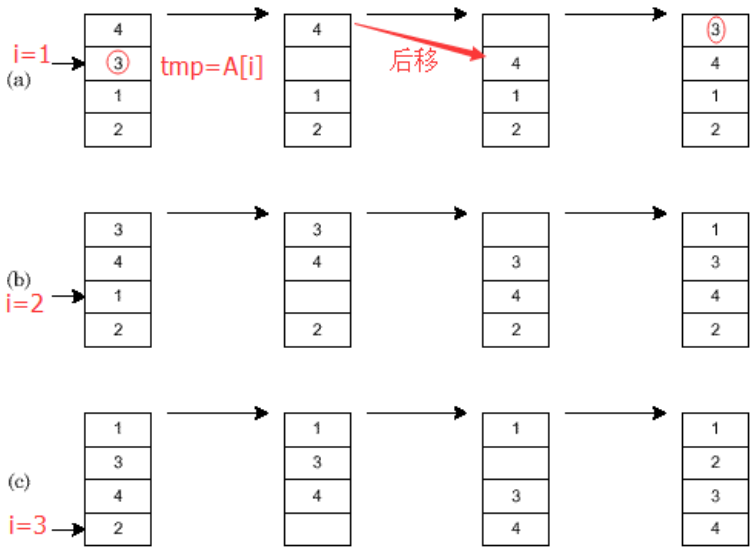
1 直接插入排序

直接插入排序(**Insertion Sort**)的基本思想是：每次将一个待排序的记录，按其关键字大小插入到前面已经排好序的子序列中的适当位置，直到全部记录插入完成为止。

图1演示了对4个元素进行直接插入排序的过程，共需要(a),(b),(c)三次插入。



设初始数组为A[0...N-1]

1. 初始时，A[0]自成1个有序区，无序区为A[1...N-1]。令i=1
2. 将A[i]并入当前的有序区A[0…i-1]中形成A[0…i]的有序区间。
3. i++并重复第2步直到i==N-1。排序完成

```
void insertionSort(int A[], int N)
{
    int j, tmp;
    for (int i = 1; i < N; i++)                    // A[1...N-1]
    {
        tmp = A[i];
        for (j = i - 1; j >= 0 && A[j] > tmp; j--)  // A[0...i-1]
            A[j + 1] = A[j];                       // 后移
        A[j + 1] = tmp;                            // 插入
    }
}

void InsertionSort(int A[], int N)
{
    int tmp, j;
    for (int i = 1; i < N; i++)                    // A[1...N-1]
    {
        tmp = A[i];
        for (j = i; j > 0 && A[j - 1] > tmp; j--)   // A[0...i-1]
            A[j] = A[j - 1];                       // 后移
        A[j] = tmp;                                // 插入
    }
}
```

算法复杂度：*TODO...* O(N^2) 平均情况：(N^2)

---

2 希尔排序

希尔排序的实质就是分组插入排序，该方法又称缩小增量排序，因DL．Shell于1959年提出而得名。

该方法的基本思想是：先将整个待排元素序列分割成若干个子序列（由相隔某个“增量”的元素组成的）分别进行直接插入排序，然后依次缩减增量再进行排序，待整个序列中的元素基本有序（增量足够小，通常为1）时，再对全体元素进行一次直接插入排序。

> **为什么希尔排序不把增量直接设置为1啊？**有位网友回答如下:
>
> 希尔排序的思想是让数组里面任意间隔为h的元素都是有序的，也就是h个相互独立的数组组合在一起。我们要做的就是逐渐的缩小h，让他部分有序，直至全部有序。你所说的增量设置为1是插入排序，插入排序最大的弊端就是太依赖于要排序数组的有序性，排序数组越有序，他排序时间越短,越无序时间越长。希尔排序是他的升级版，希尔排序先让数组部分有序，最后再将h设为1，从而实现全部有序。他权衡了数组的规模性还有有序性，对大型数组，任意排列的数组表现都很好。

