```cpp
#include<iostream>
using namespace std;

#define ElemType int
#define MAX_SIZE 10

//-----------------------------------------------------插入排序
//直接插入排序        //O(n^2)，稳定
void InsertSort(ElemType R[], int n) {
    int i, j;
    for (i = 2; i <= n; i++)
        if (R[i] < R[i - 1]) {
            R[0] = R[i];
            for (j = i - 1; R[0] < R[j]; --j)//R[0]为哨兵
                R[j + 1] = R[j];
            R[j + 1] = R[0];  //注：这里是R[j+1]
        }//if
}

//折半插入排序        //O(n^2)，稳定
void BinaryInsertSort(ElemType R[], int n) {
    int i, j, low, high, mid;
    for (i = 2; i <= n; i++)
        if (R[i] < R[i - 1]) {
            R[0] = R[i];
            low = 1; high = i - 1;
            while (low <= high) {//出循环时high+1 == low
                mid = (low + high) / 2;
                if (R[mid] > R[0])
                    high = mid - 1;
                else
                    low = mid + 1;
            }//while，出循环肯定能找到，R[i]应该插入的位置
            //出循环后R[high]<R[0]<=R[low]
            for (j = i - 1; j >= low; --j)
                R[j + 1] = R[j];
            R[j + 1] = R[0];
        }//if
}

//希尔排序          //最坏O(n^2)，不稳定，统计O(n^1.3)未证明
void ShellSort(ElemType R[], int n) {
    int i, j, dk;
    for (dk = n / 2; dk >= 1; dk = dk / 2) {
        for (int m = 0; m < 10; m++)
            cout << R[m] << "  ";
        cout << endl;
        for (i = dk + 1; i <= n; ++i)
            if (R[i] < R[i - dk]) {
                R[0] = R[i];
                //R[0]不再作为哨兵
                for (j = i - dk; j > 0 && R[0] < R[j]; j -= dk)
                    R[j + dk] = R[j];
                R[j + dk] = R[0];
            }//if
        }
}

//-----------------------------------------------------交换排序
//冒泡排序      //O(n^2), 稳定
void BubbleSort(ElemType R[], int n) {
    bool flage;
    int i, j;
    for (i = 1; i <= n; ++i) {
        flage = false;  //本趟排序是否发生交换标记
        for (j = n; j > i; --j)
            if (R[j] < R[j - 1]) {
                R[0] = R[j];
                R[j] = R[j - 1];
                R[j - 1] = R[0];
                flage = true;
            }
        if (flage == false)
            return;
    }
}

//快速排序      //O(nlog2n), 不稳定, 递归栈 O(log2n)
int partition(ElemType R[], int low, int high) {
    R[0] = R[low];
    while (low < high) {
        while (low < high && R[0] <= R[high])
            high--;
        if (low < high)
            R[low] = R[high];
        while (low < high && R[low] <= R[0])
            low++;
        if (low < high)
            R[high] = R[low];
    }
    R[low] = R[0];
    return low;
}
void QuickSort(ElemType R[], int low, int high) {
    if (low < high) {
        int part = partition(R, low, high);
        QuickSort(R, low, part - 1);
        QuickSort(R, part + 1, high);
    }
}

//-----------------------------------------------------选择排序
//简单选择      //O(n^2) 不稳定
void SelectSort(ElemType R[], int n) {
    int i, j, minp;
    for (i = 1; i < n; ++i) {
        minp = i;
        for (j = i + 1; j <= n; ++j)
            if (R[j] < R[minp])
                minp = j;
        if (minp != i) {
            R[0] = R[i];
            R[i] = R[minp];
            R[minp] = R[0];
        }
    }
}

//堆排序       //O(nlog2n) 不稳定
void AdjustDown(ElemType R[], int k, int n) {
    //表长为n，从第k个顶点相下调整
    R[0] = R[k];
    for (int i = k * 2; i <= n; i = i * 2) {
        if (i < n&& R[i] < R[i + 1])
            i++;    //选取较大的孩子，和双亲比较
        if (R[0] >= R[i])
            break;
        else {
            R[k] = R[i];
            k = i;
        }
    }//for
    R[k] = R[0];
}
void BuildMaxHeap(ElemType R[], int n) {
    for (int i = n / 2; i > 0; --i)
        AdjustDown(R, i, n);
}
void HeapSort(ElemType R[], int n) {
    BuildMaxHeap(R, n);
    for (int i = n; i > 1; --i) {
        R[0] = R[i];    R[i] = R[1];    R[1] = R[0];
        AdjustDown(R, 1, i - 1);
    }//for
}

//-----------------------------------------------------归并排序
ElemType* B = new ElemType[MAX_SIZE];       //定义辅助数组
void Merge(ElemType R[], int low, int mid, int high) {
    int i, j, k;
    for (k = low; k <= high; k++)
        B[k] = R[k];                //复制R中的元素到B
    for (i = low, j = mid + 1, k = i; i <= mid && j <= high; k++) {
        if (B[i] <= B[j])              //比较B左右两端的元素
            R[k] = B[i++];          //将较小值赋值到R中
        else
            R[k] = B[j++];
    }//for
    while (i <= mid)                   //若第一个表未检查完，复制
```

```
        R[k++] = B[i++];
    while (j <= high)                 //若第二个表未检查完，复制
        R[k++] = B[j++];
}
void MergeSort(ElemType R[], int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;         //从中间剖分
        MergeSort(R, low, mid);         //对左子序列进行递归排序
        MergeSort(R, mid + 1, high);      //对右子序列进行递归排序
        Merge(R, low, mid, high);       //归并
    }//if
}

int main() {
    ElemType R[MAX_SIZE] = { 0,2,5,4,3,1,6,7,9,8 };
    for (int i = 0; i < 10; i++)
        cout << R[i] << "  ";
    MergeSort(R, 1, MAX_SIZE - 1);
    for (int i = 0; i < 10; i++)
        cout << R[i] << "  ";
}
```