Design and Analysis of Algorithms
Unit -4

Bharathi R
Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Unit 4: Space and Time Tradeoffs

Space and Time Tradeoffs - Sorting by Counting

Bharathi R

Department of Computer Science & Engineering

Space and Time Tradeoffs- Introduction

- Space and time trade-offs in algorithm design are a well-known issue for both theoreticians and practitioners of computing.

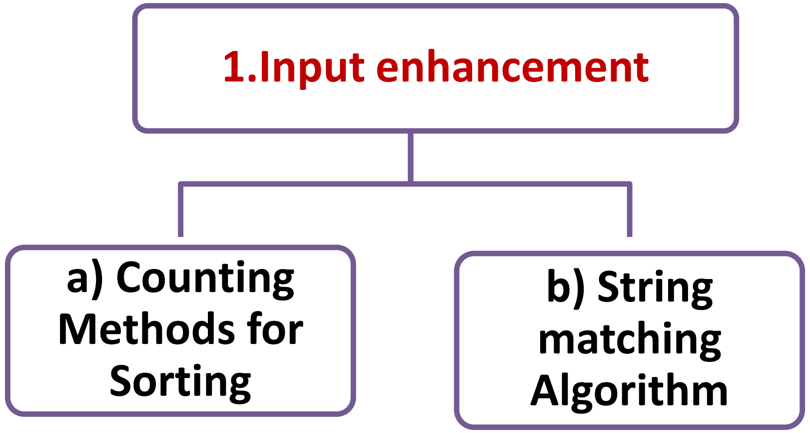
- As an algorithm design technique, trading space for time is much more prevalent than trading time for space.

Principal Varieties for trading space for time in Algorithm design

- Achieving time efficiency in computation
  - 1.Input enhancement
    - a) Counting Methods for Sorting
    - b) String matching Algorithm
  - 2.Prestructuring
    - a) Hashing
    - b) B-Trees
  - 3.Dynamic programming

1.Input enhancement

- **Input Enhancement**
  - Preprocess the problem's input, in whole or in part, and store the additional information obtained to accelerate solving the problem afterward.
  - Eg:
    1. Comparison counting sort
    2. Distribution Counting,
    3. Horspool's algorithm,
    4. Boyer-Moore's algorithm



1.Input enhancement

a)Sorting by Counting

1. **Comparison Counting Sorting**
   I. For each element of the list, count the total number of elements smaller than this element.
   II. These numbers will indicate the positions of the elements in the sorted list.

2. **Distribution Counting Sorting**
   I. Suppose the elements of the list to be sorted belong to a finite set (aka domain).
   II. Count the frequency of each element of the set in the list to be sorted.
   III. Scan the set in order of sorting and print each element of the set according to its frequency, which will be the required sorted list.

1.Input Enhancement

→ Sorting by Counting

→→ Comparison Counting Sorting

1. Find the numbers that are less than a[0] i.e, 62, by scanning the array from the index 1 to 5

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
Array *a*	62	31	84	96	19	47

2. Maintain another array called ***Count*** the elements that are **lesser than 62**

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
Array ***a***	62	31	84	96	19	47
Array ***Count***	**3**					

Example of sorting by comparison counting

Array $A[0..5]$		62	31	84	96	19	47
Initially	*Count* []	0	0	0	0	0	0
After pass $i = 0$	*Count* []	3	0	1	1	0	0
After pass $i = 1$	*Count* []		1	2	2	0	1
After pass $i = 2$	*Count* []			4	3	0	1
After pass $i = 3$	*Count* []				5	0	1
After pass $i = 4$	*Count* []					0	2
Final state	*Count* []	3	1	4	5	0	2
Array $S[0..5]$		19	31	47	62	84	96

Algorithm for Sorting by Counting

```
ALGORITHM  ComparisonCountingSort(A[0..n − 1])
    //Sorts an array by comparison counting
    //Input: An array A[0..n − 1] of orderable elements
    //Output: Array S[0..n − 1] of A's elements sorted
    for i ← 0 to n − 1 do Count[i] ← 0
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            if A[i] < A[j]
                Count[j] ← Count[j] + 1
            else Count[i] ← Count[i] + 1
    for i ← 0 to n − 1 do S[Count[i]] ← A[i]
    return S
```

It should be quadratic because the algorithm considers all the different pairs of an *n*-element array

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i) = \frac{n(n-1)}{2}$$

Drawbacks

1. Thus, the algorithm makes the same number of key comparisons as selection sort,
2. In addition it uses a linear amount of extra space.

References

“Introduction to the Design and Analysis of Algorithms”, Anany Levitin, Pearson Education, Delhi (Indian Version), 3rd edition, 2012.

Chapter- 7

THANK YOU

Bharathi R

Department of Computer Science & Engineering

rbharathi@pes.edu