第一题

直接插入排序是一种简单的排序算法，其工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。以下是直接插入排序在不同情况下的表现：

### 最好情况

最好情况发生在输入数组已经是完全有序的情况下。此时，每次插入操作只需要比较一次就可以找到插入点，因为每次插入的元素都比已排序序列中的所有元素都要小，所以它将直接插入到序列的开头。

- \*\*比较次数\*\*：\( n-1 \)（对于n个元素，每个元素只需要比较一次）

- \*\*移动次数\*\*：最多 \( n-1 \)（因为每次插入都需要将所有元素向后移动一次）

- \*\*时间复杂度\*\*：\( O(n) \)（线性时间复杂度）

### 最坏情况

最坏情况发生在输入数组是完全逆序的情况下。这意味着每次插入都需要将当前元素与所有已排序的元素进行比较，并且需要将所有元素向后移动来为新元素腾出空间。

- \*\*比较次数\*\*：\( \frac{n(n-1)}{2} \)（对于n个元素，每个元素都需要与其他所有元素比较一次）

- \*\*移动次数\*\*：\( \frac{n(n-1)}{2} \)（每次插入都需要将所有已排序的元素向后移动）

- \*\*时间复杂度\*\*：\( O(n^2) \)（二次时间复杂度）

### 平均情况

平均情况下，直接插入排序的性能介于最好和最坏情况之间。平均情况下的比较次数和移动次数都与 \( n^2 \) 成正比。

- \*\*比较次数\*\*：平均为 \( \frac{n^2}{4} \)

- \*\*移动次数\*\*：平均也为 \( \frac{n^2}{4} \)

- \*\*时间复杂度\*\*：\( O(n^2) \)（二次时间复杂度）

直接插入排序在小规模数据集或基本有序的数据集中表现较好，但对于大规模数据集，由于其时间复杂度较高，通常不是最优选择。在实际应用中，更高效的排序算法（如快速排序、归并排序或堆排序）通常被优先考虑。

第2题

直接插入排序是一种基本的排序算法，让我们来分析您的问题：

1. \*\*每一趟都能至少将一个元素放在其最终的位置上\*\*：

- 在最好的情况下（即输入数组已经有序），直接插入排序在每一趟中确实至少将一个元素放在其最终位置上。这是因为每个元素在插入时，都会找到它在已排序序列中的确切位置。

- 在最坏的情况下（即输入数组完全逆序），直接插入排序在每一趟中也会将一个元素放在其最终位置上。不过，这个元素总是当前未排序部分的第一个元素，因为它是最小的，所以会被插入到已排序序列的开头。

2. \*\*是否是稳定的排序算法\*\*：

- 稳定的排序算法是指在排序过程中，相等的元素之间的相对顺序不会改变。

- 直接插入排序是稳定的。在插入排序的过程中，如果一个元素与已排序序列中的元素相等，它将被插入到这些相等元素的后面。这意味着相等元素的原始顺序将被保留。

总结来说，直接插入排序在每一趟中都能至少将一个元素放在其最终位置上，并且它是一种稳定的排序算法。

第3题

希尔排序是一种基于插入排序的算法，通过引入一个增量序列来改进插入排序的性能。增量序列通常选择为长度逐步缩小的序列，直到最后增量为1，完成普通的插入排序。最常用的增量序列是长度每次减半。

对于提供的数据表 (78, 100, 120, 25, 85, 40, 90, 15, 60, 35, 105, 50, 30, 10, 28, 12)，我们可以用逐步减半的增量来进行排序，具体步骤如下：

初始序列

[78, 100, 120, 25, 85, 40, 90, 15, 60, 35, 105, 50, 30, 10, 28, 12]

增量为8

处理相间隔8个元素的子数组：

* 对 (78, 15) 排序：找到15的位置，将其它元素后移。
* 对 (100, 60) 排序：找到60的位置，将其它元素后移。
* 对 (120, 35) 排序：找到35的位置，将其它元素后移。
* 对 (25, 105) 排序：位置不变。
* 对 (85, 50) 排序：找到50的位置，将其它元素后移。
* 对 (40, 30) 排序：找到30的位置，将其它元素后移。
* 对 (90, 10) 排序：找到10的位置，将其它元素后移。
* 对 (15, 28) 排序：找到28的位置，将其它元素后移。

举例中的一步： [15, 60, 35, 25, 50, 30, 10, 78, 100, 105, 120, 85, 40, 90, 28, 12]

增量为4

* 继续处理相隔4的子数组。

实际步骤： [10, 28, 30, 25, 12, 35, 15, 40, 50, 60, 78, 85, 100, 105, 120, 90]

