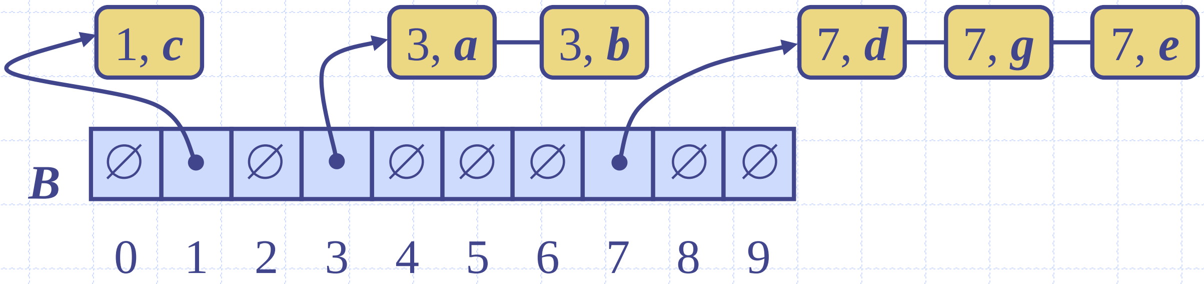
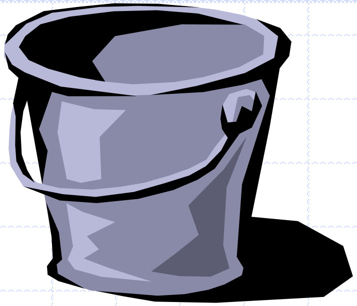
# Bucket-Sort and Radix-Sort



# Bucket-Sort

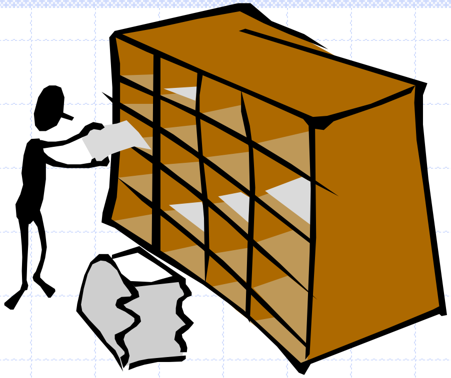


Problem: Sort a sequence $S$ which has $n$ items.

