Data structures and Algorithms

SORTING ALGORITHMS

(Part III)

Nguyễn Ngọc Thảo
nnthao@fit.hcmus.edu.vn

Outline

- Radix sort
- Counting sort
- A comparison of sorting algorithms

Radix sort

Radix sort: Idea

- The algorithm forms groups of data items and combines them to sort a collection of data.
- It does not compare the array’s entries, which is different from previous sorting algorithms.

- The idea of radix sort is similar to sorting a hand of cards.
  - Arrange the cards by rank into 13 possible groups in the following order: 2, 3, . . . , 10, J, Q, K, A.
  - Further arrange them by suit into four possible groups in this order: clubs, diamonds, hearts, and spades.
  - When taken together, the groups result in a sorted hand of cards.

Radix sort for positive integers

- Form groups according to the right-most digits, and then combine the groups in numerical order (i.e, from 0 to 9).
- Again, form groups following the next-to-last digits, and then combine them.
- And so on.

Example: Radix sort on an array of integers

- Consider the following collection of three-digit positive integers.

$$329, 457, 627, 839, 436, 720, 355$$

- Radix sort first organizes the data according to the **rightmost** (least significant) digits.

$$(72\mathbf{0})\ (35\mathbf{5})\ (43\mathbf{6})\ (45\mathbf{7}\ 65\mathbf{7})\ (32\mathbf{9}\ 83\mathbf{9})$$

- Now combine the groups into one as follows.

$$720, 355, 436, 457, 657, 329, 839$$

- The integers in each group end with the same digit, and the groups are ordered by that digit.

Example: Radix sort on an array of integers

- Next, form groups following the **middle** digit of each number.

$$(7\mathbf{2}0)\ (3\mathbf{2}9)\ (4\mathbf{3}6\ 8\mathbf{3}9)\ (3\mathbf{5}5\ 4\mathbf{5}7\ 6\mathbf{5}7)$$

- Combine these groups into one group, again preserving the relative order of the items within each group.

$$720, 329, 436, 839, 355, 457, 657$$

- Repeat similar steps for the **first** digit of each number.

$$(\mathbf{3}29\ \mathbf{3}55)\ (\mathbf{4}36\ \mathbf{4}57)\ (\mathbf{6}57)\ (\mathbf{7}20)\ (\mathbf{8}39)$$

$$329, 355, 436, 457, 657, 720, 839$$

- The numbers in each group retain their relative order from the original list

Example: Radix sort on an array of strings

- Consider the following collection of three-letter strings

$$ABC, XYZ, BWZ, AAC, RLT, JBX, RDT, KLT, AEO, TLJ$$

- Radix sort first organizes the data according to the **rightmost** (least significant) letters.

$$(AB\boldsymbol{C}, AA\boldsymbol{C})\ (TL\boldsymbol{J})\ (AE\boldsymbol{O})\ (RL\boldsymbol{T}, RD\boldsymbol{T}, KL\boldsymbol{T})\ (JB\boldsymbol{X})\ (XY\boldsymbol{Z}, BW\boldsymbol{Z})$$

- Now combine the groups into one as follows.

$$ABC, AAC, TLJ, AEO, RLT, RDT, KLT, JBX, XYZ, BWZ$$

Example: Radix sort on an array of strings

- Next, form groups following the **middle** letter of each string

$$(A\boldsymbol{A}C)\ (A\boldsymbol{B}C, J\boldsymbol{B}X)\ (R\boldsymbol{D}T)\ (A\boldsymbol{E}O)\ (T\boldsymbol{L}J, R\boldsymbol{L}T, K\boldsymbol{L}T)\ (B\boldsymbol{W}Z)\ (X\boldsymbol{Y}Z)$$

