Non-comparison Sorts

Kuan-Yu Chen (陳冠宇)

2019/04/10 @ TR-310-1, NTUST
Quick sort is a widely used sorting algorithm developed by C. A. R. Hoare. Quick sort is also known as partition exchange sort.

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 \( A[1 \ldots n] \)
    - \( A.length = n \)
  - The array \( B[1 \ldots n] \) holds the sorted output
  - The array \( C[0 \ldots k] \) provides temporary working storage
Example.

- Please sort a given array by using counting sort

- Step1: Counting the frequencies

- Step2: Determining the number of elements less than $x$

- Step3: Putting each element at its own correct position
Example.

- Step 3: Putting each element at its own correct position

![Diagram showing the process of repositioning elements to their correct positions.](image)
Example...

- Step 3: Putting each element at its own correct position

A

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
2 & 5 & 3 & 0 & 2 & 3 & 0 & 3 \\
0 & 1 & 2 & 3 & 4 & 5 & & \\
1 & 2 & 3 & 5 & 7 & 8 & & \\
\end{array}
\]

B

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
0 & 0 & & 2 & 3 & 3 & & \\
0 & 1 & 2 & 3 & 4 & 5 & & \\
0 & 2 & 3 & 4 & 7 & 8 & & \\
\end{array}
\]

C

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 2 & 3 & 5 & 7 & 8 & & \\
0 & 2 & 3 & 4 & 7 & 8 & & \\
\end{array}
\]
- Step 3: Putting each element at its own correct position

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort

```
COUNTING-SORT(A, B, k)
1  let C[0..k] be a new array
2  for i = 0 to k
3      C[i] = 0
4  for j = 1 to A.length
5      C[A[j]] = C[A[j]] + 1
6    // C[i] now contains the number of elements equal to i.
7  for i = 1 to k
8      C[i] = C[i] + C[i - 1]
9    // C[i] now contains the number of elements less than or equal to i.
10   for j = A.length downto 1
12   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)
1   let C[0..k] be a new array
2   for i = 0 to k
3       C[i] = 0
4   for j = 1 to A.length
5       C[A[j]] = C[A[j]] + 1
6   // C[i] now contains the number of elements equal to i.
7   for i = 1 to k
8       C[i] = C[i] + C[i - 1]
9   // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
12     C[A[j]] = C[A[j]] - 1
```

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?

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>329</td>
<td>720</td>
<td>720</td>
<td>329</td>
</tr>
<tr>
<td>457</td>
<td>355</td>
<td>329</td>
<td>355</td>
</tr>
<tr>
<td>657</td>
<td>436</td>
<td>436</td>
<td>436</td>
</tr>
<tr>
<td>839</td>
<td>457</td>
<td>839</td>
<td>457</td>
</tr>
<tr>
<td>436</td>
<td>657</td>
<td>355</td>
<td>657</td>
</tr>
<tr>
<td>720</td>
<td>329</td>
<td>457</td>
<td>720</td>
</tr>
<tr>
<td>355</td>
<td>839</td>
<td>657</td>
<td>839</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>345</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>345</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>654</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>654</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>924</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>924</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>567</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>567</td>
</tr>
<tr>
<td>472</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>472</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>555</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>555</td>
<td></td>
<td></td>
</tr>
<tr>
<td>808</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>808</td>
</tr>
<tr>
<td>911</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>911</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>911</td>
<td></td>
<td></td>
<td>911</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>472</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>472</td>
</tr>
<tr>
<td>123</td>
<td></td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>654</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>654</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>924</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>924</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>345</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>345</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>555</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>555</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>567</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>567</td>
<td></td>
<td></td>
</tr>
<tr>
<td>808</td>
<td>808</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
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
  - \texttt{RADIX-SORT}(A, d)
    1. \texttt{for} $i = 1$ \texttt{to} $d$
    2. 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
  - \texttt{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)
1   let B[0..n - 1] be a new array
2   n = A.length
3   for i = 0 to n - 1
4       make B[i] an empty list
5   for i = 1 to n
6       insert A[i] into list B[⌊nA[i]⌋]
7   for i = 0 to n - 1
8       sort list B[i] with insertion sort
9   concatenate the lists B[0], B[1], \ldots, 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)**

1. let \( B[0..n-1] \) be a new array
2. \( n = A.length \)
3. for \( i = 0 \) to \( n-1 \)
4. make \( B[i] \) an empty list
5. for \( i = 1 \) to \( n \)
6. insert \( A[i] \) into list \( B[[nA[i]]] \)
7. for \( i = 0 \) to \( n-1 \)
8. sort list \( B[i] \) with insertion sort
9. concatenate the lists \( B[0], B[1], \ldots, B[n-1] \) together in order
For analyzing the average-case running time of bucket sort, we take the expectation over the input distribution

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

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

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

Next, we define indicator random variables \( X_{ij} \)

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

\[ n_i = \sum_{j=1}^{n} X_{ij} \]
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 $P(X_{ij} = 1) = \frac{1}{n}$

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

$\therefore$ $X_{ij}$ and $X_{ik}$ are independent

$\therefore E[X_{ij}X_{ik}] = E[X_{ij}]E[X_{ik}]$

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

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

$$= \sum_{j=1}^{n} \frac{1}{n} + \sum_{j=1}^{n} \sum_{k=1 \& j \neq k}^{n} \frac{1}{n^2}$$

$$= n \times \frac{1}{n} + n \times (n - 1) \times \frac{1}{n^2}$$

$$= 2 - \frac{1}{n}$$

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Worst-case running time</th>
<th>Average-case/expected running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion sort</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>Merge sort</td>
<td>$\Theta(n \lg n)$</td>
<td>$\Theta(n \lg n)$</td>
</tr>
<tr>
<td>Heapsort</td>
<td>$O(n \lg n)$</td>
<td>—</td>
</tr>
<tr>
<td>Quicksort</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n \lg n)$ (expected)</td>
</tr>
<tr>
<td>Counting sort</td>
<td>$\Theta(k + n)$</td>
<td>$\Theta(k + n)$</td>
</tr>
<tr>
<td>Radix sort</td>
<td>$\Theta(d(n + k))$</td>
<td>$\Theta(d(n + k))$</td>
</tr>
<tr>
<td>Bucket sort</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n)$ (average-case)</td>
</tr>
</tbody>
</table>
Conclusions

- [https://en.wikipedia.org/wiki/Best,_worst_and_average_case](https://en.wikipedia.org/wiki/Best,_worst_and_average_case)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Worst-case running time</th>
<th>Average-case/expected running time</th>
<th>Best-case running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion sort</td>
<td>Θ(n²)</td>
<td>Θ(n²)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Merge sort</td>
<td>Θ(n lg n)</td>
<td>Θ(n lg n)</td>
<td>Θ(n log₂ n)</td>
</tr>
<tr>
<td>Heapsort</td>
<td>O(n lg n)</td>
<td>O(n lg n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Quicksort</td>
<td>Θ(n²)</td>
<td>Θ(n lg n)</td>
<td>Θ(n log₂ n)</td>
</tr>
<tr>
<td>Counting sort</td>
<td>Θ(k + n)</td>
<td>Θ(k + n)</td>
<td>Θ(k + n)</td>
</tr>
<tr>
<td>Radix sort</td>
<td>Θ(d(n + k))</td>
<td>Θ(d(n + k))</td>
<td>Θ(d(k + n))</td>
</tr>
<tr>
<td>Bucket sort</td>
<td>Θ(n²)</td>
<td>Θ(n)</td>
<td>Θ(n)</td>
</tr>
</tbody>
</table>
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