增量为2

* 继续处理相隔2的子数组。

步骤演示： [10, 12, 15, 25, 28, 30, 35, 40, 50, 60, 78, 85, 90, 100, 105, 120]

增量为1

* 最后进行一次全数组的插入排序。

最终结果： [10, 12, 15, 25, 28, 30, 35, 40, 50, 60, 78, 85, 90, 100, 105, 120]

在实现时，应该注意每次操作的结果，确保每个步骤都正确执行，并调整相应的元素位置。这些排序操作中要维持希尔排序的性质，处理逻辑应当清晰规范。每趟排序后，打印当前的数组状态可以帮助追踪排序进度。

第4题

第1趟排序

* 比较105和50，交换 -> [50, 105, 30, 25, 85, 40, 100, 12, 10, 28]
* 比较105和30，交换 -> [50, 30, 105, 25, 85, 40, 100, 12, 10, 28]
* 比较105和25，交换 -> [50, 30, 25, 105, 85, 40, 100, 12, 10, 28]
* 比较105和85，交换 -> [50, 30, 25, 85, 105, 40, 100, 12, 10, 28]
* 比较105和40，交换 -> [50, 30, 25, 85, 40, 105, 100, 12, 10, 28]
* 比较105和100，无需交换
* 比较100和12，交换 -> [50, 30, 25, 85, 40, 105, 12, 100, 10, 28]
* 比较100和10，交换 -> [50, 30, 25, 85, 40, 105, 12, 10, 100, 28]
* 比较100和28，交换 -> [50, 30, 25, 85, 40, 105, 12, 10, 28, 100]

第2趟排序

* [30, 25, 50, 40, 85, 12, 10, 28, 105, 100]
* 进行相应位置的比对和交换，最大值位于从右端开始的正确位置

第3趟排序

* [25, 30, 40, 50, 12, 10, 28, 85, 100, 105]
* 继续比对和交换

依此类推，直到完成所有排序

* [10, 12, 25, 28, 30, 40, 50, 85, 100, 105]

每一趟排序后，你会发现列表的未排序部分的最大值会被推向它应在的位置，就像气泡一样从水底浮到水面。冒泡排序最坏的情况是O(n²)，当整个数组都是逆序时。

第5题

第1趟排序

选择第一个元素50作为基准点：

* 找到比50小的放左边，比50大的放右边。
* 移动情况：30, 25, 40, 12, 15, 35, 10, 28 都比 50 小，都移动到了左边。

排序后的数组：[28, 30, 40, 25, 15, 12, 35, 10, 50, 120, 85, 100, 90, 60, 105, 78]

第2趟排序 - 左半部分

选择28作为基准点：

* 找到比28小的放左边，比28大的放右边。
* 移动情况：所有元素都比28大，没有移动。

对于10这部分，接着以10作为基准点进行排序：

* 所有比10小的点都在左边，10是最小的，无实际移动。

接着继续递归处理剩余子数组，以30为基准点：

* 移动情况：25, 15, 12, 不需移动因为都已经在正确的位置。

最后左边结果：[10, 12, 15, 25, 28, 30, 35, 40]

