一、归并排序

归并排序是采用分治法的一个非常典型的应用。归并排序的思想就是先递归分解数组，再合并数组。

将数组分解最小之后，然后合并两个有序数组，基本思路是比较两个数组的最前面的数，谁小就先取谁，取了后相应的指针就往后移一位。然后再比较，直至一个数组为空，最后把另一个数组的剩余部分复制过来即可。

1.归并排序实现

```
def merge_sort(alist):
    if len(alist) <= 1:
        return alist
    # 二分分解
    num = len(alist)//2
    left = merge_sort(alist[:num])
    right =merge_sort(alist[num:])
    # 合并
    return merge(left,right)

def merge(left, right):
    '''合并操作，将两个有序数组left[]和right[]合并成一个大的有序数组'''
    #left与right的下标指针
    l, r = 0, 0
    result = []
    while l<len(left) and r<len(right):
        if left[l] <= right[r]:
            result.append(left[l])
            l += 1
        else:
            result.append(right[r])
            r += 1
    result += left[l:]
    result += right[r:]
    return result

alist =[22,67,34,23,97,89,56,13,71]
sorted_alist =mergeSort(alist)
print(sorted_alist)
```

- 最优时间复杂度：O(nlogn)
- 最坏时间复杂度：O(nlogn)
- 稳定性：稳定

二、希尔排序

希尔排序(Shell Sort)是插入排序的一种。也称缩小增量排序，是直接插入排序算法的一种更高效的改进版本。希尔排序是非稳定排序算法。该方法因DL．Shell于1959年提出而得名。 希尔排序是把记录按下标的一定增量分组，对每组使用直接插入排序算法排序；随着增量逐渐减少，每组包含的关键词越来越多，当增量减至1时，整个文件恰被分成一组，算法便终止。

1.希尔排序过程

希尔排序的基本思想是：将数组列在一个表中并对列分别进行插入排序，重复这过程，不过每次用更长的列（步长更长了，列数更少了）来进行。最后整个表就只有一列了。将数组转换至表是为了更好地理解这算法，算法本身还是使用数组进行排序。

例如，假设有这样一组数[13 14 94 33 82 25 59 94 65 23 45 27 73 25 39 10]，如果我们以步长为5开始进行排序，我们可以通过将这列表放在有5列的表中来更好地描述算法，这样他们就应该看起来是这样(竖着的元素是步长组成)：

```
13 14 94 33 82
25 59 94 65 23
45 27 73 25 39
10
```

然后我们对每列进行排序：

```
10 14 73 25 23
13 27 94 33 39
25 59 94 65 82
45
```

将上述四行数字，依序接在一起时我们得到：[10 14 73 25 23 13 27 94 33 39 25 59 94 65 82 45]。这时10已经移至正确位置了，然后再以3为步长进行排序：

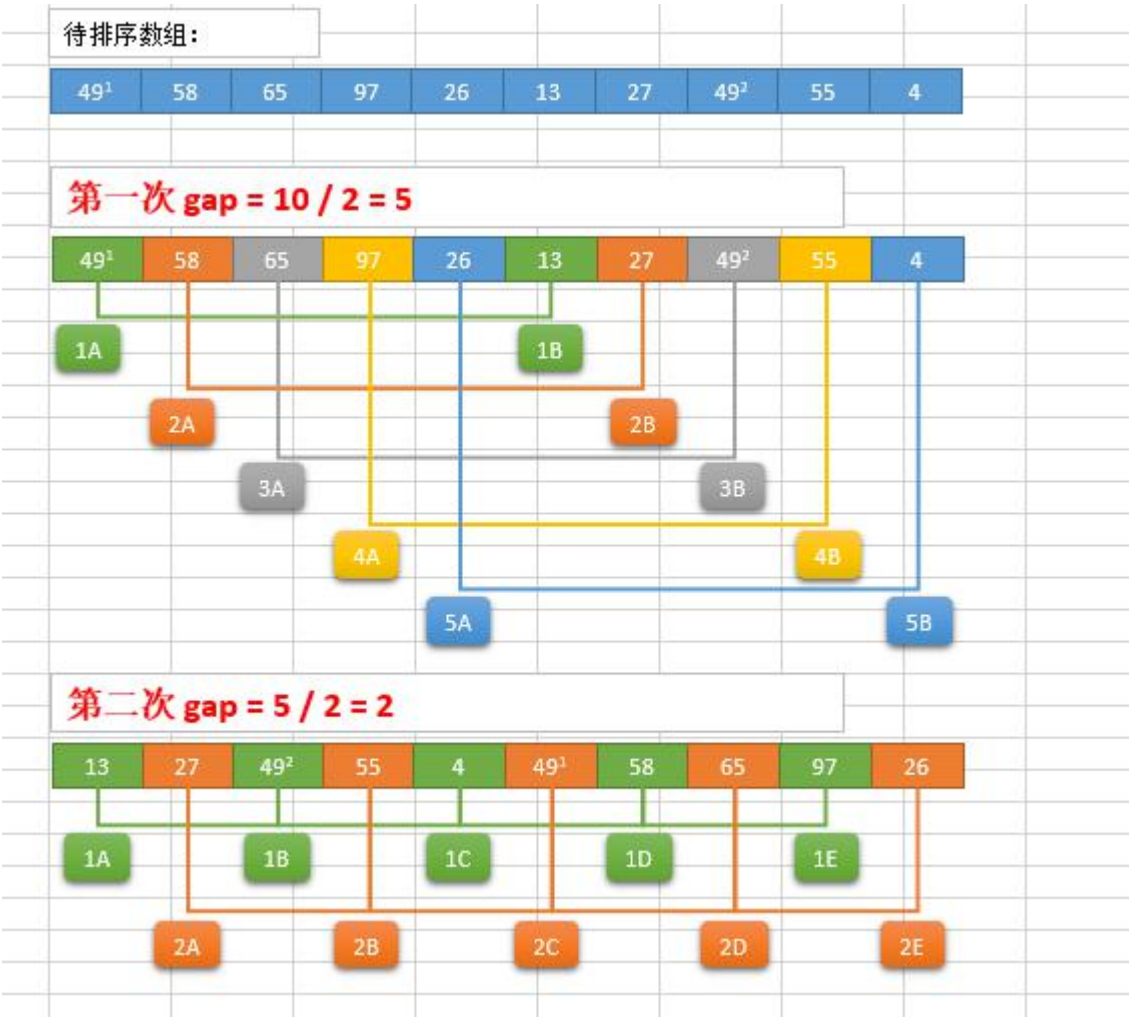
```
10 14 73
25 23 13
27 94 33
39 25 59
94 65 82
45
```