- Combine these groups into one group, again preserving the relative order of the items within each group

$$AAC, ABC, JBX, RDT, AEO, TLJ, RLT, KLT, BWZ, XYZ$$

- Repeat similar steps for the **first** letter of each string

$$(\boldsymbol{A}AC, \boldsymbol{A}BC, \boldsymbol{A}EO)\ (\boldsymbol{B}WZ)\ (\boldsymbol{J}BX)\ (\boldsymbol{K}LT)\ (\boldsymbol{R}DT, \boldsymbol{R}LT)\ (\boldsymbol{T}LJ)\ (\boldsymbol{X}YZ)$$

$$AAC, ABC, AEO, BWZ, JBX, KLT, RDT, RLT, TLJ, XYZ$$

Example: Another example of radix sort for strings

mom, dad, god, fat, bad, cat, mad, pat, bar, him	Original strings
(da**d**, go**d**, ba**d**, ma**d**) (mo**m**, hi**m**) (ba**r**) (fa**t**, ca**t**, pa**t**)	Grouped by the third characters
dad, god, bad, mad, mom, him, bar, fat, cat, pat	Combined
(d**a**d, b**a**d, m**a**d, b**a**r, f**a**t, c**a**t, p**a**t) (h**i**m) (g**o**d, m**o**m)	Grouped by the second characters
dad, bad, mad, bar, fat, cat, pat, him, god, mom	Combined
(**b**ad, **b**ar) (**c**at) (**d**ad) (**f**at) (**g**od) (**h**im) (**m**ad, **m**om) (**p**at)	Grouped by the first characters
bad, bar, cat, dad, fat, god, him, mad, mom, par	Combined (sorted)

Elements with varying lengths

- If the numbers have varying lengths, let them have the same length by padding on the left with zeros as necessary.
  - E.g., if the common length is 3, then the integer 10 becomes 010 and integer 2 turns into 002, while integer 123 does not change.
- Meanhile, if the character strings have varying lengths, pad them on the right with blanks.
  - E.g., if the common length is 3, then the string “a” becomes “a _ _”
- That common length is usually determined by the maximum lengths of all elements in the list.

Radix sort: Implementation

```
void radixSort(int arr[], int n){
  int max_val = arr[0];          // get maximum value in the array
  for (int i = 1; i < n; ++i)
    if (arr[i] > max_val) max_val = arr[i];

  int digits = 0, div;           // find the maximum number of digits
  do{
    digits++;
    div = max_val / pow(10, digits);
  } while (div > 0);

  int *tempArr[10];              // declare variables for temp store
  for (int i = 0; i < 10; ++i)
    tempArr[i] = new int[n];
  int tempCount[10];
  ......
}
```

Radix sort: Implementation

```
void radixSort(int arr[], int n){
    ......
    // form groups of numbers and combine groups
    for (int i = 0; i < digits ; ++i) {
        int exp = pow(10, i);
        for (int j = 0; j < 10; ++j)              // reset the counting
            tempCount[j] = 0;

        for (int j = 0; j < n; ++j) {             // form groups
             int idx = (arr[j] / exp) % 10;
            tempArr[idx][tempCount[idx]++] = arr[j];
        }

        int idx = 0;                              // combine groups
        for (int j = 0; j < 10; ++j)
            for (int k = 0; k < tempCount[j]; ++k)
                arr[idx++] = tempArr[j][k];
    }
}
```

Radix sort: An analysis

- It takes $2 \times n \times d$ moves to sort n strings, each of which has d characters.
  - n moves for forming groups and n moves for combining them into one group. The algorithm performs these $2 \times n$ moves d times.
- No comparisons are necessary.
- Thus, radix sort is $\mathbf{O}(\boldsymbol{n})$.
- It is not appropriate as a general-purpose sorting algorithm.
  - Substantial demand of memory if using arrays $\rightarrow$ use chain instead

Checkpoint 08: Radix sort on an array

Trace the **radix sort** as it sorts the following array into **ascending order**: {3812, 1600, 4012, 3934, 1234, 2724, 3333, 5432}.

