

排序

代码题 1

题干

- 题目：
 - 1、直接插入排序
- 总结：
 - 核心思想：双指针，i 指针负责大循环，从头往后走；j 指针负责小循环，在 i 的前一个位置开始、一直往前走，直到找到某个元素其大小比 `R[i]` 还要小，把元素插到这个元素的前面；

解答：基于顺序表

```
// R为数据序列，i为个数
void Insert(int R[],int n)
{
    // 因为当前使用了哨兵，因此R[0]位置是临时存放的当前用于比较的元素，所以数据序列是
    R[1]~R[n]
    for(int i = 2;i<=n;++i)
    {
        R[0] = R[i];
        // j 从 i-1 开始，因为 i 指向的元素已经放到了R[0]
        for(int j = i-1;R[0]<R[j];--j)
        {
            // 把元素往后移
            R[j+1] = R[j];
        }

        // 跳出循环时，当前j指向的元素其值要小于R[0]
        R[j+1] = R[0];
    }
}
```

解答：基于链表

```
void LinkSort(LNode* &L)
{
    LNode* pre,p,r;
    pre = L;
    p = L->next;

    // r到达了第二个结点
    r = p->next;

    // 将第一个结点的next和后背序列断开，此时前面的为有序子序列，r之后的为待排子序列
```

```

p->next = null;
p = r;

while(p!=null)
{
    r = p->next;
    pre = L;

    while(pre->next->data<p->data&&pre->next!=null)
    {
        pre = pre->next;
    }

    // 从头开始走，一直往后走到比当前p结点的值还大的结点
    p->next = pre->next;
    pre->next = p;
    p = r;
}
}

```

代码题 2

题干

- 题目：

2、折半插入排序

- 总结：

解答

```

// A[0]为哨兵
void BinInsert(int A[],int n)
{
    for(int cur = 2;cur<=n;cur++)
    {
        int low = 1;
        int high = cur -1;
        A[0] = A[cur];

        while(low<high)
        {
            // 折半查找到目标位置
            int mid = (low+high)/2;
            if(A[0]<A[mid])
            {
                high = mid-1;
            }
            else

```

```

        {
            low = mid+1;
        }

        // 将目标位置后的元素，全部后移一位
        for(j = i-1; j>=high+1; j--)
        {
            A[j+1] = A[j];
        }

        // 全部移动之后，将元素放入目标位置
        A[high+1] = A[0];
    }
}

```

代码题 4

题干

- 题目：
 - **4、冒泡排序**
- 总结：
 - 双指针，外层指针 i 负责从右走到左， i 指针右边的序列表示已经有序， i 指针左边的序列无序；
 - j 指针负责每一轮从左边无序的数字中，通过交换、将最大的元素移动至 i 的位置；

解答

```

void Sort(int R[],int n)
{
    bool flag = false;
    int temp;

    for(int i =n-1;i>=1;i--)
    {
        flag = false;

        // j从index=1而不是0开始，因为要将 R[j] 和 R[j-1] 进行比较
        for(int j = 1;j<=i;++j)
        {
            if(R[j]<R[j-1])
            {
                temp = R[j];
                R[j] = R[j-1];
                R[j-1] = temp;
                flag = true;
            }
        }
    }
}

```

```

        // flag为真表示这一轮遍历未移动任何元素，因此表示左序列已经有序
        if(flag == false)
        {
            return;
        }
    }
}
}

```

代码题 5

题干

- 题目：
 - **5、快速排序**
- 总结：
 -

解答

```

int part(int A[],int low,int high)
{
    // 默认A[low]是pivot
    int pivot = A[low];
    while(low<high)
    {
        // 如果当前A[high]比pivot大，则high继续往左走
        while(A[high]>pivot&&low<high)
        {
            high--;
        }
        // 当A[high]<pivot时，跳出循环
        A[low] = A[high];

        while(A[low]<pivot&&low<high)
        {
            low++;
        }
        A[high] = A[low];
    }

    A[low] = pivot;
    // 当跳出循环时，此时 low = high
    return low;
}

// n为元素长度
void QuickSort(int A[],int low,int high)

