数据结构导论

线性表

顺序表的结构定义

```
const int Maxsize = 100;    //预先定义一个足够大的常数
typedef struct
{
    DataType data[Maxsize];     //存放数据的数组
    int length;                 //顺序表的实际长度
} SeqList;      //顺序表类型名为SeqList
SeqList L;  //定义L为一个顺序表
```

单链表的类型定义

```
typedef struct node  //struct node 表示链表的结点，结点包含两个域：数据域（data）和指针域（next）
{
    DataType data;  //数据域
    struct node * next; //指针域
} Node, *LinkList;
```

循环链表 ：在单链表中，如果让最后一个结点的指针域指向第一个结点可以构成循环链表。在循环链表中，从任一结点出发能够扫描整个链表。

双向循环链表

```
struct dbnode
{
    DataType data;
    struct dbnode *prior, *next;
}
typedef struct dbnode *dbpointer;
typedef struct DLinkList;
```

栈、队列、数组

顺序栈

```
const int maxsize = 6;  //顺序栈的容量
typedef struct seqstack
{
    DataType data[maxsize];    //存储栈中数据元素的数组
    int top;                   //标志栈顶位置的变量
} SeqStk;
```

链栈

```
typedef struct node
{
    DataType data;
    struct node *next;
} LkStk;
```

顺序队列

```
const int maszize = 20;
typedef struct seqqueue
{
    DataType data[maxsize];
    int front, rear;
}SeqQue;
SeqQue SQ;
```

链接队列：队列的链接实现实际上是使用一个带有头结点的单链表来表示队列，称为链队列。头指针指向链表的头结点，单链表的头结点的 next 域指向队列首结点，尾指针指向队列尾结点，即单链表的最后一个结点。

```
typedef struct LinkQueueNode
{
    DataType data;
    struct LinkQueueNode *next;
} LkQueNode;

typedef struct LKQueue
{
    LKQueNode *front, *rear;
} LKQue;
LkQue LQ;
```

排序

排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	In-place	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
希尔排序	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(1)$	In-place	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Out-place	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	In-place	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	In-place	不稳定
计数排序	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	Out-place	稳定
桶排序	$O(n + k)$	$O(n + k)$	$O(n^2)$	$O(n + k)$	Out-place	稳定
基数排序	$O(n \times k)$	$O(n \times k)$	$O(n \times k)$	$O(n + k)$	Out-place	稳定