Radix sort

Counting sort: Idea

- For each element, its final position in the sorted array is indicated by the total number of smaller elements.
- The sorted order is determined based on counting.
- This works in a situation that elements to be sorted belong to a known small set of values:

$$\forall i \in [1, n]\colon a_i \in \mathbb{N} \land a_i \in [l.u]$$

- For simplicity, let $l = 0$

Counting sort: Algorithm

- Compute the frequency of each of those values and store them in array

```
f[0..u] = 0;
for (i = 1; i ≤ n; i++)
   f[a[i]] ++;
```

- Distribution counting: add up sums of frequencies

```
for (i = 1; i ≤ u; i++)
   f[i] = f[i - 1] + f[i]
```

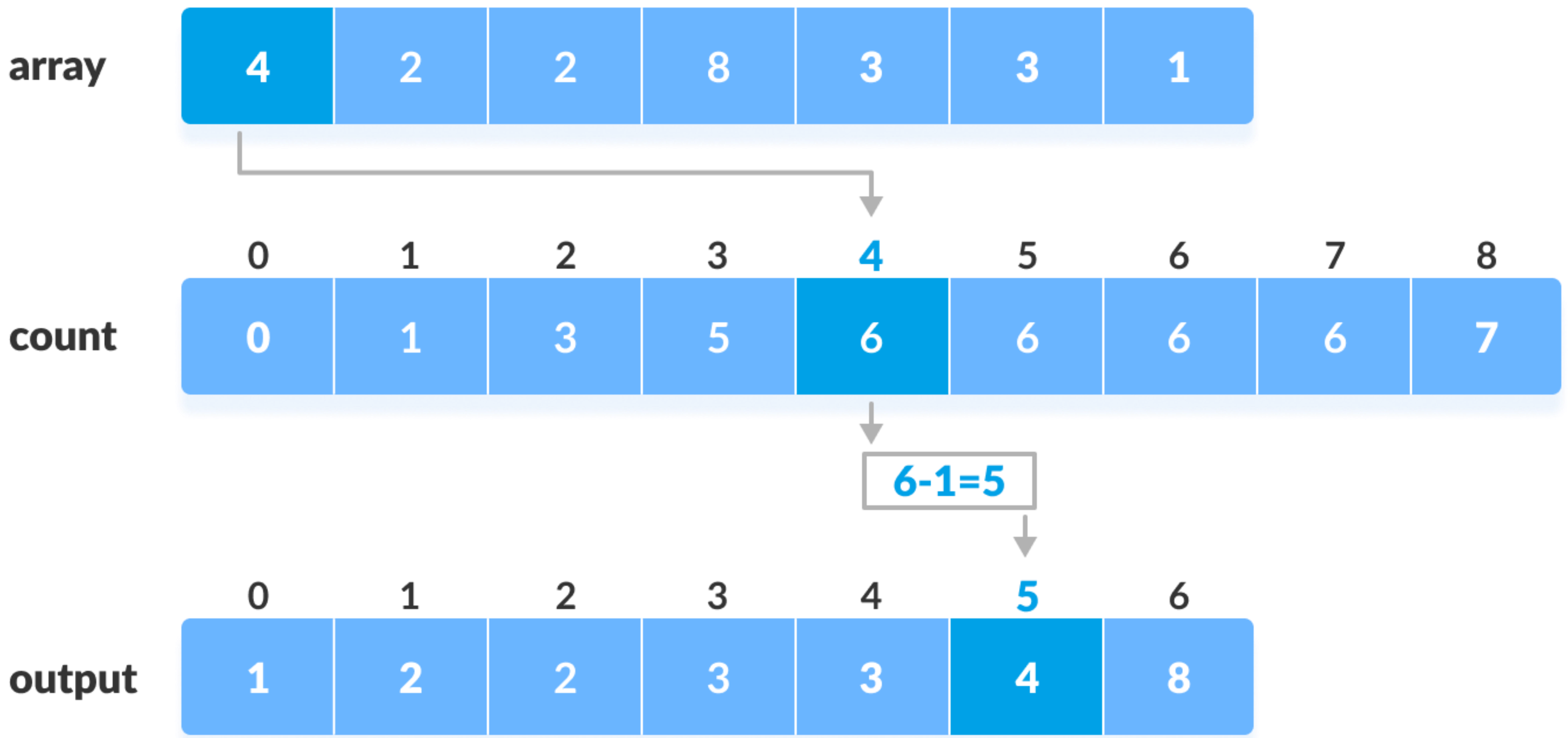
- The distribution values decide the proper positions for the last occurrences of their elements in the sorted array.
  - The array should be visited from right to left.

```
for (i = n; i ≥ 1; i--) {
   b[f[a[i]]] = a[i];
   f[a[i]] --;
}
a[1..n] = b[1..n];
```

