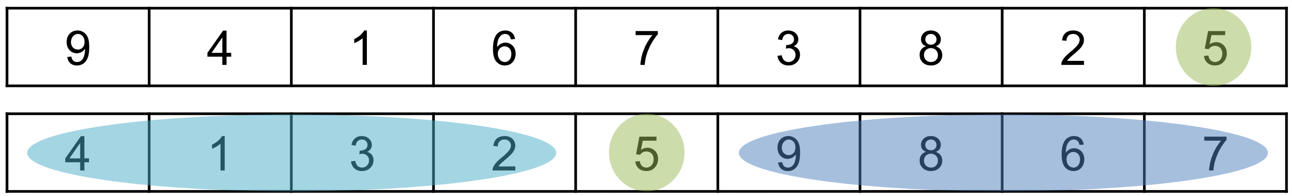
Non-comparison Sorts

Kuan-Yu Chen (陳冠宇)

2019/04/10 @ TR-310-1, NTUST

Review

- Quick sort is a widely used sorting algorithm developed by C. A. R. Hoare
  - Quick sort is also known as partition exchange sort




```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

- The running time of the partition function
  - Worst-case partition: $T(n) = \Theta(n^2)$
  - Best-case partition: $T(n) = \Theta(n \log_2 n)$
  - RANDOMIZED-PARTITION is $O(n \log_2 n)$

Counting Sort.

- ***Counting sort*** assumes that each of the n input elements is an integer in the range 0 to k
