线性时间复杂度

```
1 桶排序
2 计数排序
3 基排序
```

桶排序（bucket）

> d：创建n个桶
> n:每个数取值放到对应桶末尾
> d：链接桶

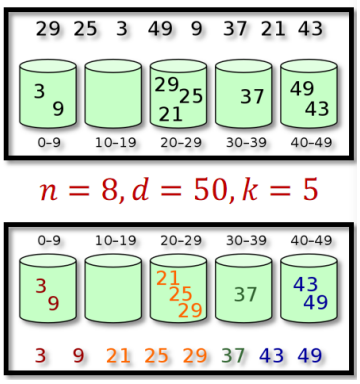
BucketSort(A, d):

```
<L1, L2, …, Ld> = CreateBuckets(d)
for (i=1 to A.length)
  AssignToBucket(A[i])
CombineBuckets(L1, L2, …, Ld)
```

- 复杂度Q（n+d）

改进

> 防止n<<d
> 每个桶d/k大小，使k约等于n
> 桶内插入排序，稳定



$n = 8, d = 50, k = 5$

BucketSort(A, k):

```
<L_1, L_2, …, L_k> = CreateBuckets(k)
for (i=1 to A.length)
  AssignToBucket(A[i])
for (j=1 to k)
  SortWithinBucket(L_j)
CombineBuckets(L_1, L_2, …, L_k)
```

- 复杂度Q（n）

- Runtime is $\Theta(n+k)$, plus cost for sorting within buckets.
- If items are uniformly distributed and we use insertion sort, expected cost for sorting is $O(k \cdot (n/k)^2) = O(n^2/k)$.
- Expected total runtime is $O\big(n + k + (n^2/k)\big)$, which is $O(n)$ when we have $k \approx n$ buckets.
- `BucketSort` can be stable.

基排序（Radix）

> 按位排序，d位，进行d次循环
> 从最低位开始

RadixSort(A, d):

```
for (i=1 to d)
  use-a-stable-sort-to-sort-A-on-digit-i
```

RadixSort(A, d):

```
for (i=1 to d)
  use-bucket-sort-to-sort-A-on-digit-i
```

- 复杂度O（dn）