Counting sort: Implementation

```
void countingSort(int arr[], int n, int u){ // u is the maximum value
   int *f = new int[u+1] {0};      // perform distribution counting
   for (int i = 0; i < n; i++)
      f[arr[i]] ++;
   for (int i = 1; i <= u; i++)    // accumulate sum of frequencies
      f[i] = f[i - 1] + f[i];
   int *b = new int[n];     // distribute values to their final positions
   for (int i = n - 1; i >= 0; i--) {
      b[f[arr[i]]-1] = arr[i];
      f[arr[i]]--;
   }
   for (int i = 0; i < n; i++)
      arr[i] = b[i];
}
```

Example: Counting sort on an array of integers



Counting sort: An analysis

- Better efficiency yet high space complexity
  - Two consecutive passes are made through the input array.
- In practice, this algorithm should be used when $u \leq n$
  - It would be disaster if $u \gg n$
- Counting radix sort: an alternative implementation of radix sort that avoids using bins

Checkpoint 09: Counting sort on an array

Trace the **counting sort** as it sorts the following array into **ascending order**: {1, 4, 1, 2, 7, 5, 2}.

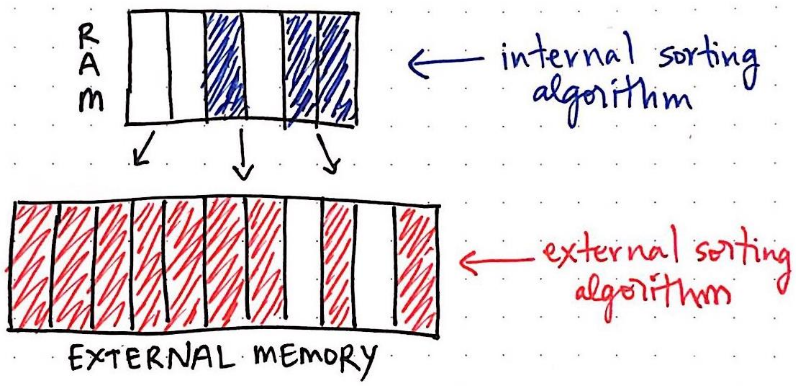
Radix sort

Time complexities: A summary

Algorithm	Time Complexity		
	Best	Average	Worst
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))
Quicksort	O(n log(n))	O(n log(n))	O(n^2)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))
Bubble Sort	O(n)	O(n^2)	O(n^2)
Insertion Sort	O(n)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Radix Sort	O(nk)	O(nk)	O(nk)

Internal sort vs. External sort

- An internal sort requires that the collection of data fits entirely in the computer's main memory.