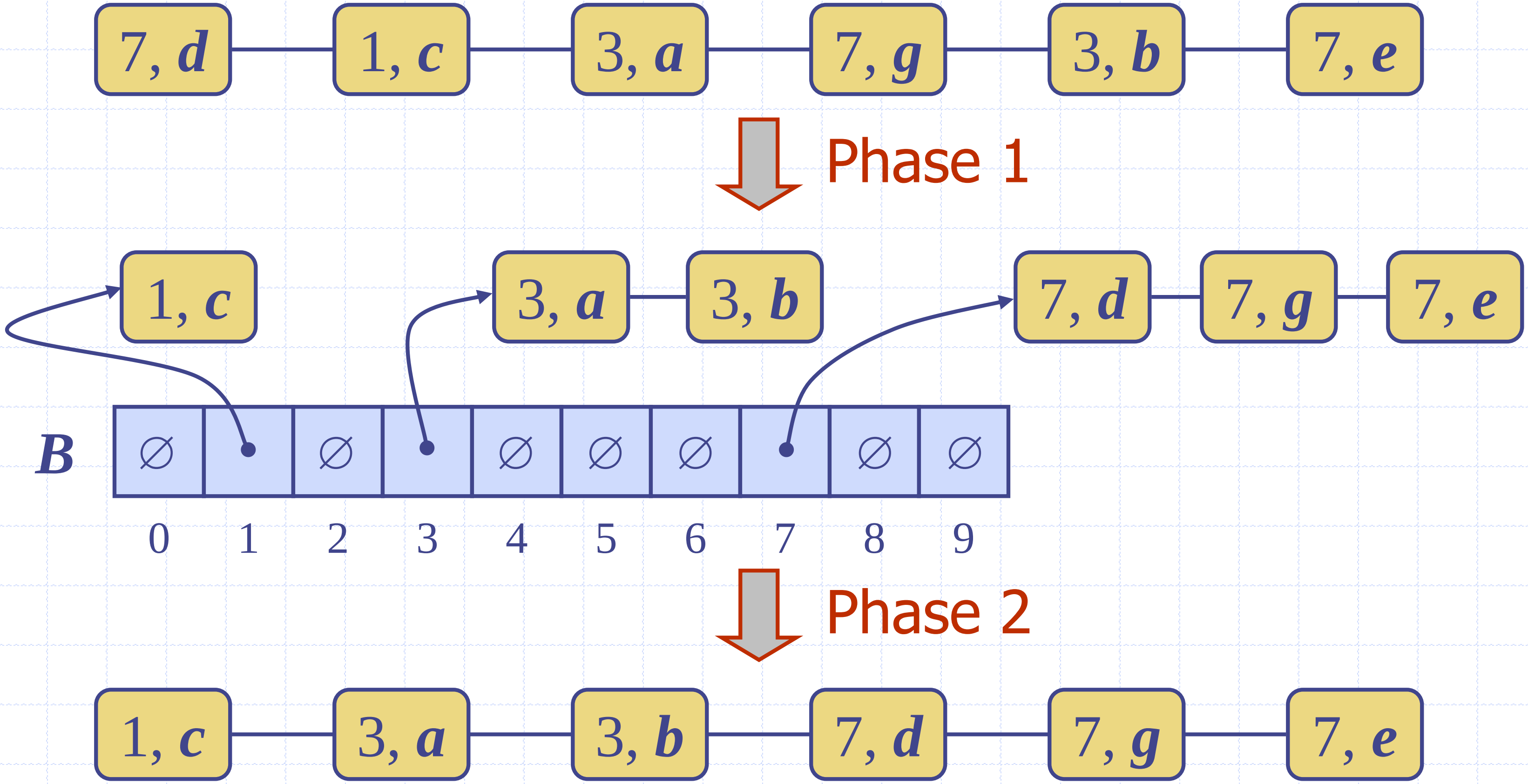
Condition: Each item has a key, and the items should be sorted based on their key values.

Range: The range of the key values is $[0, N-1]$

# Example



- 6 items with key range [0, 9]



# Properties and Complexity

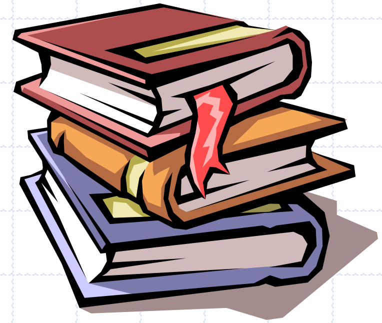
## Stable:

The relative order of any two items with the same key is preserved after the execution of the algorithm

## Complexity:

If there are n items and the range of keys is [0, N] then the complexity of bucket sort is $O(n + N)$.

# Lexicographic Order

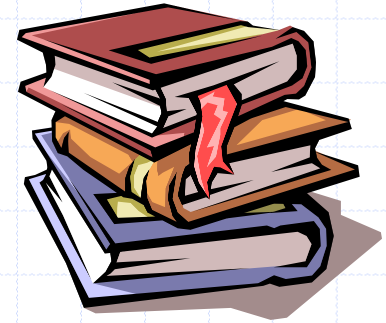


- A $d$-tuple is a sequence of $d$ keys $(k_1, k_2, \ldots, k_d)$, where key $k_i$ is said to be the $i$-th dimension of the tuple
- Example:
  - The Cartesian coordinates of a point in space are a 3-tuple
- The lexicographic order of two $d$-tuples is defined as follows

$$(x_1, x_2, \ldots, x_d) < (y_1, y_2, \ldots, y_d) \text{ if:}$$

$$(x_1 < y_1) \text{ or}$$
$$(x_1 = y_1 \text{ and } x_2 < y_2) \text{ or}$$
$$(x_1 = y_1 \text{ and } x_2 = y_2 \text{ and } x_3 < y_3) \text{ or } \ldots.$$