以n=10的一个数组49, 38, 65, 97, 26, 13, 27, 49, 55, 4为例

第一次 gap = 10 / 2 = 5

```
49   38   65   97   26   13   27   49   55   4
1A                       1B

     2A                       2B

          3A                       3B

               4A                       4B

                    5A                       5B
```

*1A,1B，2A,2B等为分组标记，数字相同的表示在同一组，大写字母表示是该组的第几个元素， 每次对同一组的数据进行直接插入排序。即分成了五组(49, 13) (38, 27) (65, 49) (97, 55) (26, 4)这样每组排序后就变成了(13, 49) (27, 38) (49, 65) (55, 97) (4, 26)， 下同。*于是:

```
13   27   49   55   4    49   38   65   97   26
```

第二次 gap = 5 / 2 = 2

```
13   27   49   55   4    49   38   65   97   26

1A        1B        1C        1D        1E

     2A        2B        2C        2D        2E
```

分成了两组：(13, 49, 4, 38, 97)和(27， 55， 49， 65， 26)；分别进行直接插入排序，有(4, 13, 38, 49, 97)和(26, 27, 49, 55, 65)；于是：

```
4   26   13   27  38    49    49   55   97   65
```

第三次 gap = 2 / 2 = 1

```
4    26     13    27    38     49     49    55     97     65
1A   1B     1C    1D    1E     1F     1G    1H     1I     1J
```

进行直接插入排序操作，于是:

```
4   13   26   27   38    49   49   55   65   97
```

> 附注：上面希尔排序的步长选择都是从n/2开始，每次再减半，直到最后为1。其实也可以有另外的更高效的步长选择，如果读者有兴趣了解，请参阅维基百科上对希尔排序步长的说明，戳这里。

希尔排序的程序如下，可以看出来希尔排序的主程序（当内层两个for循环中的增量increment为1时）为直接插入排序的代码：

```
void shellSort(int A[], int N)
{
    int j, tmp;
    for (int increment = N / 2; increment > 0; increment /= 2)
    {
```

```
        // 插入排序(increment == 1)
        for (int i = increment; i < N; i++)
        {
            tmp = A[i];
            for (j = i; j >= increment; j -= increment)
            {
                if (tmp < A[j - increment])
                    A[j] = A[j - increment];
                else
                    break;
            }
            A[j] = tmp;
        }
    }
}
```

3 归并排序

归并排序是建立在**归并操作**上的一种有效的排序算法，即该算法是采用分治法（Divide and Conquer）的一个非常典型的应用，最坏时间复杂度为O(NlogN)。归并排序的基本操作为归并操作，即合并两个已有序的列表称为新的有序列表。

首先考虑如何将两个**有序**数列合并？这个很简单，只要从比较这两个数列的第一个数，谁小就先取谁，取了后就在对应数列中删除即可。然后再进行比较，如果到达某个数列的末端，则直接将另一个数列了的数据依次取出即可。例如，合并有序数列1,13,24,26和2,15,27,38的流程如下：

```
          A            |           B             |              C
1    13    24    26   |  2    15    27    38    |
↑                       |  ↑                      | ↑
第一次比较1<2，把1放入数列C中，数列A和数列C的下标加1，下同。
1    13    24    26   |  2    15    27    38    | 1
     ↑                  |  ↑                                  ↑
第二次比较2<13，把2放入数列C中，数列B和数列C的下标加1...
1    13    24    26   |  2    15    27    38    | 1    2
     ↑                        ↑                                  ↑
1    13    24    26   |  2    15    27    38    | 1    2    13
           ↑                  ↑                                       ↑
1    13    24    26   |  2    15    27    38    | 1    2    13    15
           ↑                        ↑                                       ↑
1    13    24    26   |  2    15    27    38    | 1    2    13    15    24
                 ↑                  ↑                                            ↑
1    13    24    26   |  2    15    27    38    | 1    2    13    15    24    26
                     ↑              ↑                                                 ↑
数列A中的数据比较完毕，把数列B余下数据依次复制到数列C中，得到最后的排序结果：
                                                 1    2    13    15    24    26   27    38
```

