```
//直接插入排序
void InserSort(int R[], int n)
{
    int i, j;
    int temp;
    for(i = 1; i < n; ++i)
    {
        temp = R[i];
        j = i - 1;
        while(j >= 0 && temp < R[j])
        {
            R[j + 1] = R[j];
            --j;
        }
        R[j + 1] = temp; // 插入
    }
}

// 折半查找排序
void InsertSort(int A[]){
    int n = A.size();
    int low, high;
    for(int i = 2; i < n; ++i){
        A[0] = A[i];
        low = 1;
        high = i - 1;
        while(low < high){
            mid = (low + high) / 2;
            if(A[mid] > A[0])
                high = mid - 1;
            else
                low = mid + 1;
        }
        for(j = i - 1; j >= high + 1; --j)
            A[j + 1] = A[j];
        A[high + 1] = A[0];
    }
}
```

```
// ShellSort
/*
    记录前后位置的增量是dk，不是1
    A[0]暂存单元，不是哨兵，当 j<=0 时，插入位置已到
*/
void ShellSort(int A[]){
    int n = A.size();
    for(int dk = n / 2; dk >= 1; dk = dk / 2)
        for(int i = dk + 1; i <= n; ++i)
            if(A[i] < A[i - dk]){
                A[0] = A[i];
                for(int j = i - dk; j > 0 && A[0] < A[j]; j -= dk)
                    A[j + dk] = A[j];
                A[j + dk] = A[0];
            }
}

// 冒泡排序
void BubbleSort(int R[]){
    int n = R.size();
    for(int i = n - 1; i >= 1; --i){
        int flag = 0;
        for(j = 1; j <=i; ++j){
            if(R[j - 1] > R[j]){
                int temp = R[j];
                R[j] = R[j - 1];
                R[j - 1] = temp;
                flag = 1;
            }
        }
        if(flag == 0)
            return;
    }
}
```

```
// 快速排序
void QuickSort(int R[], int low, int high){
    int temp;
    int i = low;
    int j = high;
    if(low < hhigh){
        temp = R[low];
        while(i < j){
            while(j > i && R[j] >= temp) --j; //从右往左扫描，找到一个小于temp的关键字
            if(i < j){
                R[i] = R[j];
                ++i;
            }
            while(i < j && R[i] < temp) ++i;
            if(i < j){
                R[j] = R[i];
                --j;
            }
        }
        R[i] = temp;
        QuickSort(R, low, i - 1); // 递归地对temp左边进行排序
        QuickSort(R, i + 1, high);
    }
}

// 简单选择排序
void SelectSort(int R[], int n){
    int i, j, k;
    int temp;
    for(int = 0; i < n; ++i){
        k = i;
        //从无序列中选择最小的一个关键字
        for(j = i + 1; j < n; ++j){
            if(R[k] > R[j])
                k = j;
        }
        // 最小关键字与无序序列第一个位置交换
        temp = R[i];
        R[i] = R[k];
        R[k] = temp;
    }
}
```

```
// 堆排序
/**
    调整函数
*/
void Sift(int R[], int low, int high){
    int i = low, j = i * 2; //R[j]是R[i]的左孩子结点
    int temp = R[i];
    while(j <= high){
        if(j < high && R[j] < R[j + 1]){  //若右孩子较大，则j指向右孩子
            ++j;
        }
        else if(temp < R[j]){    //将R[j]调整到双亲节点上
            R[i] = R[j];         //修改i和j的值，以便继续向下调整
            i = j;
            j = i * 2;
        }else{
            break;  // 调整结束
        }
    }
    R[i] = temp;  // 被调整的结的值放入最终位置
}

/**
    堆排函数
*/
void heapSort(int R[], int n){
    int i;
    int temp;
    for(i = n / 2; i >= 1; --i)
        Sift(R, i, n);
    for(i = n; i >= 2; --i){
        /* 换出了根节点中的关键字，将其放入最终位置*/
        temp = R[1];
        R[1] = R[i];
        R[i] = temp;
        Sift(R, 1, i - 1); //在减少了1个关键字地无序序列中进行调整
    }
}
```

```
void mergeSort(int A[], int low, int high){
    if(low < high){
        int mid = (low + high) / 2;
        mergeSort(A, low, mid);
        mergeSort(A, mid + 1, high);
        merge(A, low, mid, high); // 将A中low到mid和mid + 1到high范围内两段有序序列归并成一段有序序列
    }
}
```