排序算法总结

插入排序

1、直接插入排序

```
//思路：从第二个元素开始一直插入，不断找到要插入的位置并整体后移，然后插入
void InsertSort(int a[], int n){
    int i,j;
    for(i = 1; i < n; i++){
        int temp = a[i];          //需要插入的元素
        j = i-1;                  //开始比较的位置
        while(j>=0 && a[j] > temp){
            a[j+1] = a[j];        //整体后移
            j--;
        }
        a[j+1] = temp;            //插入
    }
}
//空间复杂度 O(1) 最好时间复杂度（全部有序） O(n) 最坏时间复杂度（全部逆序） O(n^2) 平均复杂度 O(n^2)  稳定的排序算法
```

2、折半插入排序

```
//思路：整体思路，还是插入排序，但是在查找插入位置中改进算法，使用折半查找改进效率
void InsertMidSort(int a[], int n){
    int i,j,low,high;
    for(int i = 1; i < n; i++){
        int temp = a[i];
        low =  0; high = i-1;         //设置查找范围
        while(low <= high){
            int mid = (high+low)/2;
            if(a[mid] > temp) high = mid - 1;
            else low = mid + 1;
        }                             //找到插入的为high+1
        for(j=i-1; j >= high+1; j--){
            a[j+1] = a[j];            //整体后移为插入的位置留出空位
        }
        a[high+1] = temp;             //插入
    }
}
//最坏时间复杂度为O(n^2)，平均时间复杂度O(n^2)，最好时间复杂度O(nlogn)，空间O(1)，也是一种稳定的排序算法
```

3、希尔排序

```
//思路：按固定步长分组然后分别采用直接插入排序
void ShellSort(int a[], int n){
    for(int d = n/2; d >= 1; d/=2){          //步长
        for(int f=0; f < d; f++){            //每组的第一个下标
            for(int i = f+d; i < n; i+=d){   //步长为d的直接插入排序
```

```
                int temp = a[i];
                int j = i-d;
                while(j >= 0 && a[j] > temp){
                    a[j+d] = a[j];
                    j = j-d;
                }
                a[j+d] = temp;
            }
        }
    }
}
//最坏时间复杂度为O(n^2)，平均时间复杂度O(n^1.3)，最好时间复杂度O(n)，空间O(1)，不稳定的排序算法
```

交换排序

1、冒泡排序

```
//思路：每次将最小的元素放到数组尾部
void BubbleSort(int a[], int n){
    for(int i = n-1; i >= 1; i--){
        int flag = 0;
        for(int j = 0; j < i; j++){
            if(a[j] > a[j+1]){          //大于才交换，  保证稳定性，升序排序
                int temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
                flag = 1;
            }
        }
        if(flag == 0) return;           //该趟未排序，已经有序
    }
}
//最坏时间复杂度为O(n^2)，平均时间复杂度O(n^2)，最好时间复杂度O(n)，空间O(1)，稳定的排序算法
```

2、快速排序

比较难懂，建议直接背

```
//一次划分
//思路：每次取low下标的值，对数组进行左右划分
int Partition(int a[], int low, int high){
    int i = low;
    int j = high;
    int key = a[i];
    j++;                            //不同之处
    while(i < j){
        i++;                        //特殊之处，先比较小的
        while(a[i] <= key) i++;
        j--;                        //先减减，然后再判断
        while(a[j] > key) j--;
        if(i < j){                  //交换
            int temp = a[i];
            a[i] = a[j];
```

```
            a[j] = temp;
        }
    }                                   //该趟排序结束
    int temp = a[j];                    //基准元素与当前下标元素交换
    a[j] = a[low];
    a[low] = temp;
    return j;
}

//快速排序
//递归
void QuickSort(int a[], int low, int high){
    if(low < high){
        int mid = Partition(a,low,high);
        QuickSort(a,low,mid-1);
        QuickSort(a,mid+1,high);
    }
}
//最坏时间复杂度为O(n^2)，平均时间复杂度O(nlogn)，最好时间复杂度O(nlogn)，空间O(1)，不稳定的排序算法
```

