归并排序：

```
#include<iostream>
using namespace std;
#define MAX_SIZE 100005
#define INF 2147483647
typedef int KeyType;
typedef struct {
    KeyType key;
}RedType;
typedef struct {
    RedType r[MAX_SIZE + 1];
    int length;
}SqList;
RedType L[MAX_SIZE / 2 + 2], R[MAX_SIZE / 2 + 2];
void Merge(SqList&S, int left, int mid, int right)
{
    int n1 = mid - left, n2 = right - mid;
    for (int i = 0; i < n1; i++)
        L[i] = S.r[left + i];
    for (int i = 0; i < n2; i++)
        R[i] = S.r[mid + i];
    int i = 0, j = 0;
    R[n2].key = L[n1].key = INF;
    for (int k = left; k < right; k++)
    {
        if (L[i].key <= R[j].key)
            S.r[k] = L[i++];
        else
            S.r[k] = R[j++];
    }
}
void MergeSort(SqList&S, int left, int right)
{
    if (left + 1 < right)
    {
        int mid = (left + right) / 2;
        MergeSort(S, left, mid);
        MergeSort(S, mid, right);
        Merge(S, left, mid, right);
    }
}
int main()
{
    int n;
    SqList L;
    cin >> n;
    L.length = n;
    for (int i = 0; i < L.length; i++)
        cin >> L.r[i].key;
    MergeSort(L, 0, L.length);
    for (int i = 0; i < L.length; i++)
        cout << L.r[i].key << " ";
    cout << endl;
    return 0;
}
```

堆排序：

```
#include<iostream>
using namespace std;
#define MAX_SIZE 100005
typedef int KeyType;
typedef struct {
    KeyType key;
}RedType;
typedef struct {
    RedType r[MAX_SIZE + 1];
    int length;
}SqList;
RedType temp;
typedef SqList HeapType;
int  pivotkey;
void HeapAdjust(HeapType &H, int s, int m)
{
    int j;
    RedType rc = H.r[s];
    for (j = 2 * s; j <= m; j *= 2)
    {
        if (j < m&&H.r[j].key < H.r[j + 1].key)j++;
        if (!(rc.key < H.r[j].key))break;
        H.r[s] = H.r[j];
        s = j;
    }
    H.r[s] = rc;
}
void HeapSort(HeapType&H)
```

```
{
    RedType temp;
    for (int i = H.length / 2; i > 0; i--)
        HeapAdjust(H, i, H.length);//建立初始大顶堆;
    for (int i = H.length; i > 1; i--)
    {
        temp = H.r[1];
        H.r[1] = H.r[i];
        H.r[i] = temp;
        HeapAdjust(H, 1, i - 1);//将1~i-1重新调整为大顶堆;
    }
}
int main()
{
    int n;
    HeapType H;
    cin >> n;
    H.length = n;
    for (int i = 1; i <= H.length; i++)
        cin >> H.r[i].key;
    HeapSort(H);
    for (int i = 1; i <= H.length; i++)
        cout << H.r[i].key << " ";
    return 0;
}
```

希尔排序:

```
#include<iostream>
using namespace std;
#define MAX_SIZE 100005
typedef int KeyType;
typedef struct {
    KeyType key;
}RedType;
typedef struct {
    RedType r[MAX_SIZE + 1];
    int length;
}SqList;

void ShellInsert(SqList &L, int gap)
```

```
{
    int j;
    for (int i = gap + 1; i <= L.length; i++)
    {
        if (L.r[i].key < L.r[i - gap].key)
        {
            L.r[0] = L.r[i];
            for (j = i - gap; j > 0 && L.r[0].key < L.r[j].key; j = j
- gap)
                L.r[j + gap] = L.r[j];
            L.r[j + gap] = L.r[0];
        }
    }
}
void ShellSort(SqList&L, int Delta[], int t)
{
    for (int k = 0; k < t; k++)
        ShellInsert(L, Delta[k]);
}
int main()
{
    int n;
    SqList L;
    cin >> n;
    L.length = n;
    for (int i = 1; i <= L.length; i++)
        cin >> L.r[i].key;
    int t = 0, Delta[MAX_SIZE];
    L.length /= 2;
    while (L.length)
    {
        Delta[t] = L.length;
        L.length /= 2;
        t++;
    }
    L.length = n;
    ShellSort(L, Delta, t);
    for (int i = 1; i <= L.length; i++)
        cout << L.r[i].key << " ";
    return 0;
}
```

快速排序 1：

```
#include<iostream>
using namespace std;
#define MAX_SIZE 100005
typedef int KeyType;
typedef struct {
    KeyType key;
}RedType;
typedef struct {
    RedType r[MAX_SIZE + 1];
    int length;
}SqList;
RedType temp;
int pivotloc;
int  pivotkey;
int ppp(SqList&L, int high, int low)
{
    pivotkey = L.r[low].key;//枢轴记录关键字
    while (low < high)
    {
        while (low < high&&L.r[high].key >= pivotkey)
            high--;
        temp = L.r[low];
        L.r[low] = L.r[high];
        L.r[high] = temp;
        while (low < high&&L.r[low].key <= pivotkey)
            low++;
        temp = L.r[low];
        L.r[low] = L.r[high];
        L.r[high] = temp;
    }
    return low;
}
int QuickSort(SqList &L, int low, int high)
{
    if (low < high)
    {
        pivotloc = ppp(L, high, low);
        QuickSort(L, low, pivotloc - 1);//低子表进行递归
        QuickSort(L, pivotloc + 1, high);//高子表进行递归
    }
}
int main()
```

```
{
    int n;
    SqList L;
    cin >> n;
    L.length = n;
    for (int i = 1; i <= L.length; i++)
        cin >> L.r[i].key;
    QuickSort(L, 1, L.length);
    for (int i = 1; i <= L.length; i++)
        cout << L.r[i].key << " ";
    return 0;
}
```