代码如下：

```
// 将有序数组A[]和B[]合并到C[]中
void merge(int A[], int m, int B[], int n, int C[])
{
    int i = 0, j = 0, k = 0;

    while (i < m && j < n)
    {
        if (A[i] < B[j])
            C[k++] = A[i++];
        else
            C[k++] = B[j++];
    }

    while (i < m)
        C[k++] = A[i++];

    while (j < n)
        C[k++] = B[j++];
}
```

可以看出合并有序数列的效率是比较高的，可以达到O(n)。

解决了上面的合并有序数列问题，再来看归并排序，其的基本思路就是将数组分成二组A，B，如果这二组组内的数据都是有序的，那么就可以很方便的将这二组数据进行排序。如何让这二组组内数据有序了？

可以将A，B组各自再分成二组。依次类推，当分出来的小组只有一个数据时，可以认为这个小组组内已经达到了有序，然后再合并相邻的二个小组就可以了。这样通过先递 **归** 的分解数列，再合 **并** 数列就完成了**归并**排序。

```
// 将有二个有序数列A[left...center]和A[center + 1...right]合并。
void MergeArray(int A[], int left, int right, int center, int Tmp[])
{
    int i = left, j = center + 1, k = 0;
    int m = center, n = right;

    while (i <= m && j <= n)
    {
        if (A[i] <= A[j])
            Tmp[k++] = A[i++];
        else
            Tmp[k++] = A[j++];
    }

    while (i <= m)
        Tmp[k++] = A[i++];

    while (j <= n)
        Tmp[k++] = A[j++];
    // copy Tmp array back
    for (i = 0; i < k; i++)
        A[left + i] = Tmp[i];
}

void MSort(int A[], int left, int right, int Tmp[])
{
    int center;
    if (left < right)
    {
        center = (left + right) / 2;
        MSort(A, left, center, Tmp);                // 左边有序
        MSort(A, center + 1, right, Tmp);           // 右边有序
        MergeArray(A, left, right, center, Tmp);    // 合并
    }
}

void mergeSort(int A[], int N)
{
    // 临时数组
    int *tmpArray = (int*)malloc(N* sizeof (int));
    if (tmpArray)
    {
        MSort(A, 0, N - 1, tmpArray);
        free(tmpArray);
    }
}
```

归并排序的效率是比较高的，设数列长为N，将数列分开成小数列一共要logN步，每步都是一个合并有序数列的过程，时间复杂度可以记为O(N)，故一共为O(N*logN)。因为归并排序每次都是在相邻的数据中进行操作，所以归并排序在O(N*logN)的几种排序方法（快速排序，归并排序，希尔排序，堆排序）也是效率比较高的。

---

4 快速排序

4.1 快速排序算法及其思想

快速排序由于排序效率在同为O(N*logN)的几种排序方法中效率较高，因此经常被采用，再加上快速排序思想----分治法也确实实用，因此很多软件公司的笔试面试，包括像腾讯，微软等知名IT公司都喜欢考这个，还有大大小的程序方面的考试如软考，考研中也常常出现快速排序的身影。 总的说来，要直接默写出快速排序还是有一定难度的，因为本人【morewindows】就自己的理解对快速排序作了下白话解释，希望对大家理解有帮助，达到快速排序，快速搞定。

快速排序是C.R.A.Hoare于1962年提出的一种划分交换排序。它采用了一种分治的策略，通常称其为分治法(Divide-and-ConquerMethod)。

