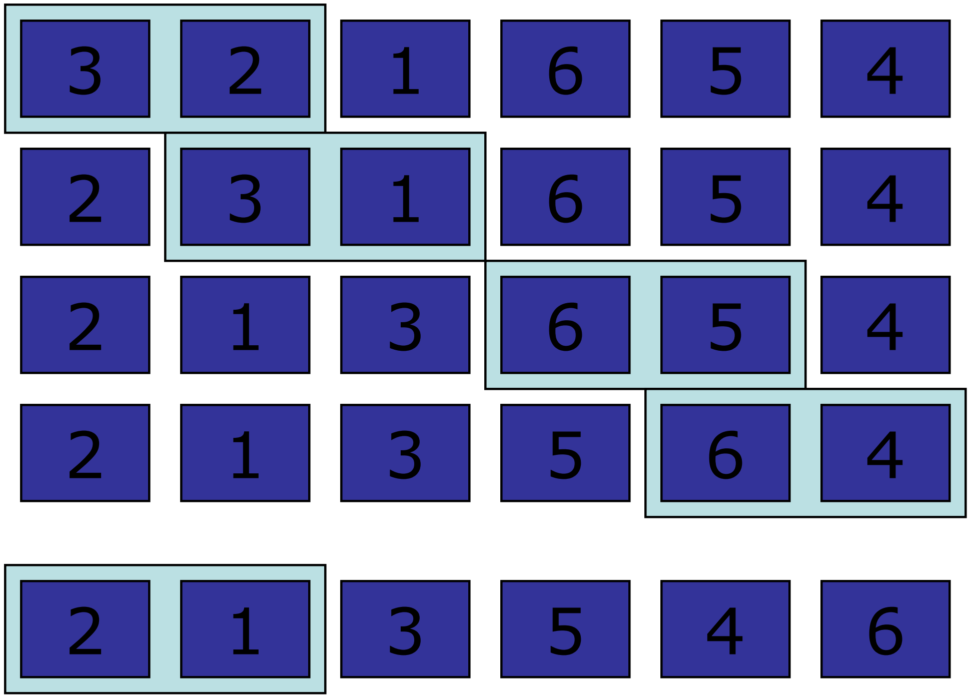
# Survey of Sorting

Ananda Gunawardena

# Naïve sorting algorithms

- Bubble sort: *scan for flips, until all are fixed*



3 2 1 6 5 4

2 3 1 6 5 4

2 1 3 6 5 4

2 1 3 5 6 4

2 1 3 5 4 6

Etc…

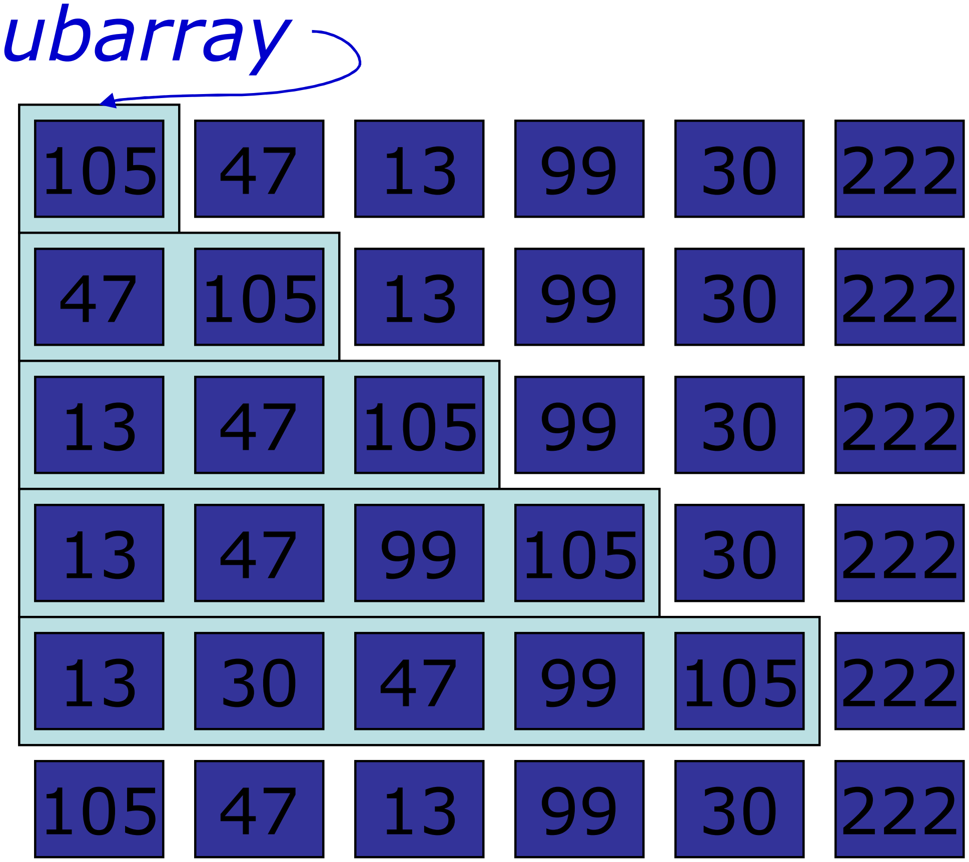
# Naïve Sorting

```
for i=1 to n-1
  { for j=0 to n-i-1
      if (A[j].compareTo(A[j+1])>0)
          swap(A[j], A[j+1]);
    if (no swaps) break;
  }
```

- What happens if
  - All keys are equal?
  - Keys are sorted in reverse order?
  - Keys are sorted?
  - keys are randomly distributed?

- Exercise: Count the number of operations in bubble sort and find a Big O analysis for bubble sort

# Insertion sort



*Sorted subarray*

| | | | | | |
|---|---|---|---|---|---|
| 105 | 47 | 13 | 99 | 30 | 222 |
| 47 | 105 | 13 | 99 | 30 | 222 |
| 13 | 47 | 105 | 99 | 30 | 222 |
| 13 | 47 | 99 | 105 | 30 | 222 |
| 13 | 30 | 47 | 99 | 105 | 222 |
| 105 | 47 | 13 | 99 | 30 | 222 |

# Insertion sort

- **Algorithm**

```
for  i = 1  to  n-1  do
     insert a[i] in the proper place
          in  a[0:i-1]
```

- **Correctness**
  - Note: after i steps, the sub-array A[0:i] is sorted

# How fast is insertion sort?

To insert a[i] into a[0:i-1], **slide** all elements larger than a[i] to the right.

```
tmp = a[i];
for (j = i; j>0 && a[j-1]>tmp; j--)
     a[j] = a[j-1];
a[j] = tmp;
```

# of slides = O(#inversions)

very fast if array is nearly sorted to begin with

# Selection sort

- **Algorithm**

```
for  i = n-1  to  1  do
     Find the largest entry in the
          in  the subarray A[0:i]
     Swap with A[i]
```