  - It first determines the number of elements less than a given element x
  - The information is used to place element x directly into its position in the output array

- In the code for counting sort
  - The input is an array $A[1 \dots n]$
    - $A.length = n$
  - The array $B[1 \dots n]$ holds the sorted output
  - The array $C[0 \dots k]$ provides temporary working storage

Example.

- Please sort a given array by using counting sort

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

  - Step1: Counting the frequencies

	0	1	2	3	4	5
C	2	0	2	3	0	1

  - Step2: Determining the number of elements less than x

	0	1	2	3	4	5
C	2	2	4	7	7	8

  - Step3: Putting each element at its own correct position

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

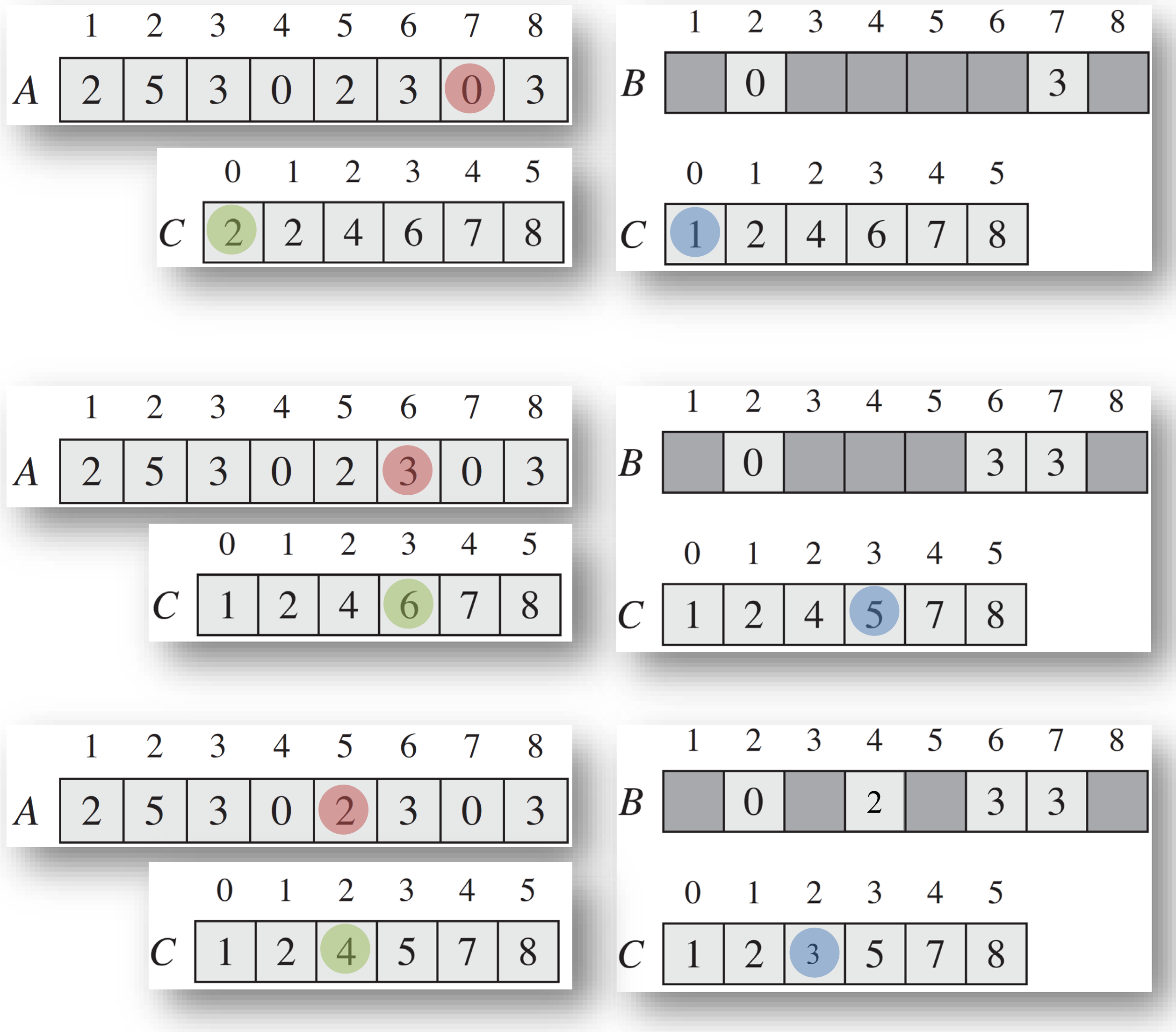
	0	1	2	3	4	5
C	2	2	4	7	7	8

	1	2	3	4	5	6	7	8
B							3	

	0	1	2	3	4	5
C	2	2	4	6	7	8

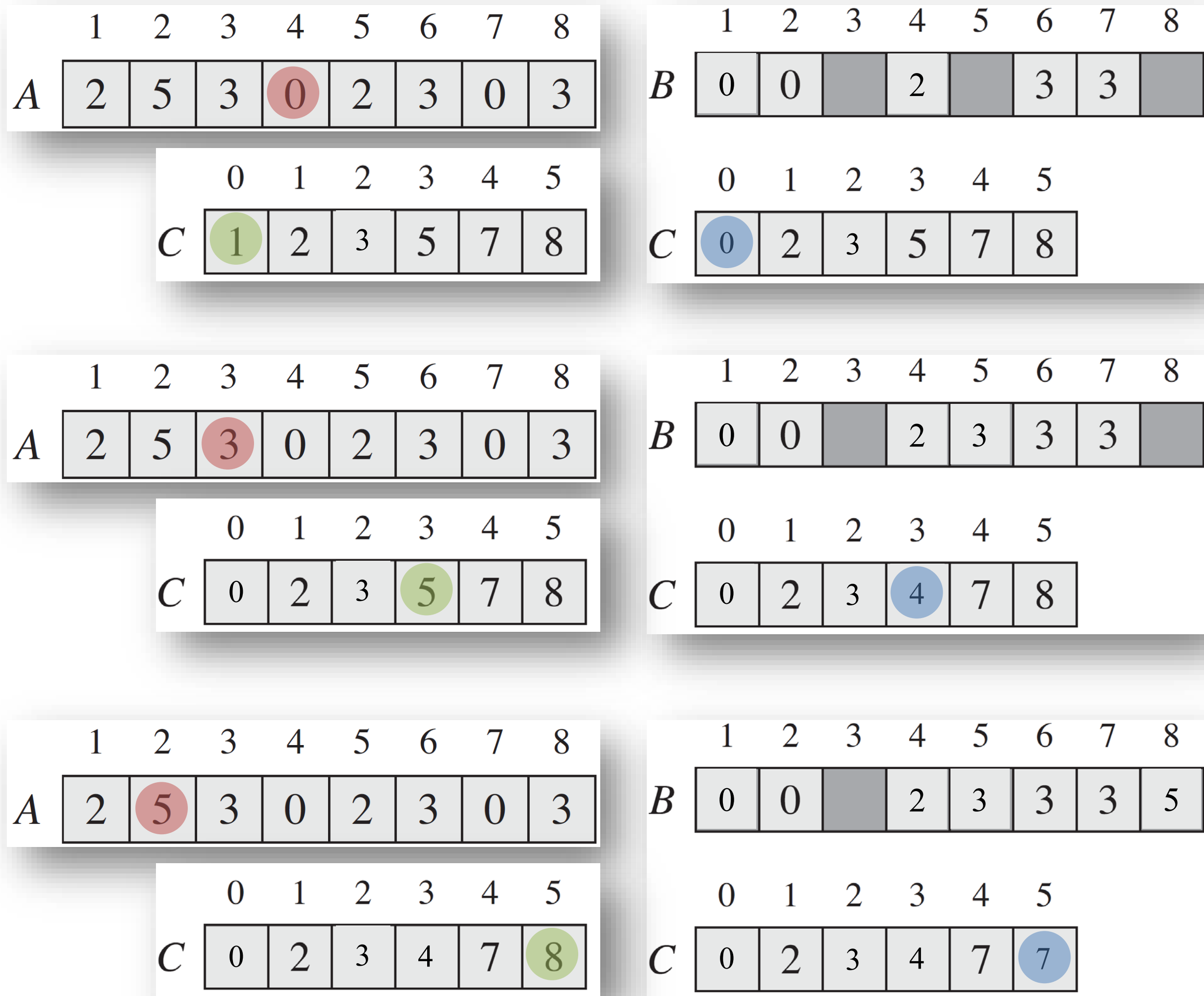
Example..

– Step3: Putting each element at its own correct position



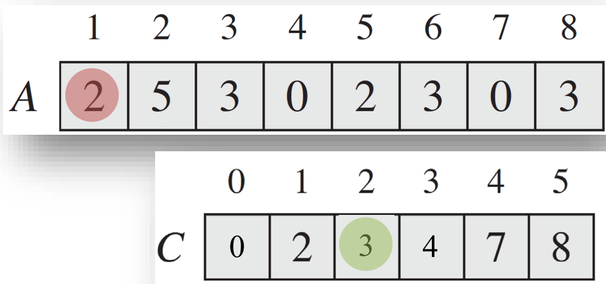
Example…

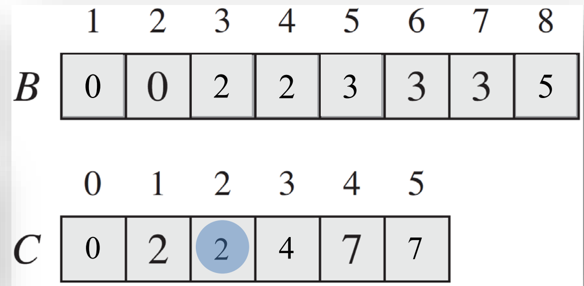
– Step3: Putting each element at its own correct position



Example….

– Step3: Putting each element at its own correct position





Counting Sort..

