提起算法，很多人都会有一个感觉，就是不太容易学会，却很容易遗忘。其实原因无外乎有这么几个：

1. 理解算法的整体设计时，过于追求某种语言的实现细节；
2. 用某种语言实现时，又不知道该如何实现算法的整体设计；
3. 只专注让代码工作，却没有话足够多的时间研究算法的实现细节；

固然学习算法不是一个容易的过程，因此，我们更是得注重学习的方法和质量。例如，说两个最基本的排序方法，选择排序和插入排序。其实，如果你按照显示中我们数数的方法来记住它们的思路，就很容易学了不再忘记。

首先，它们都假设只有一个数字的序列是已经排序好的。在这一步，我们先写上这两种算法的函数签名：

```
template<typename T, typename Pred = decltype(less<T>())>
void insertion_sort(vector<T> &v, Pred pred = Pred());

template<typename T, typename Pred = decltype(less<T>())>
void selection_sort(vector<T> &v, Pred pred = Pred());
```

可以看到，它们除了名字不同之外，没有任何差异，这里我们使用了`vector`作为数列的存储，默认执行升序排序。

其次，写出算法的第一行代码，其实往往写算法的第一行很重要，有时顺利写出了第一行，就能顺利带出来整个算法的实现。而对这两个排序来说，它们的第一行，都是获取元素的个数：

```
template<typename T, typename Pred = decltype(less<T>())>
void insertion_sort(vector<T> &v, Pred pred = Pred()) {
    int n = v.size();
}

template<typename T, typename Pred = decltype(less<T>())>
void selection_sort(vector<T> &v, Pred pred = Pred()) {
    int n = v.size();
}
```

第三，拿到要排序的所有内容了，接下来就是两种不同的数数方式了，对于选择排序来说，理解这个过程有一个关键点：第一次遍历，找到的最小的数字，第二次，是次小的数字，和第一次的结果拼起来，最后一次，是序列中最大的，拼在结果的末尾。整个序列就排好了。

这个过程，是通过一个嵌套的`for`循环实现的：

```
template<typename T, typename Pred = decltype(less<T>())>
void selection_sort(vector<T> &v, Pred pred = Pred()) {
    int n = v.size();

    for (int i = 0; i < n; ++i) {
        int candidate = i;
        for (int j = i + 1; j < n; ++j) {
            if (pred(v[j], v[candidate])) { candidate = j; }
        }

        std::swap(v[i], v[candidate]);
    }
}
```

其中，外层的`for`循环，`i`从0开始，表示**只有一个元素的数组是已经排序好的**。在外层循环内部，`candidate`表示下一个要拼接进来的元素所在的位置。

在内层`for`循环里`int j = i + 1`表示，从当前已经排序好的位置的下一个位置开始向后遍历，只要我们的假设（这里默认是小于比较）成立，就表示我们找到了更小的值，我们就更新这次要拼接的数字所在的位置。这样内层循环结束了，就可以确定当前所有剩余数字中最小的一个了，我们把它和candidate交换完成拼接。

说白了，选择排序是一个从前向后，不断刷小数字的过程。

第四，而插入排序，正好和这个过程相反，它很像洗牌，当我们手里只有一张牌的时候，一定就是排序好的。然后，我们把桌子上的牌理解成所有待排序的，每抓一张，我们就给他找到合适的位置排好，这样所有的牌都抓完了，自然就是排好的状态的。

这个过程，同样要用一个双层循环来模拟：

```
template<typename T, typename Pred = decltype(less<T>())>
void insertion_sort(vector<T> &v, Pred pred = Pred()) {
    int n = v.size();

    for (int i = 1; i < n; ++i) {
        for (int j = i; j > 0 && pred(v[j], v[j - 1]); --j) {
            std::swap(v[j], v[j - 1]);
        }
    }
}
```

这里，外层循环从`i = 1`开始，表示，从手里的第二张牌开始处理，在内层循环里，我们从牌的自然位置开始不断向前查找，只要摸来的牌比当前位置的牌小，我们就交换两张牌的位置，并

且，一旦条件不满足了，就表示当前位置，就是这张牌的位置了。

因此，插入排序是一个从后向前，不断实现相邻交换以找到最合适位置的过程。

完整代码

```
#include <algorithm>
#include <vector>
#include <functional>
#include <iostream>
using namespace std;

template<typename T, typename Pred = decltype(less<T>())>
void insertion_sort(vector<T> &v, Pred pred = Pred()) {
    int n = v.size();

    for (int i = 1; i < n; ++i) {
        for (int j = i; j > 0 && pred(v[j], v[j - 1]); --j) {
            std::swap(v[j], v[j - 1]);
        }
    }
}

template<typename T, typename Pred = decltype(less<T>())>
void selection_sort(vector<T> &v, Pred pred = Pred()) {
    int n = v.size();

    for (int i = 0; i < n; ++i) {
        int candidate = i;
        for (int j = i + 1; j < n; ++j) {
            if (pred(v[j], v[candidate])) { candidate = j; }
        }

        std::swap(v[i], v[candidate]);
    }
}

int main() {
    vector<int> v { 8, 3, 6, 9, 1, 7, 2, 5, 10, 3, 22 };

    insertion_sort(v);
    insertion_sort<int, greater<>>(v);
    selection_sort(v);
    selection_sort<int, greater<>>(v);

    for (auto &i: v) {
```

```
        cout<<i<<" ";
    }

    cout<<endl;
    return 0;
}
```