# Lexicographic Order



- A $d$-tuple is a sequence of $d$ keys $(k_1, k_2, \ldots, k_d)$, where key $k_i$ is said to be the $i$-th dimension of the tuple
- Example:
  - The Cartesian coordinates of a point in space are a 3-tuple
- The lexicographic order of two $d$-tuples is recursively defined as follows

$$(x_1, x_2, \ldots, x_d) < (y_1, y_2, \ldots, y_d)$$

$$\Leftrightarrow$$

$$x_1 < y_1 \ \vee \ x_1 = y_1 \wedge (x_2, \ldots, x_d) < (y_2, \ldots, y_d)$$

# Radix-Sort

- Radix-sort sorts a sequence of $d$-tuples in lexicographic order by executing $d$ times algorithm ***Bucket-sort***, one per dimension
- Radix-sort runs in $O(dT(n))$ time, where $T(n)$ is the running time of ***Bucket-Sort***

**Algorithm** ***Radix-sort***$(S)$

**Input** sequence $S$ of $d$-tuples
**Output** sequence $S$ sorted in lexicographic order

**for** $i \leftarrow 1$ **upto** d
  ***Bucket-sort***$(S, C_i)$

Example:

(7,4,6) (5,1,5) (2,4,6) (2, 1, 4) (3, 2, 4)

# Radix-Sort

- Radix-sort sorts a sequence of $d$-tuples in lexicographic order by executing $d$ times algorithm ***Bucket-sort***, one per dimension
- Radix-sort runs in $O(dT(n))$ time, where $T(n)$ is the running time of ***Bucket-Sort***

**Algorithm** ***Radix-sort***$(S)$

**Input** sequence $S$ of $d$-tuples
**Output** sequence $S$ sorted in lexicographic order

**for** $i \leftarrow 1$ **upto** d
  ***Bucket-sort***$(S, C_i)$

Example:

(7,4,6) (5,1,5) (2,4,6) (2, 1, 4) (3, 2, 4)

This will result in wrong answer!!

# Radix-Sort (other way around)

- Radix-sort sorts a sequence of $d$-tuples in lexicographic order by executing $d$ times algorithm ***Bucket-sort***, one per dimension
- Radix-sort runs in $O(dT(n))$ time, where $T(n)$ is the running time of ***Bucket-Sort***

**Algorithm** *lexicographicSort*($S$)
**Input** sequence $S$ of $d$-tuples
**Output** sequence $S$ sorted in lexicographic order

**for** $i \leftarrow d$ **downto** 1
*stableSort*($S$, $C_i$)

Example:

(7,4,6) (5,1,5) (2,4,6) (2, 1, 4) (3, 2, 4)

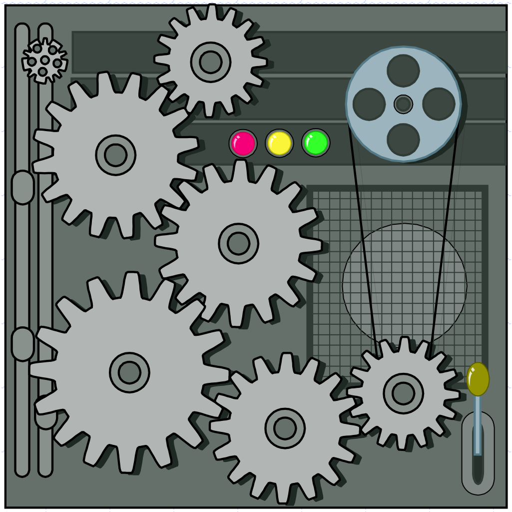
(2, 1, 4) (3, 2, 4) (5,1,5) (7,4,6) (2,4,6)

(2, 1, 4) (5,1,5) (3, 2, 4) (7,4,6) (2,4,6)

(2, 1, 4) (2,4,6) (3, 2, 4) (5,1,5) (7,4,6)

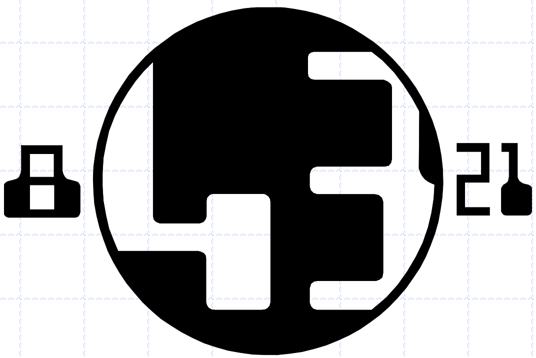
This is the correct way!