**What is the runtime complexity of selection sort?**

# Sorting Comparison

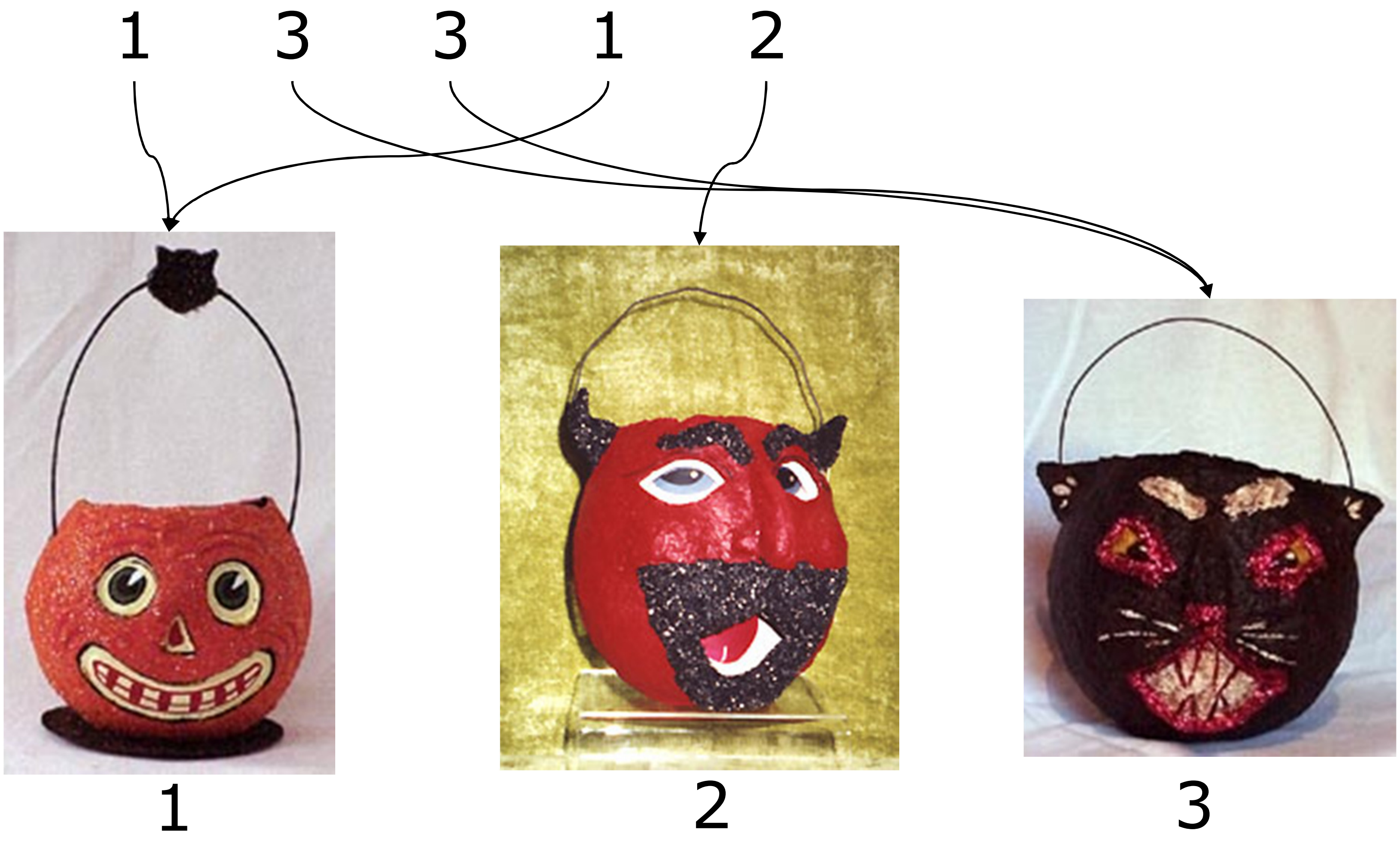
- Discuss the pros and cons of each of the naïve sorting algorithms

# *Bucket Sort*

# Bucket sort

- In addition to comparing pairs of elements, we require these additional restrictions:
  - all elements are non-negative integers
  - all elements are less than a predetermined maximum value
- Elements are usually keys paired with other data

# Bucket sort



1 3 3 1 2

1 2 3

# Bucket sort characteristics

- Runs in *O*(N) time.
- Easy to implement each bucket as a linked list.
- Is *stable*:
  - If two elements (A,B) are equal with respect to sorting, and they appear in the input in order (A,B), then they remain in the same order in the output.

