六种排序算法

选择排序

```
def Select_Sort(array_in):
    """选择排序-从前往后遍历选择出最小数据"""
    length = len(array_in)
    if length<=1:
        return array_in
    for i in range(len(array_in)-1):
        small=i
        for j in range(i+1,len(array_in)):
            if array_in[small]>array_in[j]:
                small=j
        if small!=i:
            temp=array_in[i]
            array_in[i]=array_in[small]
            array_in[small]=temp
    return array_in
```

冒泡排序

```
def Bubble_Sort(array_in):
    """冒泡排序-依次将最大数据吹到后面"""
    length=len(array_in)
    if length<=1:
        return array_in
    while(length>0):
        for i in range(length-1):
            if array_in[i]>array_in[i+1]:
                array_in[i]=array_in[i]+array_in[i+1]
                array_in[i+1]=array_in[i]-array_in[i+1]
                array_in[i]=array_in[i]-array_in[i+1]
        length-=1
    return array_in
```

插入排序

```
def Insert_Sort(arr):
    """插入排序-从后往前将元素插入到合适的位置"""
    length=len(arr)
    if length<=1:
        return arr
    for i in range(1,length):
        for j in range(i,0,-1):  #从后往前遍历
            if arr[j]<arr[j-1]:
                arr[j],arr[j-1]=arr[j-1],arr[j]
            else:
                break
    return arr
```

快速排序

```
# 快速排序
def swap(array, i, j):
    """交换"""
    temp = array[i]
    array[i] = array[j]
    array[j] = temp


def Partition(array, left, right):
    """实现划分操作"""
    pivot = left  # 基准值
    index = pivot + 1
    i = index
    while i <= right:
        if array[i] < array[pivot]:
            swap(array, i, index)
            index += 1
        i += 1
    swap(array, pivot, index - 1)
    return index - 1


def Quick_Sort(array, left=None, right=None):
    """快速排序算法的最终实现，递归实现"""
    # 初始化left、right isinstance方法主要是针对None情况
    left = 0 if not isinstance(left, int) else left
    right = len(array) - 1 if not isinstance(right, int) else right
    if left < right:
        PartitionIndex = Partition(array, left, right)  # 划分
        Quick_Sort(array, left, PartitionIndex - 1)
        Quick_Sort(array, PartitionIndex + 1, right)
    return array
```

堆排序

```
# 堆排序
def heapify(array, i):
```

```
    "堆调整算法"
    left = 2 * i + 1
    right = 2 * i + 2
    largest = i
    if left < arrayLen and array[left] > array[largest]:
        largest = left
    if right < arrayLen and array[right] > array[largest]:
        largest = right
    if largest != i:
        swap(array, i, largest)
        heapify(array, largest)


def buildMaxHeap(array):
    """构造大根堆"""
    # math.floor相当于向下取整
    for i in range(math.floor(len(array) / 2), -1, -1):
        heapify(array, i)


def Heap_Sort(array):
    global arrayLen
    arrayLen = len(array)
    buildMaxHeap(array)
    for i in range(len(array) - 1, 0, -1):
        swap(array, 0, i)
        arrayLen -= 1
        heapify(array, 0)
    return array
```

归并排序

```
# 归并排序
def merge(left, right):
    """归并操作"""
    result = []
    while left and right:  # 当两个子集都有元素时，比较它们顶部的两个元素
        if left[0] <= right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))
    while left:
        result.append(left.pop(0))
    while right:
        result.append(right.pop(0))
    return result


def Merge_Sort(array):
    """归并排序"""
    if len(array) < 2:
        return array
    middle = math.floor(len(array) / 2)
```

```
    # 分解区间
    leftArray = array[0:middle]
    rightArray = array[middle:]
    return merge(Merge_Sort(leftArray), Merge_Sort(rightArray))  # 递归
```