排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	In-place	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
希尔排序	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(1)$	In-place	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Out-place	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	In-place	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	In-place	不稳定
计数排序	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	Out-place	稳定
桶排序	$O(n + k)$	$O(n + k)$	$O(n^2)$	$O(n + k)$	Out-place	稳定
基数排序	$O(n \times k)$	$O(n \times k)$	$O(n \times k)$	$O(n + k)$	Out-place	稳定

名称	数据对象	稳定性	时间复杂度		额外空间复杂度	描述
			平均	最坏		
冒泡排序	数组	✓	$O(n^2)$		$O(1)$	（无序区，有序区）。 从无序区透过交换找出最大元素放到有序区前端。
选择排序	数组	✗	$O(n^2)$		$O(1)$	（有序区，无序区）。 在无序区里找一个最小的元素跟在有序区的后面。对数组：比较得多，换得少。
	链表	✓				
插入排序	数组、链表	✓	$O(n^2)$		$O(1)$	（有序区，无序区）。 把无序区的第一个元素插入到有序区的合适的位置。对数组：比较得少，换得多。
堆排序	数组	✗	$O(n \log n)$		$O(1)$	（最大堆，有序区）。 从堆顶把根卸出来放在有序区之前，再恢复堆。
归并排序	数组	✓	$O(n \log^2 n)$		$O(1)$	把数据分为两段，从两段中逐个选最小的元素移入新数据段的末尾。 可从上到下或从下到上进行。
			$O(n \log n)$		$O(n) + O(\log n)$ 如果不是从下到上	
	链表				$O(1)$	
快速排序	数组	✗	$O(n \log n)$	$O(n^2)$	$O(\log n)$	（小数，基准元素，大数）。 在区间中随机挑选一个元素作基准，将小于基准的元素放在基准之前，大于基准的元素放在基准之后，再分别对小数区与大数区进行排序。
希尔排序	数组	✗	$O(n \log^2 n)$	$O(n^2)$	$O(1)$	每一轮按照事先决定的间隔进行插入排序，间隔会依次缩小，最后一次一定要是1。
计数排序	数组、链表	✓	$O(n + m)$		$O(n + m)$	统计小于等于该元素值的元素的个数i，于是该元素就放在目标数组的索引i位（i≥0）。
桶排序	数组、链表	✓	$O(n)$		$O(m)$	将值为i的元素放入i号桶，最后依次把桶里的元素倒出来。
基数排序	数组、链表	✓	$O(k \times n)$	$O(n^2)$		一种多关键字的排序算法，可用桶排序实现。

```python
# 冒泡排序O(n^2)
def bubbleSort(arr):
    for i in range(len(arr)-1, 0, -1):
        for j in range(0, i):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# 选择排序O(n^2)
def selectionSort(arr):
    for i in range(len(arr)-1):
        minIndex = i
        for j in range(i+1, len(arr)):
            if arr[j] < arr[minIndex]:
                minIndex = j
        if i != minIndex:
            arr[i], arr[minIndex] = arr[minIndex], arr[i]
    return arr
```

```python
# 插入排序O(n^2)
def insertionSort(arr):
    for i in range(1, len(arr)):
        cur = arr[i]
        pre = i - 1
        while pre >= 0 and cur < arr[pre]:
            arr[pre + 1] = arr[pre]
            pre -= 1
        arr[pre + 1] = cur
    return arr

# 希尔排序O(nlogn)
def shellSort(arr):
    gap = len(arr) // 2
    while gap > 0:
        for i in range(gap, len(arr)):
            cur = arr[i]
            pre = i - gap
            while pre >= 0 and cur < arr[pre]:
                cur, arr[pre] = arr[pre], cur
                pre -= gap
        gap //= 2
    return arr
```

```python
# 归并排序O(nlogn)
def mergeSort(arr):
    if len(arr) <= 1: return arr
    mid = len(arr) // 2
    left, right = arr[0: mid], arr[mid:]
    left = mergeSort(left)
    right = mergeSort(right)
    res = []
    while left and right:
        if left[0] <= right[0]:
            res.append(left.pop(0))
        else:
            res.append(right.pop(0))
    while left:
        res.append(left.pop(0))
    while right:
        res.append(right.pop(0))
    return res
```

```python
# 快速排序O(nlogn)
# start为起始索引，end为终止索引
def quickSort(arr, start, end):
    if start >= end: return
    left, right = start, end
    flag = arr[left]
    while left < right:
        while left < right and arr[right] > flag:
            right -= 1
        arr[left] = arr[right]
        while left < right and arr[left] <= flag:
            left += 1
        arr[right] = arr[left]
    arr[right] = flag
    quickSort(arr, start, left - 1)
    quickSort(arr, left + 1, end)
    return arr
```

```python
# 堆排序O(nlogn)
def heapSort(arr):
    def buildMaxHeap(arr):  # 建大顶堆
        for i in range(n // 2, -1, -1):
            heapify(arr, i)
    def heapify(arr, i):   # 始终保证节点值大于左右节点值
        largest = i
        left = 2*i+1
        right = 2*i+2
        if left < n and arr[largest] < arr[left]:
            largest = left
        if right < n and arr[largest] < arr[right]:
            largest = right
        if largest != i:
            arr[i], arr[largest] = arr[largest], arr[i]
            heapify(arr, largest)
    n = len(arr)
    buildMaxHeap(arr)
    # 将堆顶的最大值放至arr的最末尾，同时进行后续同样递归操作的arr长度减1
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        n -= 1
        heapify(arr, 0)
    return arr
```