第2趟排序 - 右半部分

选择120作为基准点：

* 所有比120大的点都在右边，120是最大的，无实际移动。

接着继续递归处理剩余子数组，以85为基准点：

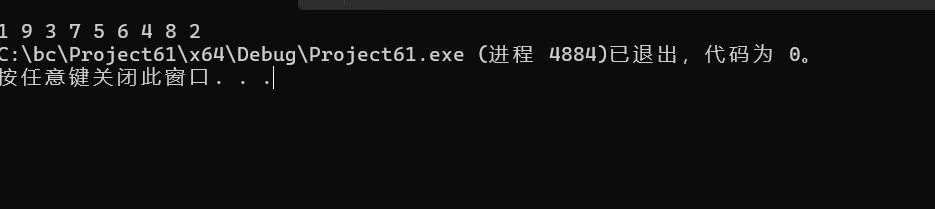
* 移动情况：78, 60, 35都比 85 小。

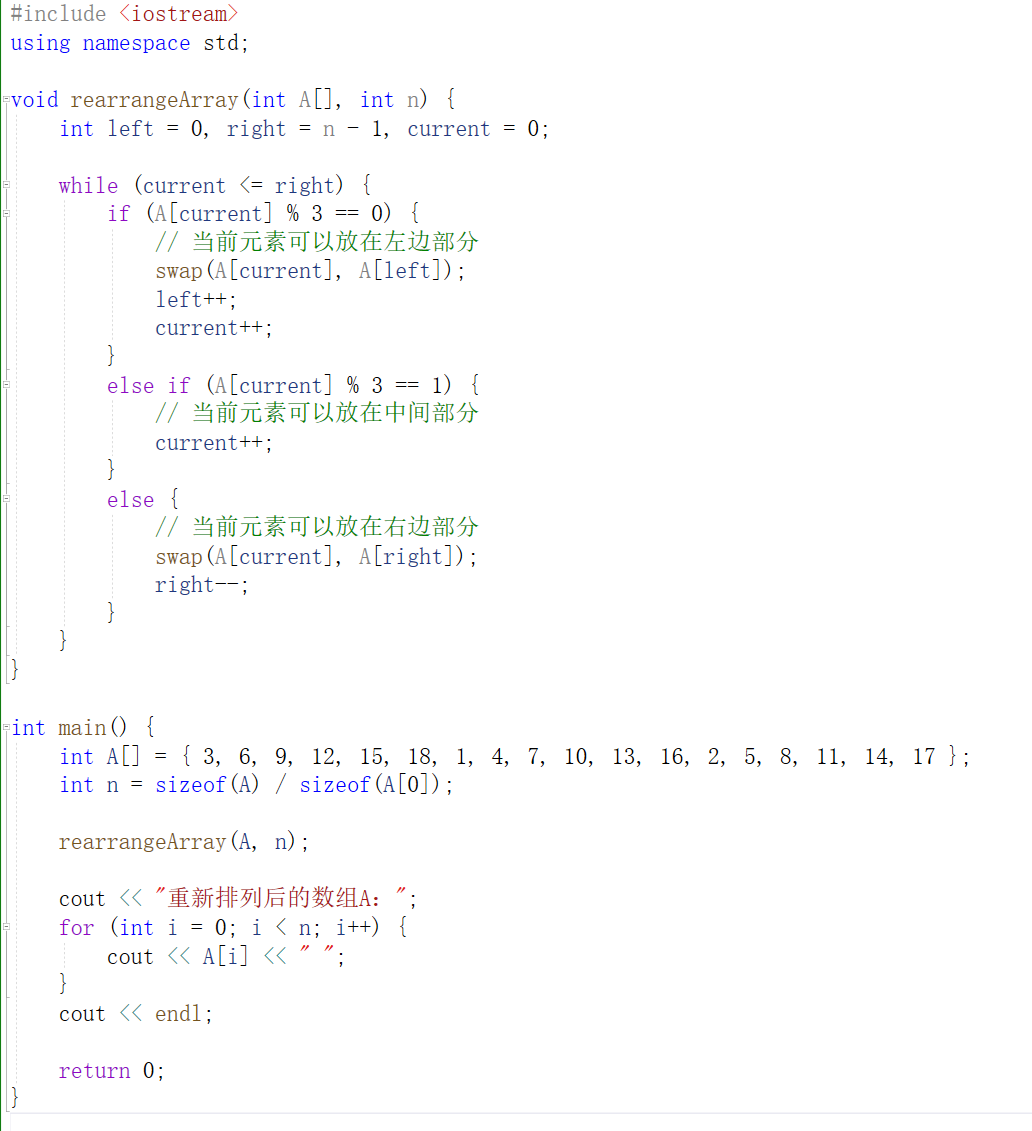
最后右边结果：[50, 60, 78, 85, 90, 100, 105, 120]

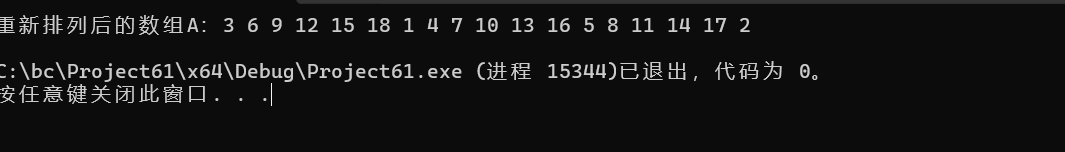
最终结果

合并左右子数组：[10, 12, 15, 25, 28, 30, 35, 40, 50, 60, 78, 85, 90, 100, 105, 120]