```
COUNTING-SORT(A, B, k)
let C[0..k] be a new array
for i = 0 to k
    C[i] = 0
for j = 1 to A.length
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i - 1]
// C[i] now contains the number of elements less than or equal to i.
for j = A.length downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

Counting the frequencies

Determining the number of elements less than x

Putting each element at its own correct position

Analyses

- The overall time for counting sort is $\Theta(n + k)$
  - In practice, we usually use counting sort when we have $k = O(n)$, in which case the running time is $\Theta(n\ )$

```
COUNTING-SORT(A, B, k)
let C[0..k] be a new array
for i = 0 to k
    C[i] = 0
for j = 1 to A.length
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i − 1]
// C[i] now contains the number of elements less than or equal to i.
for j = A.length downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] − 1
```

$\Theta(k)$

Counting the frequencies, $\Theta(n)$

Determining the number of elements less than x, $\Theta(k)$

Putting each element at its own correct position, $\Theta(n)$

Radix Sort.

- ***Radix sort*** is a linear sorting algorithm for **integers** and uses the concept of sorting names in alphabetical order
  - Radix sort is also known as bucket sort?

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Example.

- Sort the given numbers using radix sort

345, 654, 924, 123, 567, 472, 555, 808, 911

  - The first step: The numbers are sorted according to the digit at ones place
    - The new order is 911, 472, 123, 654, 924, 345, 555, 567, 808

Number	0	1	2	3	4	5	6	7	8	9
345						345				
654					654					
924					924					
123				123						
567								567		
472			472							
555						555				
808									808	
911		911								

Example..

– Based on the new order: 911, 472, 123, 654, 924, 345, 555, 567, 808

– The second step: The numbers are sorted according to the digit at the tens place
  - Consequently, the new order is: 808, 911, 123, 924, 345, 654, 555, 567, 472

Number	0	1	2	3	4	5	6	7	8	9
911		911								
472								472		
123			123							
654						654				
924			924							
345					345					
555						555				
567							567			
808	808									

Example…

- Based on the new order: 808, 911, 123, 924, 345, 654, 555, 567, 472

- The third step is: The numbers are sorted according to the digit at the hundreds place
  - Finally, the ordered sequence is: 123, 345, 555, 567, 654, 808, 911, 924

Number	0	1	2	3	4	5	6	7	8	9
808									808	
911										911
123		123								
924										924
345				345						
654							654			
555						555				
567						567				
472					472					

Radix Sort..

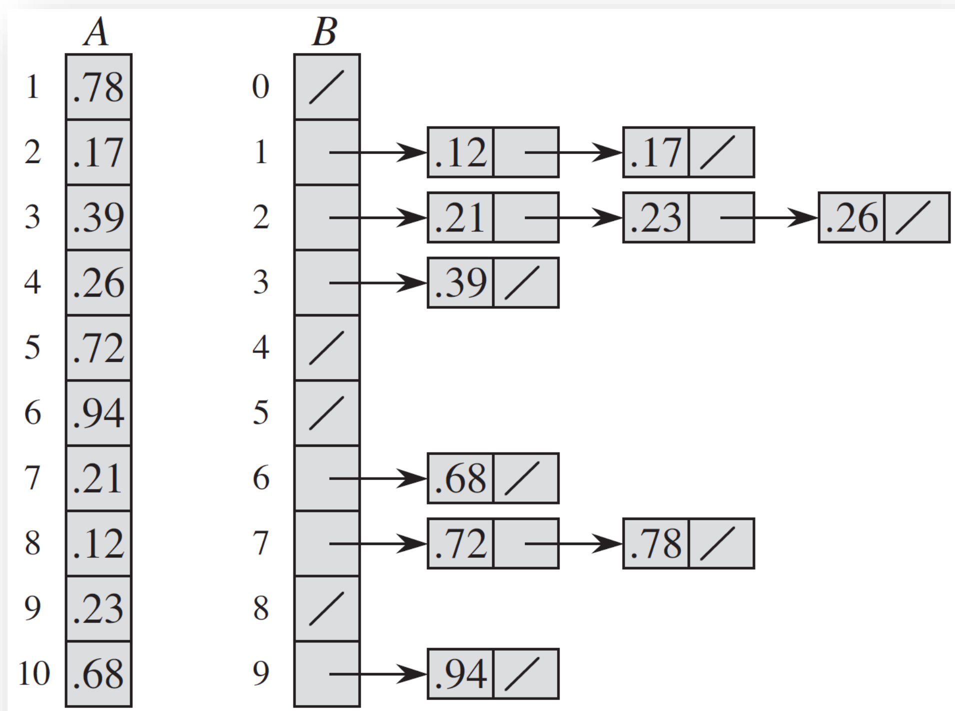
- The code for radix sort is straightforward
  - It assumes that each element in the n-element array A has d digits
    - Digit 1 is the lowest-order digit and digit d is the highest-order digit

