Problem Solving Techniques 문제해결

Jinkyu Lee

Dept. of Computer Science and Engineering,
Sungkyunkwan University (SKKU)

Contents

- **Chapter 4. Sorting**

Time Complexity: big O notation

- $f(x) = O(g(x))$
  - When $\lim_{x \to \infty} | f(x)/g(x) | < \infty$
- $N^3 + 100000000*N^2 + N + 10000000000000$
  - $O(N^3)$
- $1000000\ N^3 + 100000000*N^2 + N + 10000000000000$
  - $O(N^3)$
- Bubble sort (worst-case)
  - $O(N^2)$
- Merge sort (worst-case)
  - $O(N \log N)$

Example

Matrix Multiplication

- Matrices A and B are multiplied; **C = AB**
- What is its (computational) complexity?
- Any efficient matrix multiplication method?

Complexity?
$O(n^3)$

$$A = \begin{bmatrix} \mathbf{A_1} & \mathbf{A_2} \\ \mathbf{A_3} & \mathbf{A_4} \end{bmatrix} \quad B = \begin{bmatrix} \mathbf{B_1} & \mathbf{B_2} \\ \mathbf{B_3} & \mathbf{B_4} \end{bmatrix}$$

$$C = \begin{bmatrix} \sum_{k=1}^{n} a_{1k} b_{k1} & \cdots & \sum_{k=1}^{n} a_{1k} b_{kn} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} a_{nk} b_{k1} & \cdots & \sum_{k=1}^{n} a_{nk} b_{kn} \end{bmatrix}$$

$$C = \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \times \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}$$

Complexity?
$O(n^3)$

$$C_1 = A_1B_1 + A_2B_3; \quad C_2 = A_1B_2 + A_2B_4$$
$$C_3 = A_3B_1 + A_4B_3; \quad C_4 = A_3B_2 + A_4B_4$$

$$P_1 = A_1(B_2 - B_4); P_2 = (A_1 + A_2)B_4;$$
$$P_3 = (A_3 + A_4)B_1; P_4 = A_4(-B_1 + B_3);$$
$$\cdots\cdots \; ; P_7 = (-A_1 + A_3)(B_1 + B_2);$$

Complexity?
$O(n^{\log 7 = 2.81})$

$$C_1 = -P_2 + P_4 + P_5 + P_6; C_2 = P_1 + P_2$$
$$C_3 = P_3 + P_4; \quad C_4 = P_1 - P_3 + P_5 + P_7$$

Example

- **Matrix Multiplication**

$T(n) = 7 * T(n/2)$
$T(1) = 1$

  - Matrices A and B are multiplied; **C = AB**
  - What is its (computational) complexity?
  - Any efficient matrix multiplication method?

Complexity?
$O(n^3)$

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \quad B = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}$$

$$C = \begin{bmatrix} \sum_{k=1}^{n} a_{1k} b_{k1} & \cdots & \sum_{k=1}^{n} a_{1k} b_{kn} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} a_{nk} b_{k1} & \cdots & \sum_{k=1}^{n} a_{nk} b_{kn} \end{bmatrix}$$

$$C = \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \times \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}$$

Complexity?
$O(n^3)$

$$C_1 = A_1 B_1 + A_2 B_3; \quad C_2 = A_1 B_2 + A_2 B_4$$
$$C_3 = A_3 B_1 + A_4 B_3; \quad C_4 = A_3 B_2 + A_4 B_4$$

Complexity?
$O(n^{\log 7 = 2.81})$

$$P_1 = A_1(B_2 - B_4); P_2 = (A_1 + A_2)B_4;$$
$$P_3 = (A_3 + A_4)B_1; P_4 = A_4(-B_1 + B_3);$$
$$\ldots\ldots \; ; P_7 = (-A_1 + A_3)(B_1 + B_2);$$

$$C_1 = -P_2 + P_4 + P_5 + P_6; C_2 = P_1 + P_2$$
$$C_3 = P_3 + P_4; \quad C_4 = P_1 - P_3 + P_5 + P_7$$

Sorting Algorithms

https://www.toptal.com/developers/sorting-algorithms

- Bubble sort
- Selection sort
- Insertion sort
- Shell sort
- Merge sort
- Quick sort
- Bitmap sort
- Counting sort
- Radix sort

Bubble Sort

- Bubble sort
  - is a simple sorting algorithm that repeatedly steps through the list to be sorted, compare each pair of adjacent items and swaps them if they are in the wrong order. (Wikipedia)

  - http://en.wikipedia.org/wiki/Bubble_sort
  - https://visualgo.net/en/sorting
    - The first item

  - Worst-case: $O(n^2)$
  - Average-case: $O(n^2)$

Selection Sort

- Selection sort
  - divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front (left) of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. (Wikipedia)
  - http://en.wikipedia.org/wiki/Selection_sort
  - https://visualgo.net/en/sorting
    - The second item
  - Worst-case: $O(n^2)$
  - Average-case: $O(n^2)$