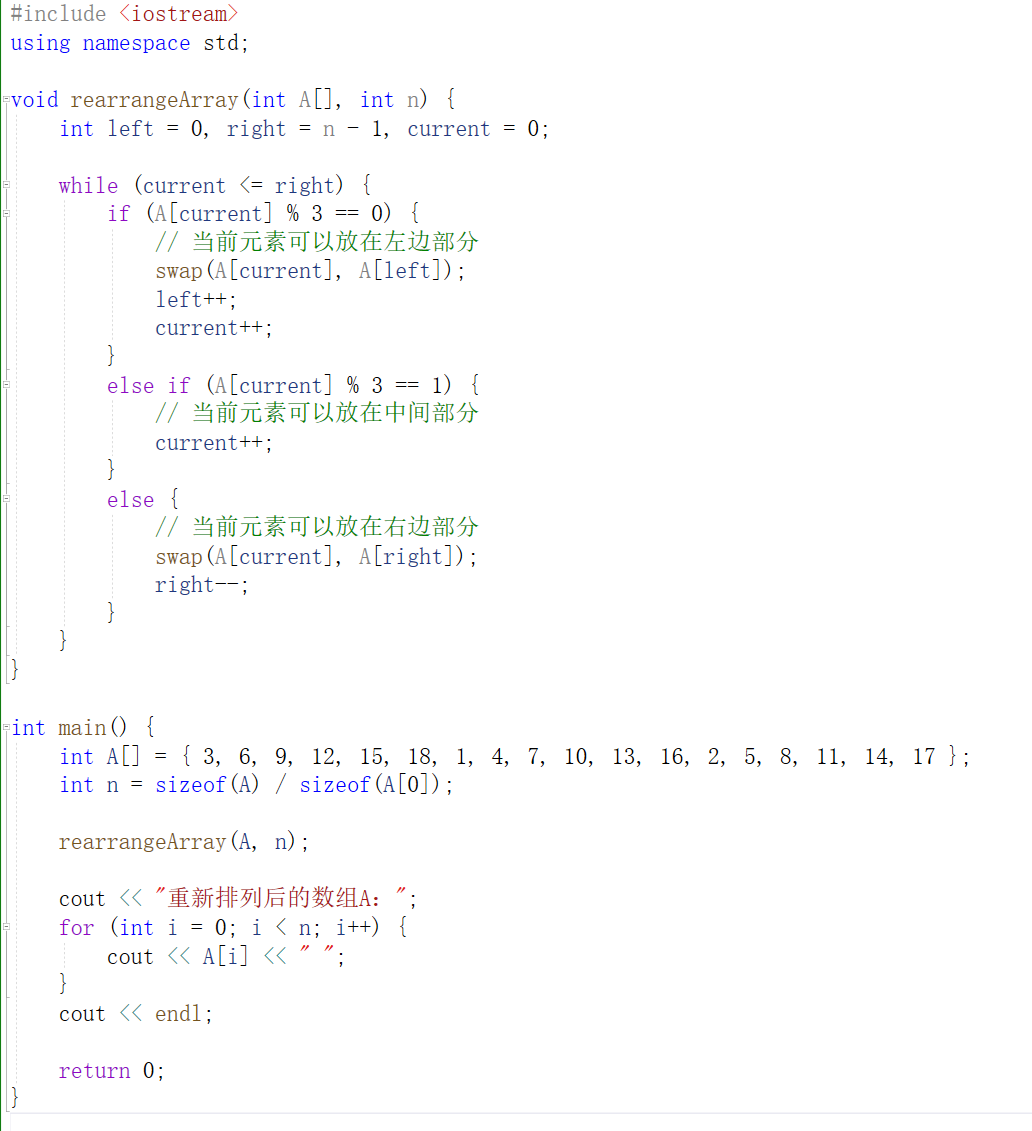
第6题

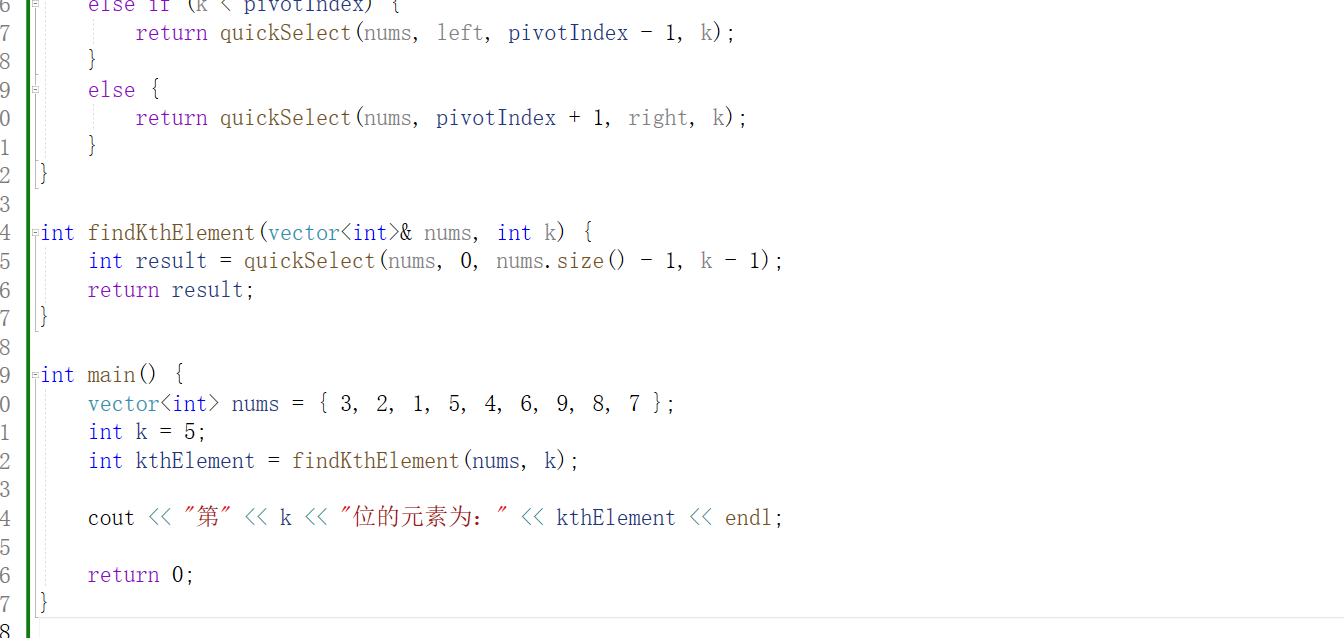


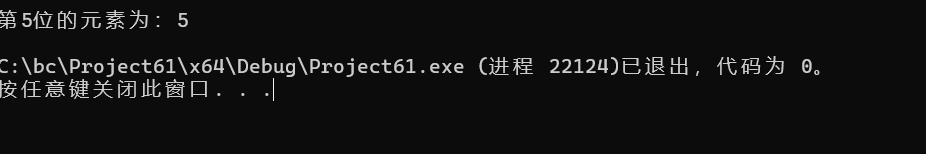
第7题



第8题







第9题