快速排序 2:

```
#include<iostream>
using namespace std;
#define MAX_SIZE 100005
typedef int KeyType;
typedef struct {
    KeyType key;
}RedType;
typedef struct {
    RedType r[MAX_SIZE + 1];
    int length;
}SqList;
RedType temp;
int  pivotkey;
void QuickSort(SqList&L, int low, int high)
{
    if (low < high)
    {
        int pl = low, ph = high;
        pivotkey = L.r[low].key;//枢轴记录关键字
        RedType p = L.r[low];
        while (low < high)
        {
            while (low < high&&L.r[high].key >= pivotkey)
                high--;
            if (low < high)
                L.r[low++] = L.r[high];
            while (low < high&&L.r[low].key <= pivotkey)
                low++;
            if (low < high)
                L.r[high--] = L.r[low];
        }
        L.r[low] = p;
        QuickSort(L, pl, low - 1);//低子表进行递归
        QuickSort(L, low + 1, ph);//高子表进行递归
    }
}
int main()
{
    int n;
    SqList L;
    cin >> n;
    L.length = n;
    for (int i = 1; i <= L.length; i++)
        cin >> L.r[i].key;
    QuickSort(L, 1, L.length);
    for (int i = 1; i <= L.length; i++)
        cout << L.r[i].key << " ";
    return 0;
}
```

选择排序:

```
#include<iostream>
using namespace std;
#define MAX_SIZE 100005
typedef int KeyType;
typedef struct {
    KeyType key;
}RedType;
typedef struct {
    RedType r[MAX_SIZE + 1];
    int length;
}SqList;

void SelectSort(SqList &L)
{
    int k;
    for (int i = 1; i < L.length; i++)
    {
        k = i;
        for (int j = i + 1; j <= L.length; j++)//选出剩余元素中最小值
```

```
            if (L.r[j].key < L.r[k].key)
                k = j;
        if (k != i)//将第i个与剩余元素中的最小值交换
        {
            RedType temp = L.r[k];
            L.r[k] = L.r[i];
            L.r[i] = temp;
        }
    }
}
int main()
{
    int n;
    SqList L;
    cin >> n;
    L.length = n;
    for (int i = 1; i <= L.length; i++)
        cin >> L.r[i].key;
    SelectSort(L);
    for (int i = 1; i <= L.length; i++)
        cout << L.r[i].key << " ";
    return 0;
}
```

折半插入排序:

```
#include<iostream>
using namespace std;
#define MAX_SIZE 100005
typedef int KeyType;
typedef struct {
    KeyType key;
}RedType;
typedef struct {
    RedType r[MAX_SIZE + 1];
    int length;
}SqList;

void BInsertSort(SqList &L)
{
    int j;
    for (int i = 2; i <= L.length; i++)
    {
        L.r[0] = L.r[i];
        int low = 1;
        int high = i - 1;
        while (low <= high)
        {
            int m = (low + high) / 2;
            if (L.r[0].key < L.r[m].key)
                high = m - 1;
            else
                low = m + 1;
        }
        for (j = i - 1; j >= high + 1; j--)
            L.r[j + 1] = L.r[j];
        L.r[high + 1] = L.r[0];
    }
}
int main()
{
    int n;
    SqList L;
    cin >> n;
    L.length = n;
    for (int i = 1; i <= L.length; i++)
        cin >> L.r[i].key;
    BInsertSort(L);
    for (int i = 1; i <= L.length; i++)
        cout << L.r[i].key << " ";
    return 0;
}
```

插入排序:

```
#include<iostream>

using namespace std;

#define MAX_SIZE 100005

typedef int KeyType;

typedef struct {

        KeyType key;

}RedType;
```

```
typedef struct {
        RedType r[MAX_SIZE + 1];
        int length;
}SqList;

void InsertSort(SqList &L)
{
        int j;
        for (int i = 2; i <= L.length; i++)
        {
                if (L.r[i].key < L.r[i - 1].key)
                {
                        L.r[0] = L.r[i];
                        L.r[i] = L.r[i - 1];
                        for
                                (j = i - 2; L.r[0].key <
L.r[j].key; j--)
                                L.r[j + 1] = L.r[j];
                        L.r[j + 1] = L.r[0];
                }
        }
}
int main()
{
        int n;
        SqList L;
        cin >> n;
```

```
	L.length = n;
	for (int i = 1; i <= L.length; i++)
		cin >> L.r[i].key;
	InsertSort(L);
	for (int i = 1; i <= L.length; i++)
		cout << L.r[i].key << " ";
	return 0;
}
```

冒泡排序:

```
#include<iostream>
using namespace std;
#define MAX_SIZE 100005
typedef int KeyType;
typedef struct {
    KeyType key;
}RedType;
typedef struct {
    RedType r[MAX_SIZE + 1];
    int length;
}SqList;

void BubbleSort(SqList &L)
{
    for (int i = 1; i < L.length; i++)
        for (int j = 1; j <= L.length - i; j++)
        {
            if (L.r[j + 1].key < L.r[j].key)
            {
                RedType temp = L.r[j + 1];
                L.r[j + 1] = L.r[j];
                L.r[j] = temp;
            }
        }
}
int main()
{
    int n;
    SqList L;
    cin >> n;
    L.length = n;
    for (int i = 1; i <= L.length; i++)
        cin >> L.r[i].key;
    BubbleSort(L);
    for (int i = 1; i <= L.length; i++)
        cout << L.r[i].key << " ";
    return 0;
}
```