排序的活动，我们必须要重新按照结束时间再排一遍吗？

显然不是，我们可以这样进行递归：

OPT(i)=max{opt(p[i])+vi, opt(i+1)}，其中p数组存储从前往后，第一个与活动i不冲突的活动j，初始化OPT(n+1)=0或者OPT(n)=vn，返回OPT(1)即可。如果需要活动序列，则从前往后再进行一次遍历，比较OPT(i)和OPT(i+1)的值，从而确认是否选择活动i。

1. **动态规划经典问题补充1：股票买卖问题**
   1. **基本问题：**

给定一组股票在一段时间内的价格，要求在这段时间至少进行一次买卖所得到的最大收益：



比如第一行为天数，第二行为价格。约束：允许同一天进行买入和卖出，这样收益为0。如果不允许，则也可以理解为没有进行交易。

之前在使用分治法时，我们得到了复杂度函数为T(n)=2T(n/2)+θ(n)=θ(nlogn)的分治算法，现在我们看看是否能使用动归的思想，将复杂度降低:

类似地，我们定义f(j)为第j天卖出的最大收益，则分两种情况，f(j)=max{0, f(j-1)+p[j]-p[j-1]}，其中p数组表示每天的价格，f(1)=0，因为第1天卖出相当于第1天先买入，收益为0。

f(2)=max{0,f(1)+p[2]-p[1]}=0,f(3)=5。理解上，如果第j天卖出，考虑两种情况，第1种，买入也在第j天，收益为0，第二种，买入在前j-1的某天，则最大收益为第j天的收益p[j]-p[j-1]加上如果在第j-1天卖出的收益。注意，此处是第j-1天卖出的收益，而不是前j-1天的最大收益。

（反例3、5、4、3.5、7，f值分别为：0、2、1、0.5、4，在计算f(3)也就是4那天对应的收益时，前三天最大收益为2，即第一天买，第二天卖，而f(3)=max{0, 2+(-1)}，其中-1就是第三天卖出的最大收益，而不是前三天最大收益）

求解过程类似于计算最大子序列和的过程。返回时，返回最大的f值即可。如果要输出买卖天数，则从卖出的j天开始往前遍历，到第一个f[i]=0时，就是买入的对应天数。

该算法时间复杂度为：O(n)，因为只需要一次遍历就能得到最终结果。

* 1. **空间复杂度优化：**

假设数据已读入内存，则该算法的空间复杂度为：O(n)，因为对每天的股票，都需要存储一个f值。如何将空间复杂度优化至常数级别？

方法1：直观上，存储f(j-1)和fmax即可。

方法2：存储计算f(j)时，存储前j天的最低价格pmin和当前最大f值fmax即可，计算f值时，f=p[j]-pmin。比如，对于价格序列：5、2、7、13、1、9来说，其pmin值分别为5、2、2、2、1、1，这样计算f值更容易。

* 1. **如果允许买卖两次呢？**

设f2(j)为第2次卖出在第j天，g1(j)表示前j天进行1次买卖的最大收益，同时将2.1中的f定义为f1(j)，表示在第j天进行第1次卖出所获的最大收益。

则显然，g1(j)=max{f1(j)}，即我们在2.1中区别的两个概念。

所以f2(j)可以定义为：f2(j)=max{p[j]-p[i]+g1(i-1)} for i=1:j

可以这样理解，我们如果在第j天进行第2次卖出，则收益为我们在前i-1天进行了一次交易的最大收益，然后在第i天买入，在第j天卖出。当然，我们需要循环遍历每个可能的i值。最后返回最大的f2即可。

该算法的时间复杂度为O(n2)，因为我们可以分别用O(n)的时间计算出f1(i)和g1(i)，之后对于每个f2(j)，内部循环为O(n)，共n个f2(j)，所以总共为O(n2)。空间复杂度为O(n)，因为需要存储n个f1和n个g1。

* 1. **对于2.3的问题，如何去掉循环？**

将f2(j)定义为：max{g1(j-1), f2(j-1)+p[j]-p[j-1]}，同样，f2(1)=0这个定义就与问题2.1中的定义形式相似了，只不过2.1中买卖1次可以理解为买卖2次，但第一次必须在同一天买入卖出，所以收益最大为0。而本题就是将0替换为了g­1，最终返回max(f2(j))即可。

