排序篇

- 内部排序
  - 插入排序
    - 直接插入排序
    - 折半插入排序
    - 希尔排序
  - 交换排序
    - 冒泡排序
    - 快速排序
  - 选择排序
    - 简单选择排序
    - 堆排序
  - 归并排序
  - 基数排序
- 外部排序

排序前的准备：

```
typedef int ElemType;   // 定义关键字类型
#define MAXSIZE 100000  // 定义序列大小
// 交换函数
inline void swap(ElemType *a, ElemType *b) {
    ElemType temp = *a;
    *a = *b;
    *b = temp;
}
```

一、插入排序

1. 直接插入排序

```
void InsertSort(ElemType a[], int len) {
    int i, j;
    ElemType temp;
    for (i = 1; i < len; ++i) {  // a[0]为有序序列，所以i从1开始
        if (a[i - 1] > a[i]) {
            temp = a[i];
            for (j = i - 1; a[j] > temp && j >= 0; --j) {
                a[j + 1] = a[j];
            }
            a[j + 1] = temp;
        }
    }
}
```

2. 折半插入排序

```
void BInsertSort(ElemType a[], int len) {
    int i, j, low, high, mid;
    ElemType temp;
    for (i = 1; i < len; ++i) {
        if (a[i - 1] > a[i]) {
            temp = a[i];
            low = 0;
            high = i - 1;
            while (low <= high) {  // 折半查找
                mid = (high + low) / 2;
                if (a[mid] > temp) {
                    high = mid - 1;
                } else {
                    low = mid + 1;
                }
            }
            for (j = i - 1; j >= low; --j) {
                a[j + 1] = a[j];
            }
            a[j + 1] = temp;
        }
    }
}
```

3. 希尔排序

```
int Incre[10] = {1023, 511, 255, 127, 63, 31, 15, 7, 3, 1};
void ShellSort(ElemType a[], int len, int incre[], int in_len) {
    int i, j, k, increment;
    ElemType temp;
    for (i = 0; i < in_len; ++i) {
        increment = incre[i];
```

```
        for (j = increment; j < len; ++j) {
            if (a[j] < a[j - increment]) {
                temp = a[j];
                for (k = j - increment; a[k] > temp && k >= 0; k -= increment) {
                    a[k + increment] = a[k];
                }
                a[k + increment] = temp;
            }
        }
    }
}
```

二、交换排序

1. 冒泡排序

```
void BubbleSort(ElemType a[], int len) {
    bool flag = true;  // 用于标记，若flag为假，则顺序表已经全部有序，无需进行之后的排序操作
    for (int i = 0; i < len - 1 && flag == true; ++i) {
        flag = false;
        for (int j = len - 1; j > i; --j) {
            if (a[j - 1] > a[j]) {
                swap(&a[j - 1], &a[j]);
                flag = true;
            }
        }
    }
}
```

2. 快速排序

```
void QSort(ElemType a[], int low, int high) {
    ElemType pivot = a[low];  // 取首元素为基准
    int left = low, right = high;
    if (low < high) {
        while (1) {  // 序列中比基准小的移到左边，大的移到右边
            while ((low < high) && (pivot <= a[high])) --high;
            while ((low < high) && (pivot >= a[low])) ++low;
            if (low < high) {
                swap(&a[low], &a[high]);
            } else {
                break;
            }
        }
        swap(&a[high], &a[left]);
        QSort(a, left, high - 1);
```

```
        QSort(a, high + 1, right);
    }
}
void QuickSort(ElemType a[], int len) {
    // 调用快排
    QSort(a, 0, len - 1);
}
```

三、选择排序

1. 简单选择排序

```
void SelectSort(ElemType a[], int len) {
    for (int i = 0, min = 0; i < len - 1; ++i) {
        min = i;
        for (int j = i + 1; j < len; ++j) {
            if (a[min] > a[j]) min = j;
        }
        if (min != i) swap(&a[i], &a[min]);
    }
}
```