# *Radix Sort*

# Radix sort

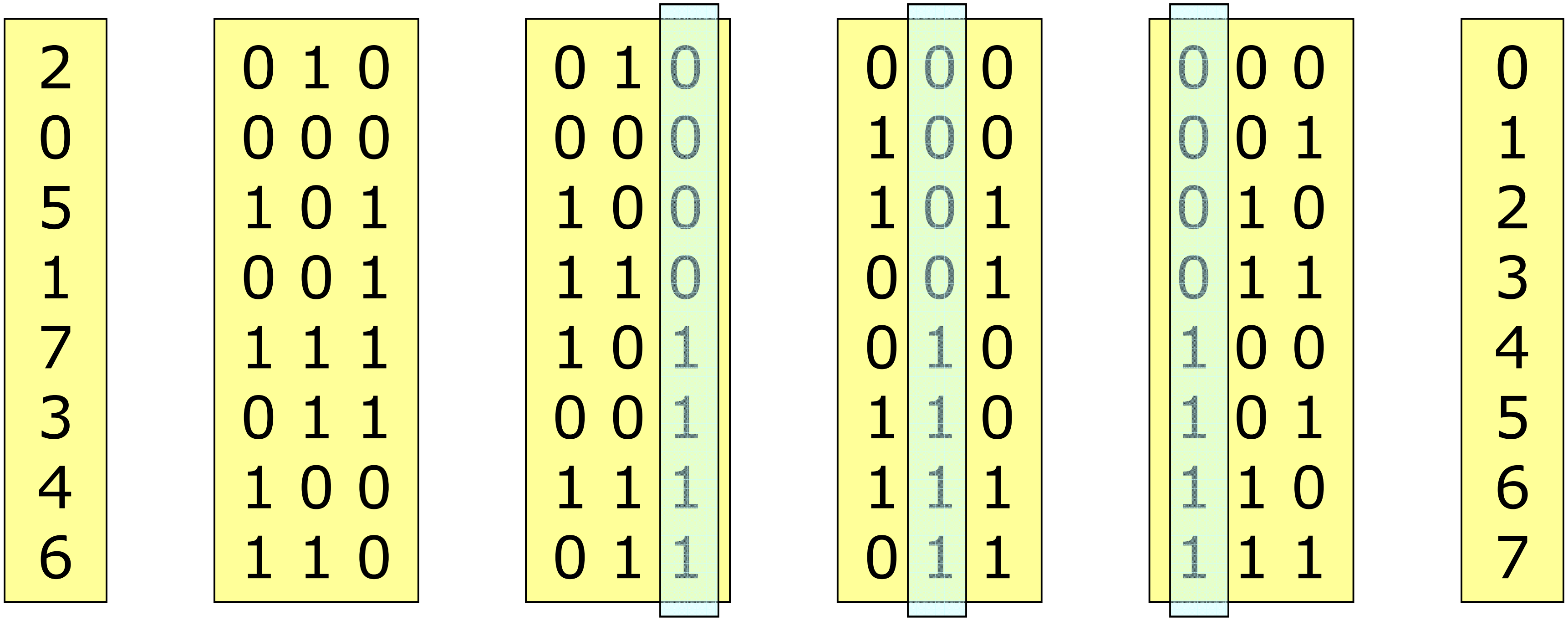
- If your integers are in a larger range then do bucket sort on each digit

- Start by sorting with the low-order digit using a STABLE bucket sort.

- Then, do the next-lowest,and so on

# Radix sort

- Example:



| | | | | | |
|---|---|---|---|---|---|
| 2 | 0 1 0 | 0 1 0 | 0 0 0 | 0 0 0 | 0 |
| 0 | 0 0 0 | 0 0 0 | 1 0 0 | 0 0 1 | 1 |
| 5 | 1 0 1 | 1 0 0 | 1 0 1 | 0 1 0 | 2 |
| 1 | 0 0 1 | 1 1 0 | 0 0 1 | 0 1 1 | 3 |
| 7 | 1 1 1 | 1 0 1 | 0 1 0 | 1 0 0 | 4 |
| 3 | 0 1 1 | 0 0 1 | 1 1 0 | 1 0 1 | 5 |
| 4 | 1 0 0 | 1 1 1 | 1 1 1 | 1 1 0 | 6 |
| 6 | 1 1 0 | 0 1 1 | 0 1 1 | 1 1 1 | 7 |