排序之后变为：

```
10 14 13
25 23 33
27 25 59
39 65 73
45 94 82
94
```

最后以1步长进行排序（此时就是简单的插入排序了）

2.希尔排序的分析




```python
def shell_sort(alist):
    n = len(alist)
    # 初始步长
    gap = n / 2
    while gap > 0:
        # 按步长进行插入排序
        for i in range(gap, n):
            j = i
            # 插入排序
            while j>=gap and alist[j-gap] > alist[j]:
                alist[j-gap], alist[j] = alist[j], alist[j-gap]
                j -= gap
```

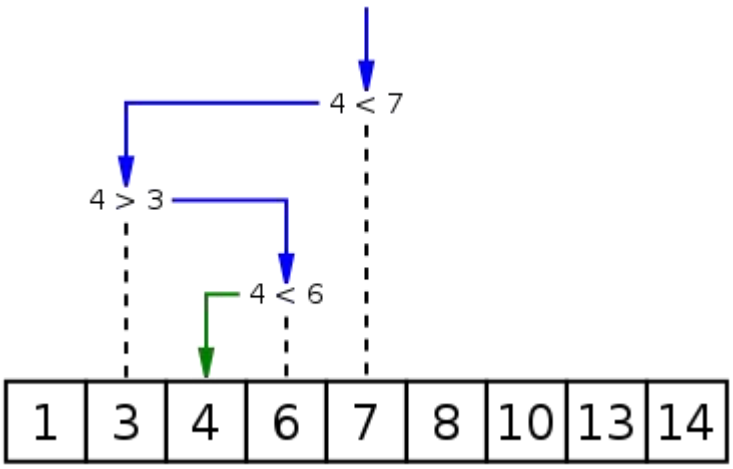
```
        # 得到新的步长
        gap = gap / 2

alist =[22,67,34,23,97,89,56,13,71]
shell_sort(alist)
print(alist)
```

3.时间复杂度

- 最优时间复杂度：根据步长序列的不同而不同
- 最坏时间复杂度：O(n2)
- 稳定性：不稳定

三、二分法查找

二分查找又称折半查找，优点是比较次数少，查找速度快，平均性能好；其缺点是要求待查表为有序表，且插入删除困难。因此，折半查找方法适用于不经常变动而查找频繁的有序列表。首先，假设表中元素是按升序排列，将表中间位置记录的关键字与查找关键字比较，如果两者相等，则查找成功；否则利用中间位置记录将表分成前、后两个子表，如果中间位置记录的关键字大于查找关键字，则进一步查找前一子表，否则进一步查找后一子表。重复以上过程，直到找到满足条件的记录，使查找成功，或直到子表不存在为止，此时查找不成功。



1.二分法查找实现

（非递归实现）

```
def binary_search(alist, item):
      first = 0
      last = len(alist)-1
      while first<=last:
          midpoint = (first + last)/2
          if alist[midpoint] == item:
              return True
          elif item < alist[midpoint]:
              last = midpoint-1
          else:
```

```
                first = midpoint+1
    return False
testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
print(binary_search(testlist, 3))
print(binary_search(testlist, 13))
```

（递归实现）

```
def binary_search(alist, item):
    if len(alist) == 0:
        return False
    else:
        midpoint = len(alist)//2
        if alist[midpoint]==item:
          return True
        else:
          if item<alist[midpoint]:
            return binary_search(alist[:midpoint],item)
          else:
            return  binary_search(alist[midpoint+1:],item)

testlist = [22,67,34,23,97,89,56,13,71]
print(binary_search(testlist, 22))
print(binary_search(testlist, 33))
```

2.时间复杂度

- 最优时间复杂度：O(1)
- 最坏时间复杂度：O(logn)