选择排序

1、直接选择排序

```
//思路：每次选择最小的元素
void SelectSort(int a[], int n){
    for(int i = 0; i < n-1; i++){
        int min = i;
        for(int j = i+1; j < n; j++){
            if(a[j] < a[min]){
                min = j;                //找到当前最小
            }
        }
        if(min != i){                   //替换
            int temp = a[i];
            a[i] = a[min];
            a[min] = temp;
        }
    }
}
//最坏时间复杂度为O(n^2)，平均时间复杂度O(n^2)，最好时间复杂度O(n^2)，空间O(1)，不稳定的排序算法
```

2、堆排序

```
//堆调整的算法
void HeadAdjust(int *a, int k, int n){
    int i,temp;
    temp = a[k];                        //记录当前根节点的值，以便后序进行赋值
    for(i = 2*k; i < n; i=i*2){
        if(i < n && a[i+1] > a[i]){     //i表示较大的那个节点
            i++;
```

```
        }
        if(temp >= a[i]) break;          //根节点最大不用调整
        else{
            a[k] = a[i];                 //根节点赋值为子节点值较大的那个
            k = i;                       //修改k值，方便后序进行调整
        }
    }
    a[k] = temp;                         //调整完，根节点的值，存放在当前空着的子节点中
}

//建立堆，调整n/2次，建立的一个最大堆
void BuildMaxHeap(int *a, int n){
    for(int i = n/2; i>=0;i--){          //从[n/2]~0,反复调整堆
        HeadAdjust(a,i,n);
    }
}

//实现堆排序
void HeapSort(int *a, int n){
    BuildMaxHeap(a,n);                   //这里只是建立了一个大根堆，不一定有序，所以还得继续调整
    //每次取堆顶元素，将其置于数组最后一位，然后下次循环不需要在考虑最后一位，堆的调整不用考虑已经排好序
    for(int i = n-1; i > 0;i--){
        int temp = a[i];
        a[i] = a[0];
        a[0] = temp;
        HeadAdjust(a,0,i-1);
    }
}
//最坏时间复杂度为O(nlogn)，平均时间复杂度O(nlogn)，最好时间复杂度O(nlogn)，空间O(1)，稳定的排序算法
```

归并排序

```
//归并排序
void Merge(int a[], int low, int mid, int high){//将两段有序的有序表合成一个新的有序表
    int i,j,k;
    int b[maxsize];
    for(i = low; i <= high; i++){                //辅助数组b存储a当前所有值
        b[i] = a[i];
    }
    for(i=low,j=mid+1,k=i; i<=mid &&j<=high; k++){
        if(b[i] <= b[j]){
            a[k] = b[i++];
        }
        else{
            a[k] = b[j++];
        }
    }
    while(i<=mid) a[k++] = b[i++];
    while(j<=high) a[k++] = b[j++];
}

void MergeSort(int a[], int low, int high){
```

```
    if(low < high){
        int mid = (low+high)/2;
        MergeSort(a,low,mid);          //对左侧子序列进行划分
        MergeSort(a,mid+1,high);       //对右侧子序列进行划分
        Merge(a,low,mid,high);         //归并
    }
}
//最坏时间复杂度为O(nlogn)，平均时间复杂度O(nlogn)，最好时间复杂度O(nlogn)，空间O(n)，稳定的排序算法
```

排序算法性能总结：

排序方法	平均情况	最好情况	最坏情况	空间	稳定性
冒泡	$O(n^2)$	O(n)	$O(n^2)$	O(1)	稳定
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定
直接插入排序	$O(n^2)$	O(n)	$O(n^2)$	O(1)	稳定
希尔排序	O(nlogn) ~ $O(n^2)$	$O(n^{1.3})$	$O(n^2)$	O(1)	不稳定
堆排序	O(nlogn)	O(nlogn)	O(nlogn)	O(1)	不稳定
归并排序	O(nlogn)	O(nlogn)	O(nlogn)	O(n)	稳定
快速排序	O(nlogn)	O(nlogn)	$O(n^2)$	O(logn)~O(n)	不稳定

- **稳定排序算法：直接插入排序、冒泡排序、归并排序、基数排序**
- **时间复杂度较好的排序算法：快速排序、归并排序、堆排序、基数排序**
    - 快速排序，当元素随机分布，排序效果最好，算法性能最优，当元素基本有序，时间复杂度和空间复杂度都会达到最大
- **当元素基本有序时，可以选择直接插入排序和冒泡排序**