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

- Given n d-digit numbers in which each digit can take on up to k possible values
  - RADIX-SORT correctly sorts these numbers in $\Theta(d(n+k))$
    - Since the sorting function (counting sort) takes $\Theta(n+k)$ time
  - When d is constant and $k = O(n)$, we can make radix sort run in linear time $\Theta(n)$

Bucket Sort.

- ***Bucket sort*** assumes that the input is drawn from a uniform distribution
  - It divides the interval [0,1) into n equal-sized subintervals, or **buckets**
  - To produce the output, we simply sort the numbers in each bucket and then go through the buckets in order, listing the elements in each



Bucket Sort..

- The code for bucket sort assumes that the input is an n-element array A and that each element $A[i]$ in the array satisfies $0 \leq A[i] < 1$
  - It requires an auxiliary array $B[0, \ldots, n-1]$ of linked lists (buckets)

```
BUCKET-SORT(A)
let B[0..n − 1] be a new array
n = A.length
for i = 0 to n − 1
    make B[i] an empty list
for i = 1 to n
    insert A[i] into list B[⌊nA[i]⌋]
for i = 0 to n − 1
    sort list B[i] with insertion sort
concatenate the lists B[0], B[1], ..., B[n − 1] together in order
```

Analyses.

- The running time of bucket sort is

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

```
BUCKET-SORT(A)
let B[0..n − 1] be a new array
n = A.length
for i = 0 to n − 1
        make B[i] an empty list
for i = 1 to n
        insert A[i] into list B[⌊nA[i]⌋]
for i = 0 to n − 1
        sort list B[i] with insertion sort
concatenate the lists B[0], B[1], ..., B[n − 1] together in order
```

$\Theta(n)$

$\Theta(n)$

$\sum_{i=0}^{n-1} O(n_i^2)$

Analyses..

- For analyzing the average-case running time of bucket sort, we take the expectation over the input distribution

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} \mathrm{O}(n_i^2)$$

$$\mathrm{E}[T(n)] = \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} \mathrm{O}(n_i^2)\right] = \mathrm{E}[\Theta(n)] + \mathrm{E}\left[\sum_{i=0}^{n-1} \mathrm{O}(n_i^2)\right]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} \mathrm{E}\left[\mathrm{O}(n_i^2)\right] = \Theta(n) + \sum_{i=0}^{n-1} \mathrm{O}\left[\mathrm{E}(n_i^2)\right]$$