(a) (100,60,80,90,40,20,30,70,35)：这个序列不是堆，因为根节点的值100大于其中的其他所有节点的值。

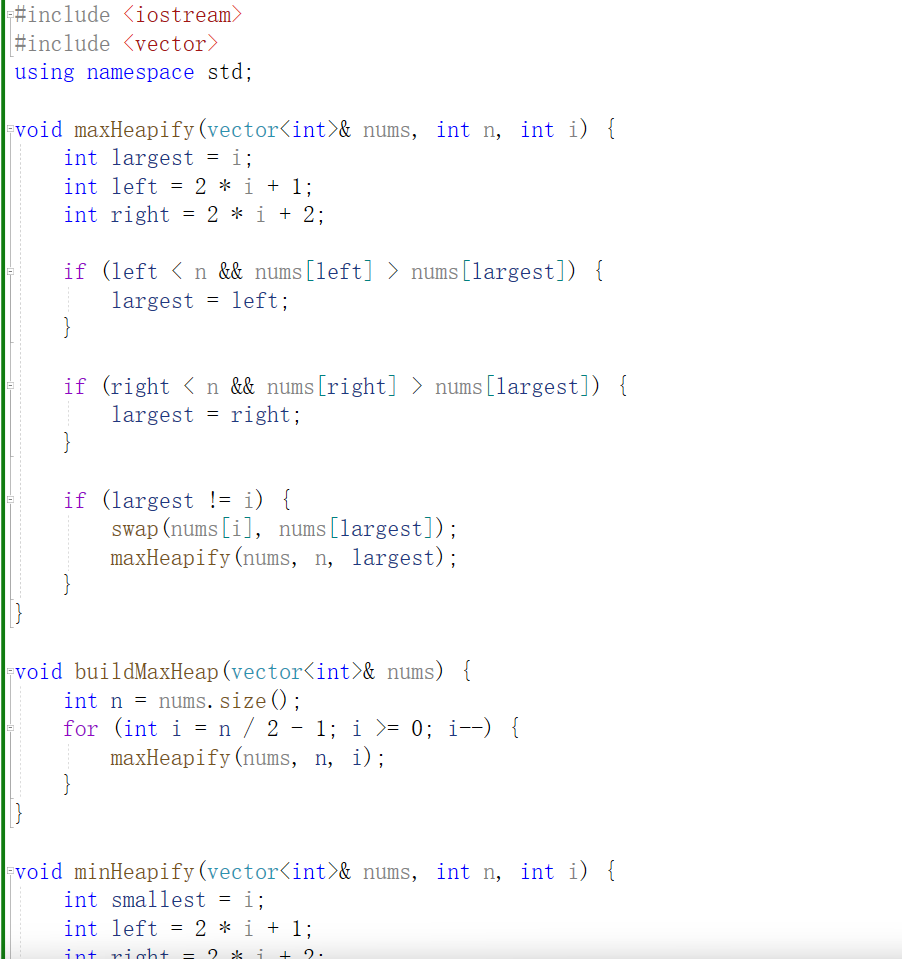
(b) (100,80,90,60,40,20,70,30,35)：这个序列是一个小根堆。因为根节点的值最小，且每个父节点的值都小于或等于其子节点的值。