Insertion Sort

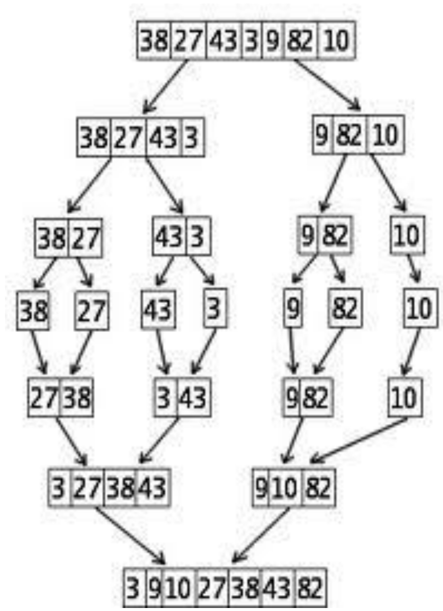
- Insertion sort
  - iterates, consuming one input element each repetition, and grows a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain. (Wikipedia)
  - https://en.wikipedia.org/wiki/Insertion_sort
  - https://visualgo.net/en/sorting
    - The third item
  - Worst-case: $O(n^2)$
  - Average-case: $O(n^2)$

Merge Sort

- Merge sort
  - Divides the unsorted list into n sublists, each containing 1 element, and repeatedly merges sublists to produce new sorted sublists until there is only 1 sublist remaining. (Wikipedia)

  - http://en.wikipedia.org/wiki/Merge_sort
  - https://visualgo.net/en/sorting
    - The fourth item

  - Worst-case: O(n*log n)
  - Average-case: O(n*log n)



Quick Sort

- Quick sort
  - Quicksort is a divide and conquer algorithm. It first divides the input array into two smaller sub-arrays: the low elements and the high elements. It then recursively sorts the sub-arrays. (Wikipedia)

  - https://en.wikipedia.org/wiki/Quicksort
  - https://visualgo.net/en/sorting
    - The fifth item

  - Worst-case: $O(n^2)$
  - Average-case: $O(n*\log n)$

Bitmap Sort

- Problem statement
  - Sort a series of numbers in O(n) time. Print out the result.
  - Assumptions
    - The numbers are positive integers.
    - Each number appears only once.
    - There is a limitation on the size of memory.

Bitmap Sort

- http://letsalgorithm.blogspot.com/2012/02/bitmap-sort.html

- This algorithm uses a bitmap (or a bit vector) to represent a finite set of distinct integers. For example, if we have an integer range 0-5, we can represent it using a 6-bit array, e.g.
  - [2,5,3] becomes 0 0 1 1 0 1
  - [4,3,1] becomes 0 1 0 1 1 0

Bitmap Sort

- Data structure
  - Byte array
  - Details
    - We will store the number n at the nth position in a byte array.
    - Positions should be calculated considering the number of bits.
      - It defers on the data type for the byte array.
    - To access nth position in O(1) time, using byte array is the best candidate.
  - The type for the byte array
    - Any types for integers can be used
      - `int`, `unsigned int`, `char` and etc.
    - Why? Why not float or double?
  - Which one will be a better choice?

Bitmap Sort

- Algorithm
  1. On each number n, nth bit is set to one on a byte array.
  2. Print out position numbers of set bit by traversing the array.
     - Why position numbers? Where are the actual input number?

- Complexity
  - O(n) for input, O(m) for output → O(m)
    - n: the number of elements to be sorted
    - m: the number of choices

- We need two operations for bitmap sorting.
  1. void SetBit(byte array, position)
     - Set a bit as 1 on each number.
  2. boolean CheckBit(byte array, position)
     - Return true only when the position bit is set.
     - Otherwise return false.

Bitmap Sort

- Implementing Bitmap Sorting
  - Short Implementation
    - The byte array is declared as a global variable for simplicity.

```
void SetBit (int n)
{
  bitmap[n / 8] |= 1 << n % 8;
}

int CheckBit (int n)
{
  return (bitmap[n / 8] >> (n % 8)) & 1;
}
```

Bitmap Sorting

- Discussion
  - It seems that this sorting method is very efficient.
  - Any disadvantage?

    - Assumptions
      - The numbers are positive integers.
      - Each number appears only once.
      - There is a limitation on the size of memory.

Counting Sort

- Problem statement
  - Sort a series of numbers in linear time. Print out the result.

  - Assumptions
    - The numbers are positive integers.
    - ~~Each number appears only once.~~
    - ~~There is a limitation on the size of memory.~~
    - The numbers are in a limited range and not sparsely distributed.

Counting Sort

- Data Structure
  - Integer array
  - Details
    - We will use the array as a count array and store the count for each number.
    - Input numbers will be used as indexes for accessing the count array.
      - `arr[num]++;`
    - To increase the count in O(1) time, using arrays is the best candidate.
    - The count will be used for printing out the result. (See algorithm.)

Jinkyu Lee
Dept. of Computer Science and Engineering

Counting Sort

- Example
  - Input
    - 20 integers in the range of [0, 9]
    - Ex: 2, 8, 2, 9, 2, 3, 3, 1, 9, 1, 8, 2, 5, 7, 4, 3, 5, 8, 2, 7
  - Count Array

Number	0	1	2	3	4	5	6	7	8	9
Count	0	2	5	3	1	2	0	2	3	2

  - Printing
    - ?

Counting Sort

- Algorithm
  1. On each number n, increase the count at index n on the count array by one.
  2. Traverse the count array and print out each index the count number of times.
     - What should we do if the sorted number are to be stored?

- Complexity
  - n: the number of input, m: the range of inputs (the number input choices)
  - O(n) for input, O(m) for output → O(n + m)

Counting Sort

- Discussion
  - What will be weak points of Counting Sort?
    - Assumptions
      - The numbers are positive integers.
      - The numbers are in a limited range and not sparsely distributed.