# Complexity



- Radix-sort runs in time $O(d(n + N))$

# Radix-Sort for Binary Numbers




- Consider a sequence of $n$ $b$-bit integers

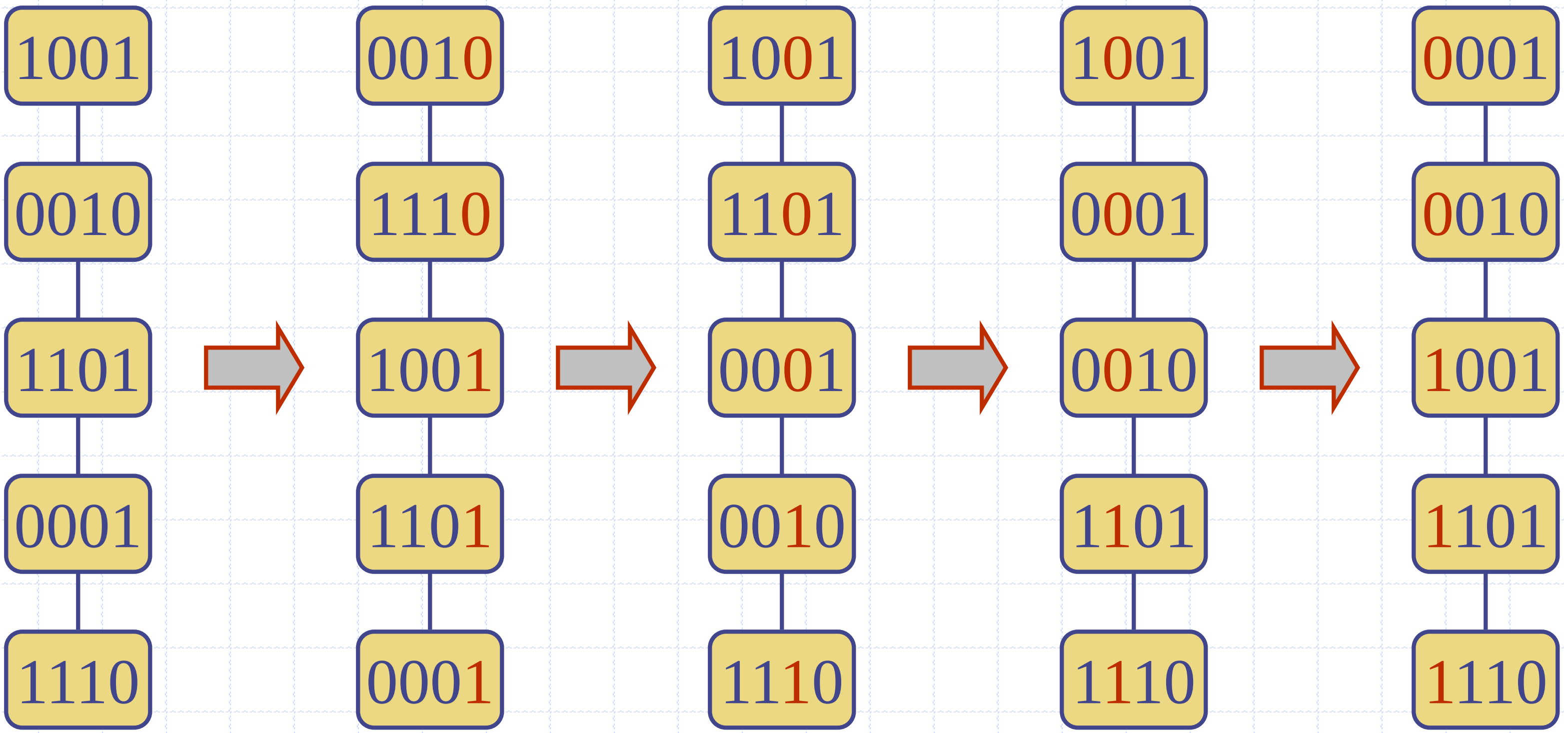
  $$x = x_{b-1} \ldots x_1 x_0$$

- We represent each element as a $b$-tuple of integers in the range $[0, 1]$ and apply radix-sort with $N = 2$
- This application of the radix-sort algorithm runs in $O(bn)$ time
- For example, we can sort a sequence of 32-bit integers in linear time