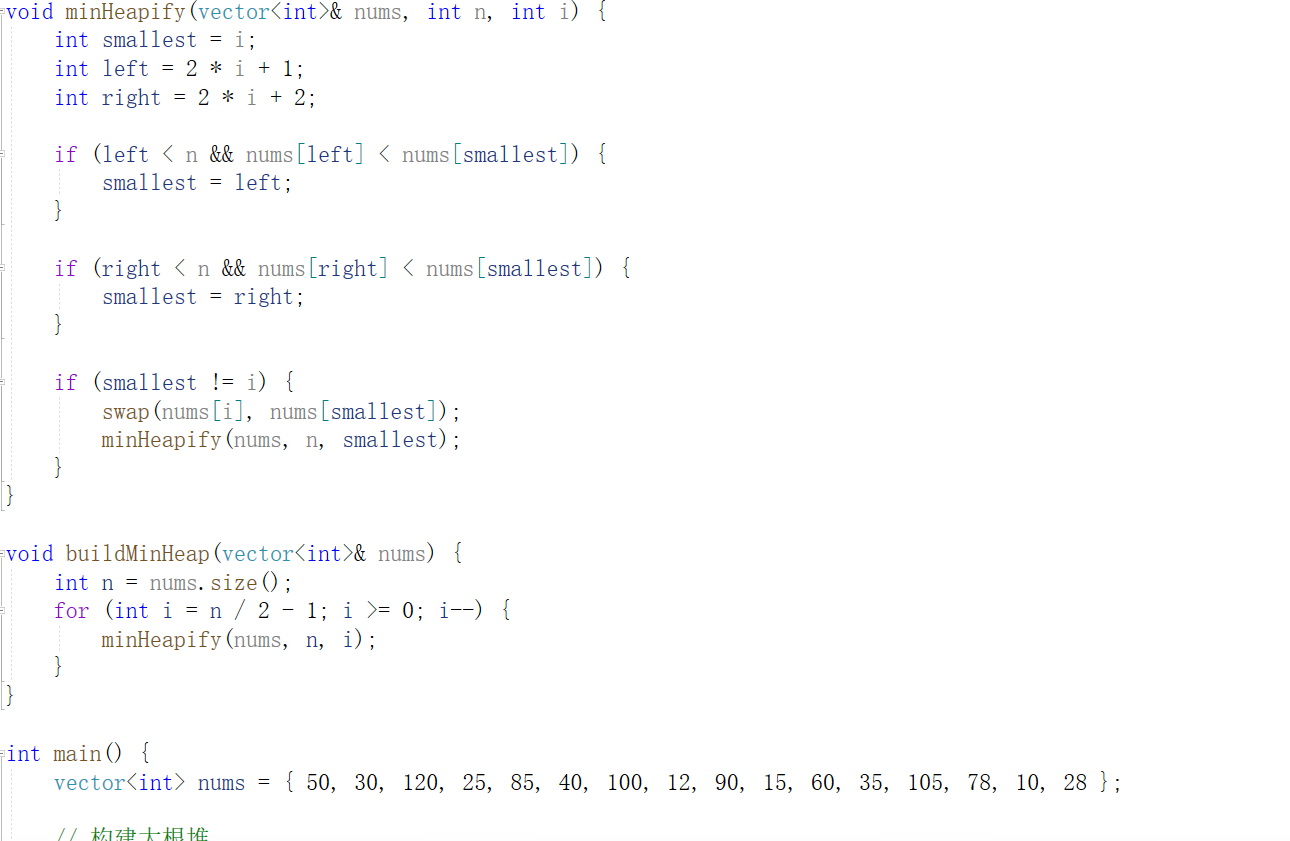
(c) (100,90,80,40,20,70,30,60,35)：这个序列不是堆，因为根节点的值100大于其中的其他所有节点的值。