2. 堆排序

```
void AdjustDown(ElemType a[], int i, int len) {
    // 从第i个元素开始进行向下调整
    int child;
    ElemType temp;
    for (temp = a[i]; 2 * i + 1 < len; i = child) {
        child = 2 * i + 1;  // 左孩子结点，因为数组下标从0开始
        if (child != len - 1 && a[child + 1] > a[child]) ++child;
        if (temp < a[child]) {
            a[i] = a[child];
        } else {
            break;
        }
    }
    a[i] = temp;
}
void HeapSort(ElemType a[], int len) {
    // 建立最大堆
    for (int i = len / 2; i >= 0; --i) {
        AdjustDown(a, i, len);
    }
    for (int i = len - 1; i > 0; --i) {
        swap(&a[0], &a[i]);
        AdjustDown(a, 0, i);
    }
```

```
}
```

四、归并排序

```
void Merge(ElemType a[], ElemType temp_a[], int left, int mid, int right)
{
    // 将有序的a[left]~a[mid]和a[mid+1]~a[right]归并成一个有序序列
    int i, j, k;
    for (i = left; i <= right; ++i) temp_a[i] = a[i];
    for (i = left, j = mid + 1, k = i; i <= mid && j <= right; ++k) {
        if (temp_a[i] <= temp_a[j]) {
            a[k] = temp_a[i++];
        } else {
            a[k] = temp_a[j++];
        }
    }
    while (i <= mid) a[k++] = temp_a[i++];
    while (j <= right) a[k++] = a[j++];
}
void MSort(ElemType a[], ElemType temp_a[], int left, int right) {
    // 递归地将a[left]~a[right]排序
    if (left < right) {
        MSort(a, temp_a, left, (left + right) / 2);
        MSort(a, temp_a, (left + right) / 2 + 1, right);
        Merge(a, temp_a, left, (left + right) / 2, right);
    }
}
void MergeSort(ElemType a[], int len) {
    // 归并排序
    ElemType *temp_a;
    temp_a = (ElemType *)malloc(len * sizeof(ElemType));
    MSort(a, temp_a, 0, len - 1);
    free(temp_a);
}
```

五、内部排序算法比较

算法种类	T(best)	T(avg)	T(worst)	Sn	是否稳定
直接插入排序	O(n)	O(n^2)	O(n^2)	O(1)	是
冒泡排序	O(n)	O(n^2)	O(n^2)	O(1)	是
简单选择排序	O(n^2)	O(n^2)	O(n^2)	O(1)	否
希尔排序	-	-	-	O(1)	否
快速排序	O(nlogn)	O(nlogn)	O(n^2)	O(logn)	否
堆排序	O(nlogn)	O(nlogn)	O(nlogn)	O(1)	否
2路归并排序	O(nlogn)	O(nlogn)	O(nlogn)	O(n)	是

六、算法测试

```
clock_t Start, End; // 包含头文件time.h
int main() {
    srand(MAXSIZE);
    ElemType a[MAXSIZE];
    cout << "原始数据: ";
    for (int i = 0; i < MAXSIZE; ++i) {
        a[i] = rand() % MAXSIZE;
        // a[i] = MAXSIZE - i;
        cout << a[i] << " ";
    }
    Start = clock();
    // SelectSort(a, MAXSIZE);
    // HeapSort(a, MAXSIZE);
    // InsertSort(a, MAXSIZE);
    // BInsertSort(a, MAXSIZE);
    // ShellSort(a, MAXSIZE, Incre, 10);
    // BubbleSort(a, MAXSIZE);
    // QuickSort(a, MAXSIZE);
    // MergeSort(a, MAXSIZE);
    // sort(a, a + MAXSIZE);
    End = clock();
    cout << endl << "排序时间为: " << (double)(End - Start) / CLOCKS_PER_SEC << "秒" << endl << "排序后数据: ";
    for (int i = 0; i < MAXSIZE; ++i) {
        cout << a[i] << " ";
    }
}
```