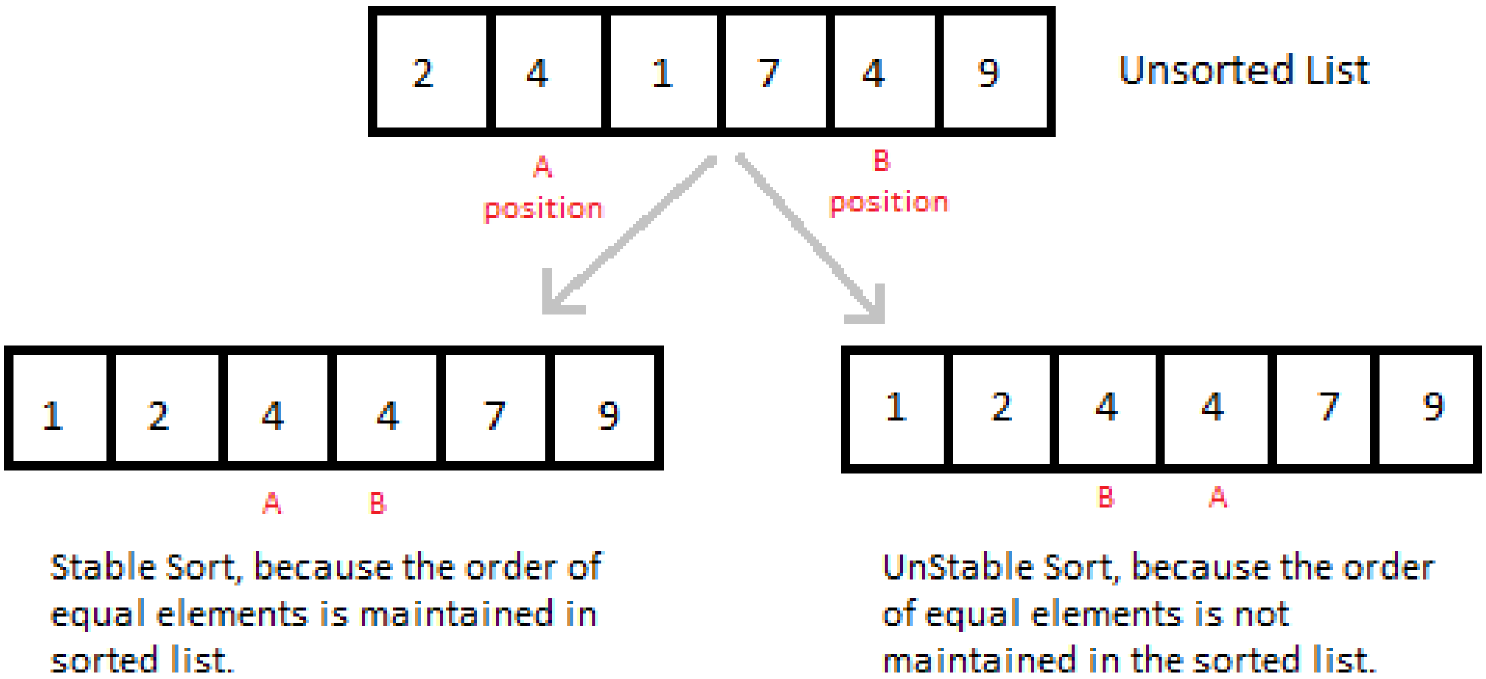
- An external sort is for situations that the collections of data do not fit in the computer's main memory all at once.
  - A part of data must reside in secondary storage, such as on a disk.

In-place sorting algorithms

- An in-place sorting algorithm does not require an extra space and produces an output in the same memory that contains the data by transforming the input “in-place”.
  - A small constant extra space can be used for variables is allowed.
- Which sorting algorithms are in-place?
  - Bubble sort, selection sort, insertion sort, heap sort
  - Not in-place: merge sort due to the need of O(n) extra space
- Quick sort is also called in-place.
  - There is extra space for recursive function calls, but this is not used to manipulate input.

Stability in sorting algorithms

- A stable sorting algorithm has any two objects with equal keys that appear in the same order in sorted output as they appear in the input array to be sorted.