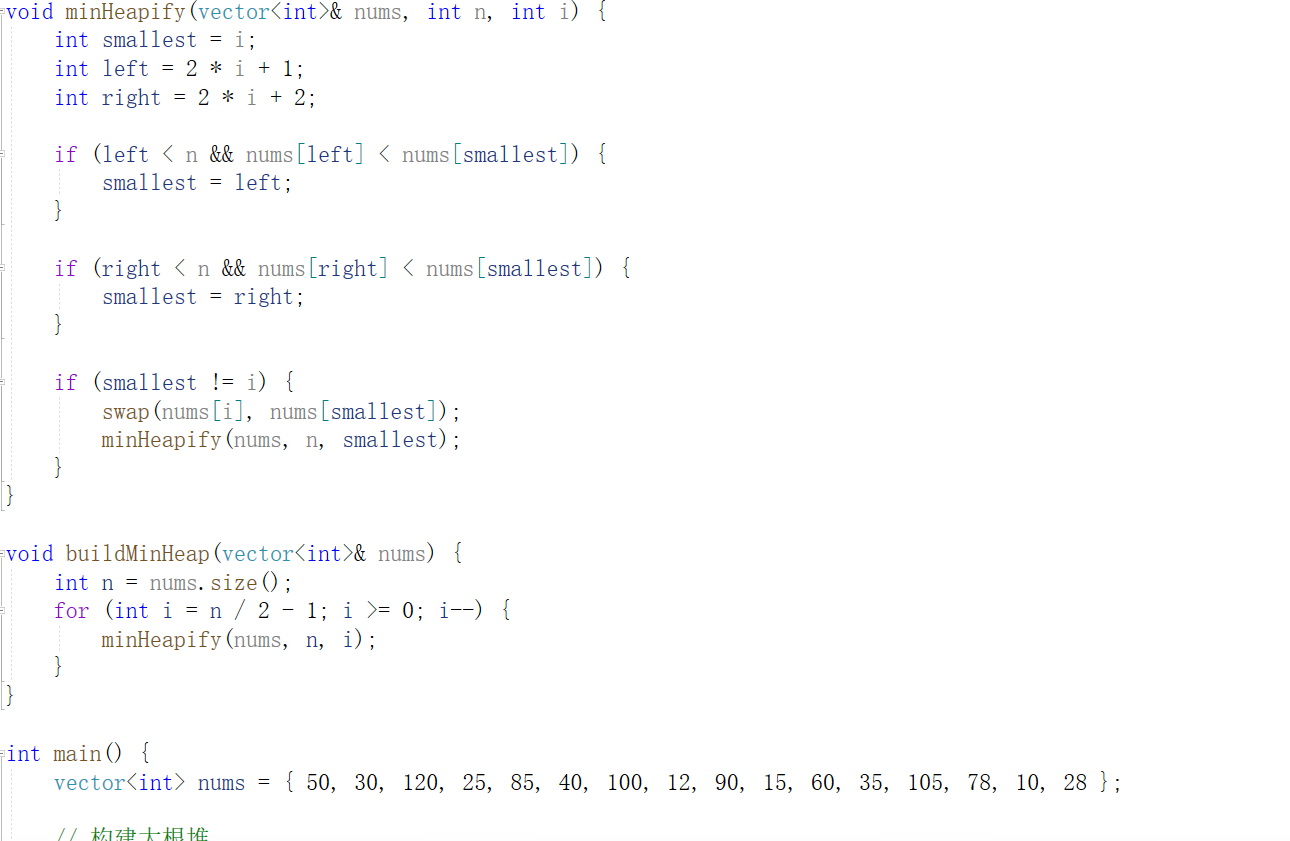
(d) (20,30,40,35,60,80,90,100,70)：这个序列不是堆，因为根节点的值20小于其中的其他所有节点的值。

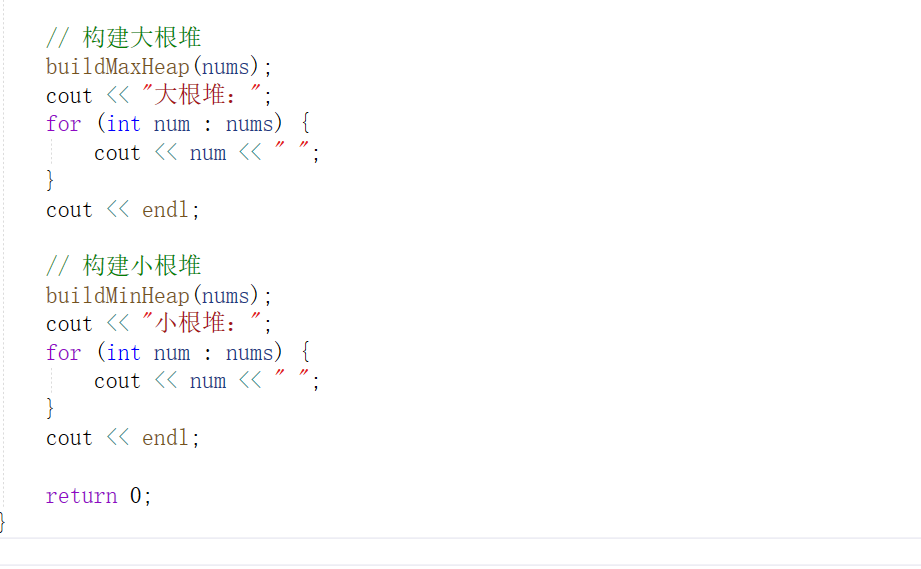
因此，只有序列(b) (100,80,90,60,40,20,70,30,35)是堆，而其他三个序列并不是堆。