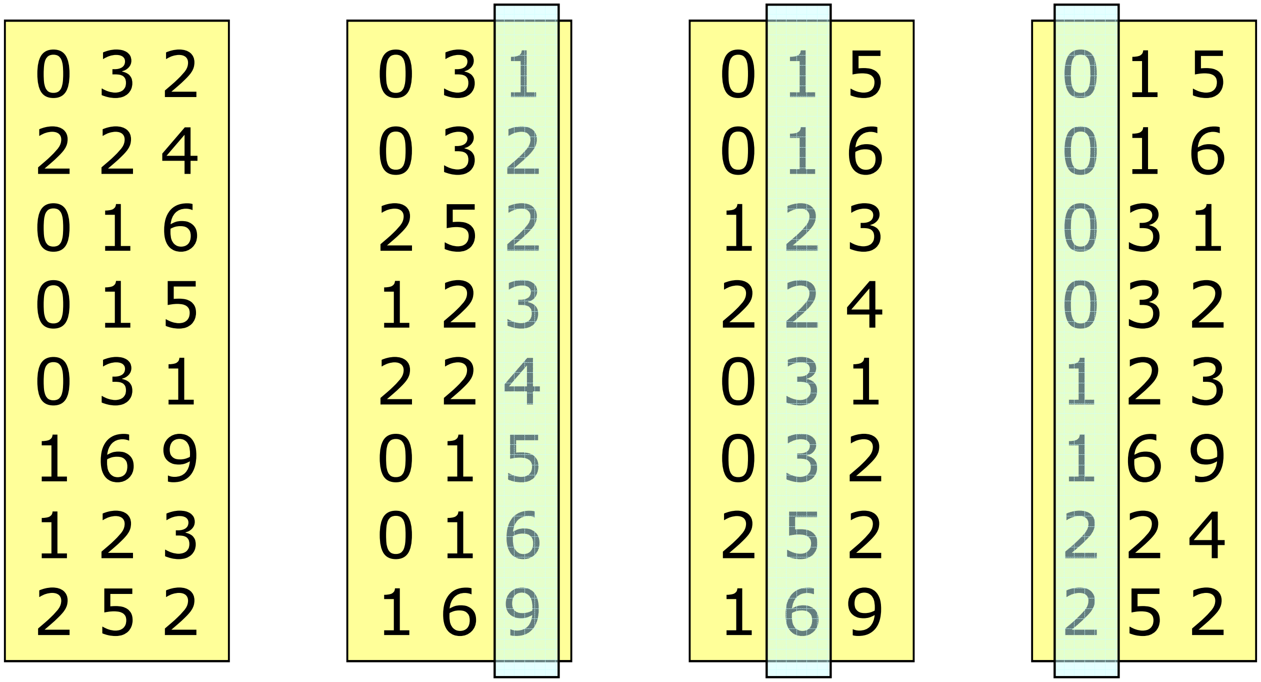
*Each sorting step must be stable.*

# Radix sort characteristics

- Each sorting step can be performed via bucket sort, and is thus $O(N)$.
- If the numbers are all b bits long, then there are b sorting steps.
- Hence, radix sort is $O(bN)$.

# What about non-binary?

- Radix sort can be used for decimal numbers and alphanumeric strings.



| | | | |
|---|---|---|---|
| 0 3 2 | 0 3 1 | 0 1 5 | 0 1 5 |
| 2 2 4 | 0 3 2 | 0 1 6 | 0 1 6 |
| 0 1 6 | 2 5 2 | 1 2 3 | 0 3 1 |
| 0 1 5 | 1 2 3 | 2 2 4 | 0 3 2 |
| 0 3 1 | 2 2 4 | 0 3 1 | 1 2 3 |
| 1 6 9 | 0 1 5 | 0 3 2 | 1 6 9 |
| 1 2 3 | 0 1 6 | 2 5 2 | 2 2 4 |
| 2 5 2 | 1 6 9 | 1 6 9 | 2 5 2 |