**Algorithm** ***binaryRadixSort***($S$)
    **Input** sequence $S$ of $b$-bit integers
    **Output** sequence $S$ sorted
    replace each element $x$ of $S$ with the item $(0, x)$
    **for** $i \leftarrow 0$ **to** $b - 1$
        replace the key $k$ of each item $(k, x)$ of $S$ with bit $x_i$ of $x$
        ***bucketSort***($S$, 2)

# Example

- Sorting a sequence of 4-bit integers



- Given a list $a_1, a_2, a_3, \ldots, a_n$ s.t each $a_i \in [0, n^3-1]$.
- How will you sort the list ?
- What is the time required in Sorting?

<u>Example</u>: If $n = 10$, then each $a_i \in [0, 999]$

Say $a_i = 567$

we have $a_i = (5 \times 10^2) + (6 \times 10^1) + (7 \times 10^0)$

<u>Generalization</u> If $a_i \in [0, n^3-1]$ s.t $n$ is a natural number, then $a_i$ Can be written as a triple $(x_i, y_i, z_i)$ s.t

$$\boxed{a_i = (x_i \times n^2) + (y_i \times n^1) + (z_i \times n^0)}$$

Here each $x_i, y_i, z_i \in [0, n-1]$

$x_i = \text{quotient}(a_i, n^2)$

$z_i = \text{remainder}(a_i, n)$

- If $a_i, a_j \in [0, n^3-1]$, then

$$\boxed{a_i \le a_j \quad \text{iff} \quad (x_i, y_i, z_i) \le (x_j, y_j, z_j)}$$

- The original problem boils down to sorting $n$ triples.

- Recall that Radix Sort takes $O(n+N)$ time.

- In our case $N = n$.

- Hence sorting of the list will take $O(n+n) = O(n)$ time.

A list of n elts $(x_1, x_2, \dots, x_n)$ $\longrightarrow$ $\boxed{\text{Permutation Algo}}$ $\longrightarrow$ A reordering of elts s.t. each possibility is equally likely.

## RANDOMIZATION

- Let Random(k) be a function st it returns an integer in the range $[0, k-1]$
- The function is Uniform
- The function is independent.

P. Algo( )

I/P : $X = (x_1, x_2, \ldots, x_n)$

O/P : Some permutation of X

For $i = 1$ to $n$

$r_i$ = Random($n^3$)

Associate $r_i$ with $x_i$

Sort $X = ((x_1, r_1), (x_2, r_2), \ldots, (x_n, r_n))$

using $r_i$'s as Keys.

If all the $r_i$'s are distinct then

return the sorted X.

Else

P. Algo( )

# ANALYSIS of P. Algo( )

- If all the keys are distinct in the Ist iteration itself, then the P. Algo( ) runs in $O(n)$ time.

- What is the probability that all the keys are distinct in the Ist iteration? Let's determine.

- The probability that $r_1 = r_2$ is $\frac{1}{n^3}$

- The probability that $(r_1 = r_2) \vee (r_1 = r_3) \vee (r_1 = r_4) \vee \dots \vee (r_1 = r_n)$ is $\frac{n}{n^3}$ [Max]

- The probability that for some $(i \neq j)$ $r_i = r_j$ is $\frac{n}{n^3} \times n = \frac{1}{n}$ [Max]

- The probability that all the $r_i$'s are distinct in the Ist iteration is at least $\left(1 - \frac{1}{n}\right)$

Note that the probability $\left(1-\frac{1}{n}\right)$ is deemed v. good (v. high)

$$\because \quad \operatorname*{Lt}_{n\to\infty}\left(1-\frac{1}{n}\right) = 1$$

<u>Theorem</u> : Given a list $X = (x_1, x_2, \ldots, x_n)$, we can permute the list uniformly in $O(n)$ time with probability $\left(1-\frac{1}{n}\right)$.