**严锦 1700011049**

**1.**

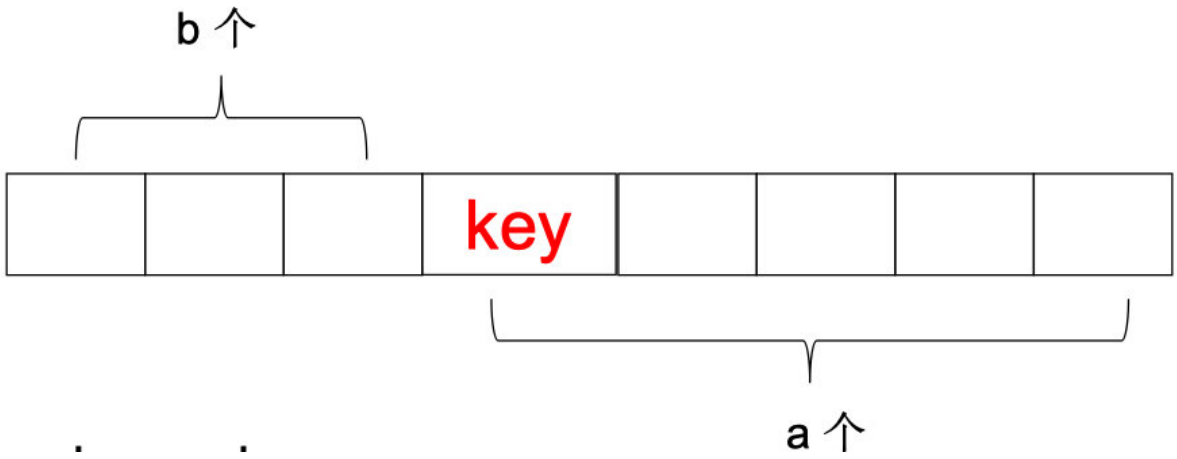
采用高位优先法。先对字符串最高位进行桶式排序，将序列分成若干个桶。然后再对每个桶分别进行次高位排序的桶式排序，分成更小的桶。依次重复，直到对最低位排序后，将所有的桶合并即可。其中需要对字符串长短不一的情况作额外处理，将字符串后端不足的部分用空字符填补，每一步中空字符占据的桶不再进一步进行桶式排序。