目前，由于去掉了循环，所以时间复杂度降到了O(n)。

* 1. **如果允许买卖k次呢？**

**O(kn)**

1. **动态规划经典问题补充2：最长递增子序列**
   1. **基本问题：**

对于一个给定的数值序列a1, a2, …, an，求出长度最长的一个子序列ai，使得对所有j>i，都有aj>ai。

例如：一个序列为 3 5 10 7 8 9，则最长子序列为3 5 7 8 9，而不是3 5 10。

* 1. **动归的错误解法**：

类似活动选择问题，我们定义f(j)为前j个元素组成的最长子序列，p[j]为第j个元素之前，第1个比j小的元素的下标。

则f(j)=max{f(p[j])+1, f(j-1)}。初始化f(0)=0或f(1)=1，最后返回f(n)。

该算法可以理解为，如果该序列选择了第j个元素，则子序列最长长度为前p[j]个元素构成序列的长度+1，如果没有选择这个元素，则子序列最长长度为f(j-1)。

这个算法存在一个严重的漏洞，即假设我们选择了第j个元素，那它一定能与p[j]的最优解连起来构成新的解吗？

下面举个反例：10、20、30、15、25。按照上面的思路，元素15的最优解是3，即{10，20，30}，对后面的25来说，它的解就是3+1=4，显然是错误的。问题的本质在于，f(p[j])与j可能存在不相容的地方。而为什么活动选择就可以呢，那是因为活动选择问题的f(p[j])包含的是按照结束时间排序后，p[j]和p[j]之前的活动，**如果j与p[j]不冲突，则j与f(p[j])中的活动一定不冲突。而本例中，即使j与p[j]能够构成递增序列，但可能与f(p[j])中的解冲突，**因此不能用这种方法来解。

但这个思想显然是有用的，我们来分析一下：

该算法时间复杂度为O(n)。空间复杂度也为O(n)，且无法优化，原因是我需要存储j之前的每一个f(p[j])值，因为后面递归受到约束，不一定会用到哪一个f值。而股票买卖问题则不同，每次递归使用的f值都是f(j-1)，所以只需要存储f(j-1)即可。

这里还要引申一个问题，如何更快地计算p[j]。显然，暴力的解法就是从当前元素开始向前遍历，复杂度为O(n2)。现在，我们使用一个辅助队列来进行计算，队列的操作如下：

初始队列为空，从头遍历序列，按次序选择ai

1. 如果队列为空，则p[i]=0，并将i从队尾添加到队列中（注意插入的不是ai）；
2. 如果队列不为空，则从队尾开始往前遍历，如果队列中下标所指的元素大于ai，则将该元素出队，并继续往前比较，直到遇到第一个小于ai的元素的下标j，赋值p[i]=j，如果都比ai大，则p[i]=0（此时队列已清空）；

举个例子说明队列的变化：10、20、30、15、5

开始，队列为{10}，之后增长到{10, 20, 30}，遍历到15时，队列为{10, 15}，最后遍历到5时，队列仅剩{5}。

该思想的本质是：如果一个元素i比我更小，而且还比我更靠后出现在序列中，则我一定不会成为i之后的元素的p值，所以我就没有必要再进行比较。同时，该队列中的元素永远是递增的，因此该队列也叫单调队列。

使用均摊分析中的记账法(accounting method)来对该思想进行复杂度分析：

定义一个规则，初始时，每个元素有两块钱，队列外的元素与队列中的元素进行比较时，如果比队尾元素大，则要花掉1块钱（空队列也算做1次比较），否则队尾元素花掉1块钱，同时出队。

结论：如果一共有2n块钱，则足够完成所有操作。

分析：元素开始有2块钱，如果入队，则需要花1块钱，剩余1块，如果被踢出队列，则剩余为0。

性质：所有队列中的元素都只有1块钱。1次比较相当于1块钱，所以总的比较次数最多2n次（剩余常数块无所谓，只需说明2n是个上界即可）。于是，我们将O(n2)的复杂度借助辅助队列优化到了多项式级别。

* 1. **动归的正确解法：**

修改一下f(j)的定义，为以第j个元素结尾的最长递增子序列，这样元素j一定与f(p[j])不矛盾，但f(p[j])也不一定是最优解。所以f(j)=max{f(i)+1}, 1<=i<j，且a[i]<a[j]。

本质上，该算法的思想是，对所有元素j之前以比j小的元素结尾序列都遍历一边，找一个最长的贴上去。

空间复杂度O(n)，存储每一个f(i)，时间复杂度O(n2)，因为要遍历所有的i。该复杂度类似PPT中的分段最小二乘法，需要遍历该点之前所有的符合条件的解（分段最小二乘所有点都符合，但本题只有比当前元素小的点符合，同时最小二乘的复杂度更高，因为任意两个点都能组成一个pair，本题只需要对单一元素检查即可）。

PS. 存在令时间复杂度降低到O(n\*logn)的方法，类似二分查找，需要先排序，但课程没有涉及。

PS2. 该问题的变式问法，对于一个序列，最少删除多少元素可以变成一个递增序列？

即序列长度减去最长递增子序列的长度。

PS3. 第三次上机作业#2合唱团问题，将数组正反各做一遍，求得两个递增序列数组后，计算和大的下标值。