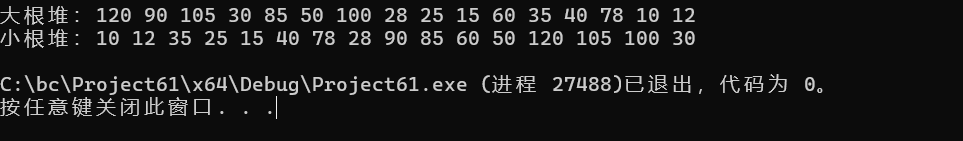
第10题



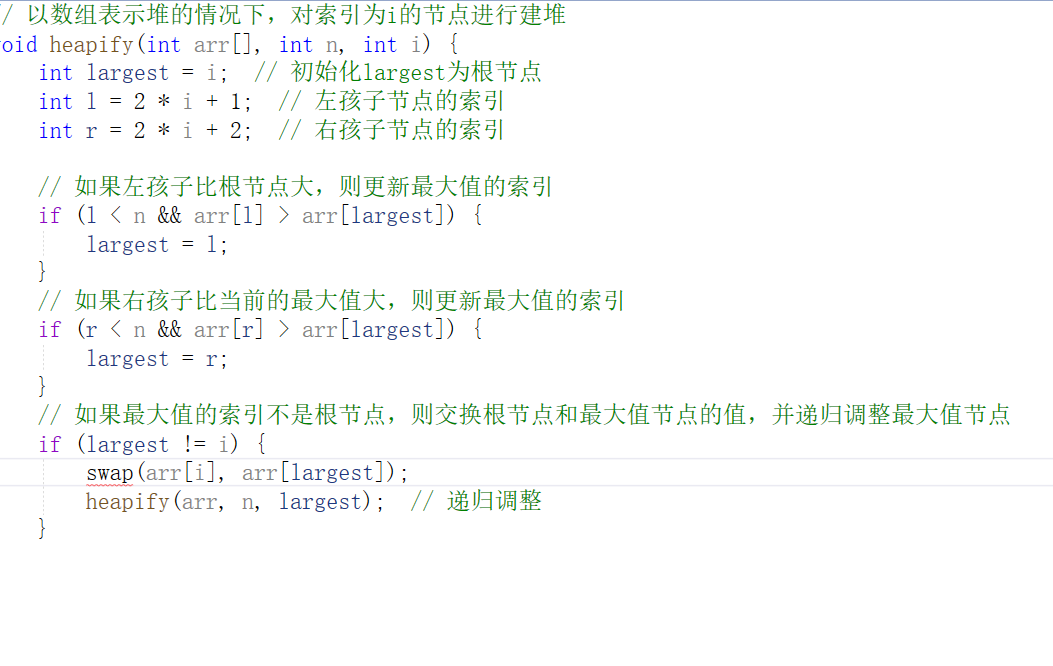








第11题



第12题

堆排序的时间复杂度为O(nlogn)，其中n为数据表的长度。堆排序是一种不稳定的排序算法，它利用堆这种数据结构进行排序。堆的特点是：对于所有的i，arr[i] <= arr[2*i+1]和arr[i] <= arr[2*i+2]，或者arr[i] >= arr[2*i+1]和arr[i] >= arr[2*i+2]。

在使用堆排序来找出最大的10个数时，可以进行以下步骤：

1. 构建一个包含前10个数的小根堆。
2. 遍历剩余的数，若比堆顶元素大，则替换堆顶元素，并重新调整堆结构。
3. 最终得到的堆即为最大的10个数。

通过使用堆排序，可以在O(nlogk)的时间复杂度内找出数据表中最大的k个数，其中n为数据表的长度，k为要找出的最大数的个数。因此，堆排序在这种需求下能够最节省时间。

第13题