```

```

{
    if(low<high)
    {
        // 进行一趟快排，使得pivotPos左边都比A[pivotPos]小，右边都比它大
        int pivotPos = part(A[],low,high);
        QuickSort(A,low,pivotPos-1);
        QuickSort(A,pivotPos+1,high);
    }
}

```

代码题 6

题干

- 题目：
 - **6、选择排序**
- 总结：
 - 双指针和一个临时变量；
 - 指针 i 负责外循环，从左指向右，i 左边的元素都有序；
 - 指针 j 负责内循环，从 i 开始往右，对右序列（无序）遍历一次，使用 `curMin` 来记录当前右序列遍历中、最小的元素位置；
 - 完成一次内循环后，当前 `R[curMin]` 指向的是最小元素，将其放到 `R[i]` 所处位置；

解答

```

void SelectSort(int R[],int n)
{
    int curMin;

    for(int i = 0;i<n;i++)
    {
        curMin = i;
        for(int j = i+1;j<n;j++)
        {
            if(R[j]<R[curMin])
            {
                curMin = j;
            }
        }
        int temp = R[i];
        R[i] = R[curMin];
        R[curMin] = temp;
    }
}

```

代码题 7

题干

- 题目：

- ## 7、堆排序

- 总结：

- 注意：因为是构建升序序列，因此是使用大根堆（每次取出最大的元素，放到序列末尾）；

解答

```
// 进行一次堆修复，以low为根，范围是low至high的树
void HeapFix(int arr[],int low,int high)
{
    // 待修复的根
    int i = low;

    // 待修复的根的左子树
    int j = i*2+1;

    // 暂存当前根元素，因为如果根比左、右小，则根的值会被左或右子树的值覆盖，因此需要暂存
    int temp = arr[i];

    while(j<=high)
    {
        // 要在 左、根、右 中选择最大的元素，需要先对左、右两棵子树进行比较，再用较大的子树
        // 去和根比较
        if(arr[j]<arr[j+1]&& j<high)
        {
            j++;
        }

        // 如果子树比根大
        if(arr[j]>arr[i])
        {
            // 把大子树的值放到根
            arr[i] = arr[j];

            // 往下再一次进行修复，将i设置为当前j的位置，并以此位置、进行修复
            i = j;
            j = i*2+1;
        }
        else
        {
            break;
        }
    }
    arr[i] = temp;
}

// n为堆长度，元素分布在 arr[0] ~ arr[n-1] 中
void HeapSort(int arr[],int n)
```

```

{
    // 第一步：构建初始堆
    // n/2-1 是当前最右下角的分支
    for(int i = n/2-1 ; i>=0;i++)
    {
        HeapFix(arr,i,n-1);
    }

    // 第二步：堆排序，每次从堆中取出最大元素、放入当前堆尾序列中
    for(int i = n-1; i>0;i--)
    {
        int temp = a[0];
        a[0] = a[i];

        // 把当前堆顶放到最后
        a[i] = temp;

        // 因为当前最右下角的元素放到了堆顶，因此要修复堆
        HeapFix(arr,0,i-1);
    }
}

```

代码题 8

题干

- 题目：

• 8、归并排序

- 总结：

-

解答

```

void Sort(int A[],int low,int mid,int high)
{
    // 使用B来暂存
    int* B = new B[maxsize];
    int i = low;
    int j = mid+1;

    for(int k = low;k<=high;k++)
    {
        B[k] = A[k];
    }

    for(int k = i;i<=mid&&j<=high;k++)
    {
        if(B[i]<B[j])
        {

```

```

        A[k] = B[i];
        i++;
    }
    else
    {
        A[k] = B[j];
        j++;
    }
}

while(i<=mid)
{
    A[k] = A[i];
    i++;
    k++;
}

while(j<=high)
{
    A[k] = A[j];
    j++;
    k++;
}
}

void MergeSort(int A[],int low,int high)
{
    if(low<high)
    {
        int mid = (high+low)/2;
        MergeSort(A,low,mid);
        MergeSort(A,mid+1,high);
        Sort(A,low,mid,high);
    }
}

```

获取