1.插入排序

它的基本思想是将一个记录插入到已经排好序的有序表中，从而一个新的、记录数增 1 的有序表。
在其实现过程使用双层循环，
外层循环对除了第一个元素之外的所有元素，
内层循环对当前元素前面有序表进行待插入位置查找，并进行移动。

基本思想：插入排序的工作原理是通过构建有序序列，在已排序序列中从后向前扫描，找到相应位置并插入。
插入排序在实现上，通常采用in-place排序（即只需用到O(1)的额外空间的排序），
因而在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间。

```
def insertion_sort(nums):
    """
    ① 从第一个元素开始，该元素可以认为已经被排序
    ② 取出下一个元素，在已经排序的元素序列中从后向前扫描
    ③如果该元素（已排序）大于新元素，将该元素移到下一位置
    ④ 重复步骤③，直到找到已排序的元素小于或者等于新元素的位置
    ⑤将新元素插入到该位置后
    ⑥ 重复步骤②~⑤
    :param nums:
    :return:
    """
    for i in range(1, len(nums)):
        cur = i
        for j in range(i-1, -1, -1):
            if nums[cur] < nums[j]:
                nums[cur], nums[j] = nums[j], nums[cur]
                cur -= 1
    return nums


print(insertion_sort([5, 4, 6, 3, 7, 2, 8, 1, 9]))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- 平均时间复杂度：O(N^2)
- 最差时间复杂度：O(N^2)
- 空间复杂度：O(1)
- 排序方式：In-place
- 稳定性：稳定

2.希尔排序

希尔排序的实质就是分组插入排序
先将整个待排元素序列分割成若干个子序列（由相隔某个“增量”的元素组成的）分别进行直接插入排序，
然后依次缩减增量再进行排序，待整个序列中的元素基本有序（增量足够小）时，再对全体元素进行一次直接插入排序。
因为直接插入排序在元素基本有序的情况下（接近最好情况），效率是很高的，因此希尔排序在时间效率上比前两种方法有较大提高。

```
def shell_sort(nums):
    """
    ① 先取一个小于n的整数step作为第一个增量，把文件的全部记录分成step个组。
    ② 所有距离为step的倍数的记录放在同一个组中，在各组内进行直接插入排序。
    ③ 取第二个增量d2小于d1重复上述的分组和排序，直至所取的增量dt=1(dt小于dt-1小于…小于d2小于d1),
    即所有记录放在同一组中进行直接插入排序为止。
    :param nums:
    :return:
    """
    step = len(nums) // 2
    while step >= 1:
        for i in range(step, len(nums)):
            cur = i
            while nums[cur - step] > nums[cur] and cur - step >= 0:
                nums[cur], nums[cur - step] = nums[cur - step], nums[cur]
                cur -= 1
        step //= 2
    return nums


print(shell_sort([4, 5, 3, 6, 7, 2, 1, 8]))
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

3.冒泡排序

```
def bubble_sort(nums):
    for i in range(len(nums)):
        for j in range(i, len(nums)):
            if nums[i] > nums[j]:
                nums[i], nums[j] = nums[j], nums[i]
    return nums


print(bubble_sort([4, 5, 3, 6, 2, 7, 1, 8]))
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

4.选择排序

首先在未排序序列中找到最小元素，存放到排序序列的起始位置，
然后再从剩余未排序元素中继续寻找最小元素，然后放到已排序序列的末尾。

```
def selection_sort(nums):
    for i in range(len(nums)):
        min_index = i
        for j in range(i + 1, len(nums)):
            if nums[min_index] > nums[j]:
                min_index = j
        nums[i], nums[min_index] = nums[min_index], nums[i]
    return nums


print(selection_sort(nums=[5, 4, 6, 3, 7, 2, 8, 1, 9]))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

5.快速排序

快速排序通过分治法和递归原理来实现，分治法的主要作用是把一个序列分成两个序列，
等递归到底部时，数列的大小是1，也就排序好了

```
import random

def quick_sort(nums):
    """
    ① 从数列中挑出一个元素，称为 “基准”（pivot），
    ② 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。
        在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作。
    ③ 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。
    :param nums:
    :return:
    """
    if len(nums) <= 1:
        return nums
    index = random.randint(0, len(nums) - 1)
    value = nums.pop(index)
    left = [i for i in nums if i < value]
    right = [i for i in nums if i >= value]
    return quick_sort(left) + [value] + quick_sort(right)


print(quick_sort([4, 5, 6, 3, 7, 7, 8, 1, 9]))
```

```
[1, 3, 4, 5, 6, 7, 7, 8, 9]
```

6.归并排序

归并排序是用分治思想，分治模式在每一层递归上有三个步骤：

- 分解（Divide）：将n个元素分成个含n//2个元素的子序列。
- 解决（Conquer）：用合并排序法对两个子序列递归的排序。
- 合并（Combine）：合并两个已排序的子序列已得到排序结果。

```
def merge(left, right):
    res = []
    while left and right:
        if left[0] < right[0]:
            res.append(left.pop(0))
        else:
            res.append(right.pop(0))
    res += left
    res += right
    return res


def merge_sort(nums):
    if len(nums) <= 1:
        return nums
    mid = len(nums) // 2
    left = merge_sort(nums[:mid])
    right = merge_sort(nums[mid:])
    return merge(left, right)


print(merge_sort([5, 4, 2, 1, 6, 9, 8, 7, 3]))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- 平均时间复杂度：O(nlogn)
- 最佳时间复杂度：O(n)
- 最差时间复杂度：O(nlogn)
- 空间复杂度：O(n)
- 排序方式：In-place
- 稳定性：稳定

```
```