由于空字符占据的桶不会进一步处理，因此每一个字符串最多的运算次数为其长度，因此时间复杂度为$\mathcal{O}(\sum_{i=1}^{m} l_i)$

**2.**

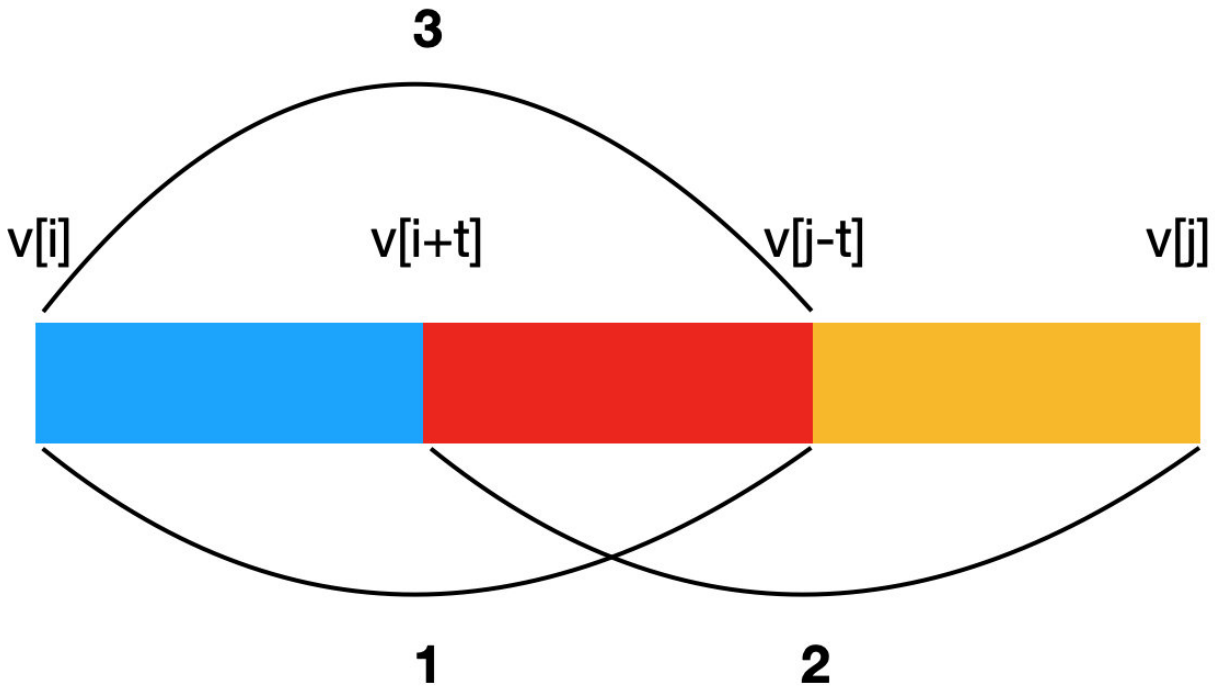
引入操作arrangeRight(k)，把数组前k小的元素都移到数组右侧。

设key=a[0]，将key移到适当位置，使得比key大的元素都在key左边，比key小的元素都在key右边。




若a=k，则停止操作。若a>k，则对这a个元素再次进行arrangeRight，若a<k，则对左边b个元素再次进行arrangeRight。最终数组右侧的k个元素即为最小的k个元素。

**3.**




如图所示，sort(A, i, j - t)后，v[i]到v[j-t]均为有序排列，此时有任意v[k] >= v[i + t] (i+t < k < j-t)，即红色区域中的任意元素大于蓝色区域。sort(A, i + t, j)后，v[i + t]到v[j]为有序排列，此时有任意v[k] >= v[j - t] (j - t < k < j)，即黄色区域中的任意元素大于红色区域。sort(A, i, j - t)后，v[i]到v[j - t]为有序排列。此时黄色区域已经有序，且黄色区域任意元素大于红色区域，而红色区域也已经有序，且红色区域任意元素大于蓝色区域，且蓝色区域也已经有序，因此整个数组完成排序。

每次递归中数组会被划分为3份，最多递归层数为$\log_3(n)$，因此时间复杂度为$\mathcal{O}(n \log n)$

**4.**

1). 对于m个桶中的每个桶，插入排序后均处于有序状态。按顺序扫描各个桶并递增输出，类似于归并排序中的合并步骤，可以最终得到有序的数组，因此算法正确。

2). 时间复杂度$T(n) = mT(\frac{m}{n}) + T_{merge}$，其中
$T_{merge} = k(\frac{n}{m} + \frac{2n}{m} + \ldots + \frac{nm}{m}) = \frac{kn(m+1)}{2} = \mathcal{O}(mn)$

最佳情况下，每一步插入排序时间复杂度均为$m\mathcal{O}(\frac{n}{m}) = \mathcal{O}(n)$，此时总时间复杂度为$\mathcal{O}(mn)$。

一般情况下或最差情况下，每一步插入排序时间复杂度为$m\mathcal{O}((\frac{n}{m})^2) = \mathcal{O}(\frac{n^2}{m})$，此时总时间复杂度为$\mathcal O(\frac{n^2}{m})$

**5.**

| **原下标** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|---|
| 数组A | 20 | 13 | 11 | 11‘ | 19 | 89 | 6 | 4 |
| 索引1的下标 | 6 | 4 | 3 | 3 | 5 | 7 | 1 | 0 |
| 索引2的下标 | 1 | 0 | 2 | 3 | 1 | 4 | 0 | 5 |
| 结果 | 4 | 6 | 11 | 11’ | 13 | 19 | 20 | 89 |

**6.**

```
void Merge(int *a, int start, int mid, int end){
    int i,j,k,tmp;
    for(i=start,j=mid+1;j<=end;++i){
        if(a[i]>a[j]){
            tmp=a[j];
            for(k=j;k>i;--k){
                a[k]=a[k-1];
            }
            a[k]=tmp;
            j++;
        }
    }
}
void MergeSort(int *a, int start, int end){
    if(start<end){
        int mid=(start+end)/2;
        MergeSort(a,start,mid);
        MergeSort(a,mid+1,end);
        Merge(a,start,mid,end);
    }
}
```

时间复杂度为$\mathcal O(n^2\log n)$