该方法的基本思想是：

1．先从数列中取出一个数作为基准数(或枢纽元)。

2．分区过程，将比这个数大的数全放到它的右边，小于或等于它的数全放到它的左边。

3．再对左右区间重复第二步，直到各区间只有一个数。

虽然快速排序称为分治法，但分治法这三个字显然无法很好的概括快速排序的全部步骤。因此我的对快速排序作了进一步的说明：**挖坑填数 + 分治法**：

先来看实例吧，定义下面再给出（最好能用自己的话来总结定义，这样对实现代码会有帮助）。

4.2 快速排序例程

以一个数组作为示例，取区间第一个数为基准数（这种取法并不好，下面有讨论，这里聚焦于对算法本身的理解）。

index	0	1	2	3	4	5	6	7	8	9
A	**72**	6	57	88	60	42	83	73	48	85
	i↑									*j*↑

初始时，选取第一个元素72为枢纽元（或基准数），i = 0; j = 9; X = A[i] = 72。由于已经将A[0]中的数保存到X中，可以理解成在数组A[0]上挖了个坑，可以将其它数据填充到这来。

从j开始向前找一个比X小或等于X的数。当j=8，符合条件，将A[8]挖出再填到上一个坑A[0]中。A[0]=A[8]; 结果如下:

index	0	1	2	3	4	5	6	7	8	9
A	**48**	6	57	88	60	42	83	73	**?**	85
	i↑								***j***↑	

i++; 这样一个坑A[0]就被搞定了，但又形成了一个新坑A[8]，这怎么办了？简单，再找数字来填A[8]这个坑。这次从i开始向后找一个大于X的数，当i=3，符合条件，将A[3]挖出再填到上一个坑中A[8]=A[3];

index	0	1	2	3	4	5	6	7	8	9
A	**48**	6	57	**?**	60	42	83	73	**88**	85
				i↑					*j*↑	

同理，j--，j向前走直到找出≤72的数字42，当j = 5时满足，填坑A[3] = A[5]，此时的新坑在j = 5处，如下表所示：

index	0	1	2	3	4	5	6	7	8	9
A	**48**	6	57	**42**	60	**?**	83	73	**88**	85
				i↑		***j***↑				

同样的，让i一直往后走直到遇到≥72的数字，然而i越过了j出现了交叉，那么此时第一趟排序完毕，结果如下表所示，坑?处应填入枢纽元72。因此再对A[0…4]和A[6…9]这两个子区间重复上述步骤就可以了。

index	0	1	2	3	4	5	6	7	8	9
A	**48**	6	57	**42**	60	**?**	83	73	**88**	85
						j↑	***i***↑			

对挖坑填数进行总结，*先从后向前找，再从前向后找（这是因为数组的第一个数为基准数；如果取最后一个数为基准数，此时应先从前向后找）*：

1. i = left; j = right; 将基准数挖出形成第一个坑A[i]。
2. j--由后向前找比它小的数，找到后挖出此数填前一个坑A[i]中。
3. i++由前向后找比它大的数，找到后也挖出此数填到前一个坑A[j]中。
4. 再重复执行2，3二步，直到i==j，将基准数填入A[i]中。

照着这个总结很容易实现挖坑填数的代码：

```
// 快速排序驱动程序，返回调整后基准数的位置
int QSort(int A[], int left, int right)
{
    int i = left, j = right;
    //有的书上是以中间的数作为基准数的，要实现这个方便非常方便，直接将中间的数和第一个数进行交换就可以了。
    int x = A[left]; // A[left]为第一个坑

    while (i < j)
    {
        while (i < j && A[j] >= x)// 从右向左找小于x的数来填A[i]
            j--;
        if (i < j){
            A[i] = A[j];// 将A[j]填到A[i]中，A[j]就形成了一个新的坑
            i++;
        }

        while (i < j && A[i] < x)// 从左向右找大于或等于x的数来填A[j]
            i++;
        if (i < j){
            A[j] = A[i];// 将A[i]填到A[j]中，A[i]就形成了一个新的坑
            j--;
        }
    }
    // 退出时，i等于j。将x填到这个坑中
    A[i] = x;

    return i;
}

void quickSort(int A[], int left, int right)
{
    if (left < right){ //  i < j
        int i = QSort(A, left, right);
        quickSort(A, left, i - 1);
        quickSort(A, i + 1, right);
    }
}
```

注：C++STL中sort() 算法底层使用快速排序。