  - Next, we define indicator random variables X_{ij}

$$X_{ij} = \mathrm{I}\{A[j] \text{ falls in bucket } i\}$$

$$n_i = \sum_{j=1}^{n} X_{ij}$$

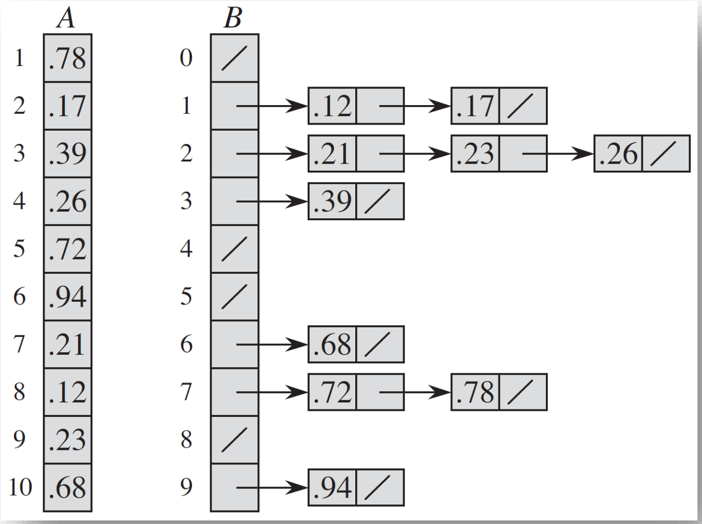
Analyses…

– To compute $E[n_i^2]$, we expand the square and regroup terms

$$E[n_i^2] = E\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] = E\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right]$$

$$= E\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{j=1}^{n}\sum_{k=1 \& j\neq k}^{n} X_{ij}X_{ik}\right]$$

$$= \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{j=1}^{n}\sum_{k=1 \& j\neq k}^{n} E[X_{ij}X_{ik}]$$



– It should be noted that $P(X_{ij} = 1) = \frac{1}{n}$

$$E[X_{ij}^2] = 1^2 \times \frac{1}{n} = \frac{1}{n}$$

- $\because X_{ij}$ and X_{ik} are independent
- $\therefore E[X_{ij}X_{ik}] = E[X_{ij}]E[X_{ik}]$

$$E[X_{ij}X_{ik}] = E[X_{ij}]E[X_{ik}] = \left(1 \times \frac{1}{n}\right) \times \left(1 \times \frac{1}{n}\right) = \frac{1}{n^2}$$

Analyses….

– Thus, we obtain

$$
\begin{aligned}
E[n_i^2] &= \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{j=1}^{n} \sum_{k=1 \& j \neq k}^{n} E[X_{ij} X_{ik}] \\
&= \sum_{j=1}^{n} \frac{1}{n} + \sum_{j=1}^{n} \sum_{k=1 \& j \neq k}^{n} \frac{1}{n^2} \\
&= n \times \frac{1}{n} + n \times (n-1) \times \frac{1}{n^2} \\
&= 2 - \frac{1}{n}
\end{aligned}
$$

- Finally, we conclude that the average-case running time for bucket sort is linear!

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

$$E[T(n)] = \Theta(n) + \sum_{i=0}^{n-1} O[E(n_i^2)] = \Theta(n) + n \times O\left(2 - \frac{1}{n}\right) = \Theta(n)$$

Conclusions.

- We can categorize that
  - Comparison Sorts
    - The sorted order they determine is based only on comparisons between the input elements
    - Insertion Sort, Merge Sort, Quick Sort
  - Non-comparison Sorts
    - Counting Sort, Radix Sort, Bucket Sort

Algorithm	Worst-case running time	Average-case/expected running time
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Heapsort	$O(n \lg n)$	—
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (expected)
Counting sort	$\Theta(k + n)$	$\Theta(k + n)$
Radix sort	$\Theta(d(n + k))$	$\Theta(d(n + k))$
Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (average-case)

Conclusions..

- https://en.wikipedia.org/wiki/Best,_worst_and_average_case

Algorithm	Worst-case running time	Average-case/expected running time	Best-case running time
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$	$O(n)$
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n \log_2 n)$
Heapsort	$O(n \lg n)$	$O(n \lg n)$	$O(n)$
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (expected)	$\Theta(n \log_2 n)$
Counting sort	$\Theta(k + n)$	$\Theta(k + n)$	$\Theta(k + n)$
Radix sort	$\Theta(d(n + k))$	$\Theta(d(n + k))$	$\Theta(d(k + n))$
Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (average-case)	$\Theta(n)$

Conclusions…

- Stable & Unstable sorting algorithm
  - A sorting algorithm is said to be stable if two elements with equal keys appear in the same order in the sorted output as they appear in the unsorted input
  - Stable Sorts
    - Insertion sort, merge sort, counting sort, radix sort, bucket sort
  - Unstable Sorts
    - Heap sort, quick sort

Questions?

kychen@mail.ntust.edu.tw