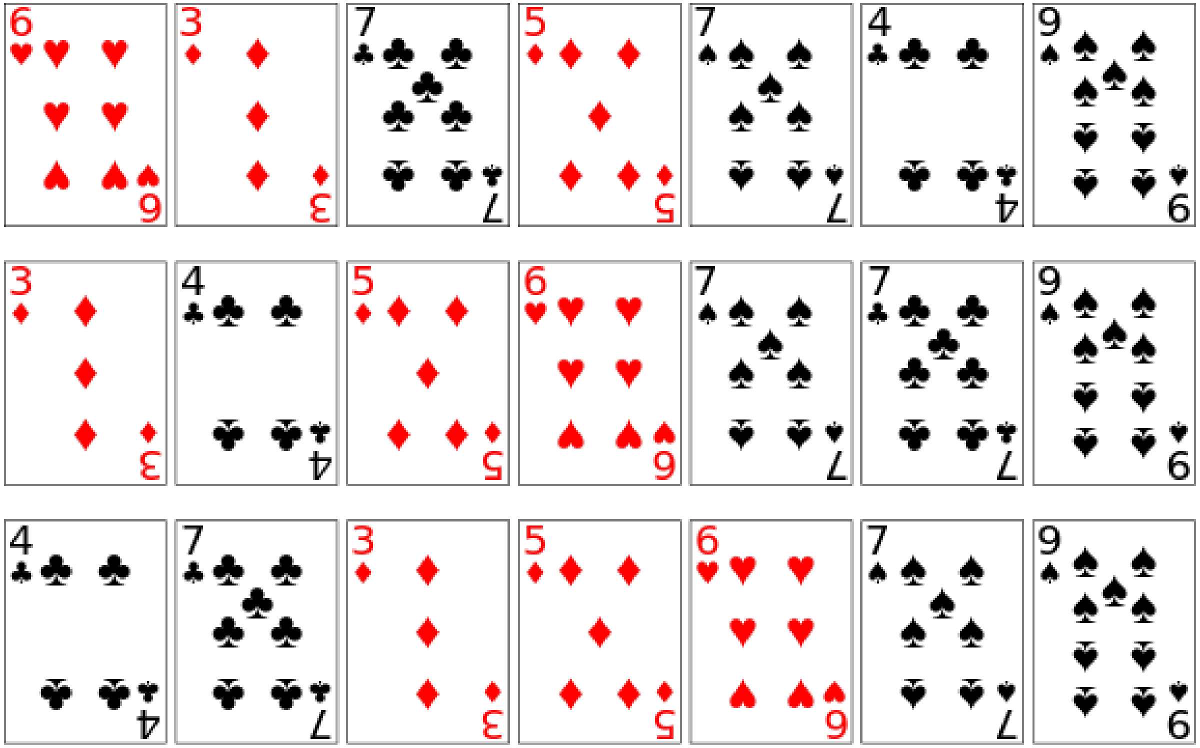
Data items to be sorted

- The data items to be sorted might be integers, character strings, or even objects.
- For objects witth several data members, identify the sort key, which is the member that determines the order of the entire object within the collection of data.

Sr. No.	Book Title ▼	Author ▼ ▲	Price ▼ ▲
1	Angels and Demons	Shivam	890
2	Harry Porter	Anuj	650
3	Hobbit	Aman	700
4	Lord of the rings	Sameer	1000
5	The little prince	Jatin	870

Stability in sorting algorithms

- One application for stable sorting algorithms is sorting a list using a primary and secondary key.
  - E.g., sort a hand of cards such that the suits are in the order {♣, ♦, ♥, ♠}, and within each suit, the cards are sorted by rank.



Acknowledgements

This part of the lecture is adapted from the following materials.

[1] Pr. Nguyen Thanh Phuong (2020) "*Lecture notes of CS163 – Data structures*" University of Science - Vietnam National University HCMC.

[2] Pr. Van Chi Nam (2019) "*Lecture notes of CSC14004 – Data structures and algorithms*" University of Science - Vietnam National University HCMC.

[3] Frank M. Carrano, Robert Veroff, Paul Helman (2014) "*Data Abstraction and Problem Solving with C++: Walls and Mirrors*" Sixth Edition, Addion-Wesley. **Chapter 10.**

[4] Anany Levitin (2012) "*Introduction to the Design and Analysis of Algorithms*" Third Edition, Pearson.

Exercises

01. Sorting algorithms on an array

- Apply radix sort on the following array of integers, {170, 45, 75, 90, 802, 24, 2, 66}, to re-arrange the array in ascending order.
- Apply counting sort on the following array of integers, {1, 5, 2, 7, 3, 4, 4, 1, 5}, to re-arrange the array in ascending order.

02. Which algorithm is it?

- Please provide an example of each of the following (or say NONE if no example exists) from among the sorting algorithms we studied.
  a) A stable comparison sort with a worst-case $O(n^2)$ running time
  b) A stable comparison sort with a worst-case $O(n)$ running time
  c) An efficient (i.e., $O(n \log_2 n)$) stable comparison sort
  d) A stable sort with a worst-case running time linear in n