

Design and Analysis of Algorithms

L08: Important Problem Types

Dr. Ram P Rustagi
Sem IV (2019-H1)
Dept of CSE, KSIT
rprustagi@ksit.edu.in

Resources

- Text book I: Levitin
- <http://interactivepython.org/courselib/static/pythonds/SortSearch/TheMergeSort.html>
- <http://interactivepython.org/courselib/static/pythonds/SortSearch/TheInsertionSort.html>
-

Important Problem Types

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

Sorting

- Rearrange the items of a given list in ascending (or descending) order.
 - Input: A sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$
 - Output: A reordering $\langle a'_1, a'_2, \dots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.
- Why sorting?
 - Helps searching
 - Algorithms often use sorting as a key subroutine.
- Sorting key
 - A specially chosen piece of information used to guide sorting. E.g., sort student records by names.
 - Key should support comparison operation

Sorting

- Examples of sorting algorithms
 - Selection sort
 - Bubble sort
 - Insertion sort
 - Merge sort
 - Heap sort ...
- Algorithm complexity: the number of key comparisons.
 - Some algos take more memory, but less time
 - Others take more time, but less memory
- Properties of sorting algorithms
 - Stability: When algorithm preserves the relative order of any two equal elements in its input.
 - In place : When algorithm does not require extra memory, except, possibly for a few memory units.

Selection Sort

Algorithm *SelectionSort*($A[0..n-1]$)

//The algorithm sorts a given array by selection sort

//Input: An array $A[0..n-1]$ of orderable elements

//Output: Array $A[0..n-1]$ sorted in ascending order

for $i \leftarrow 0$ to $n-2$ do

$\text{min} \leftarrow i$

 for $j \leftarrow i+1$ to $n-1$ do

 if $A[j] < A[\text{min}]$

$\text{min} \leftarrow j$

 swap $A[i]$ and $A[\text{min}]$

Demo: Selection Sort program

Complexity: $O(n^2)$

Bubble Sort

Algorithm *SelectionSort*($A[0..n-1]$)

//The algorithm sorts a given array by bubble sort

//Input: An array $A[0..n-1]$ of orderable elements

//Output: Array $A[0..n-1]$ sorted in ascending order

```
for  $i \leftarrow 0$  to  $n-2$  do
    for  $j \leftarrow i+1$  to  $n-1$  do
        if  $A[j] < A[i]$ 
            swap  $A[j]$  and  $A[i]$ 
```

Demo: Bubble Sort program

Complexity: $O(n^2)$

Insertion Sort

- Starts with a sorted sublist (initially 1 element)
- Increase this sorted sublist, one item at a time.
- Time complexity: $O(n^2)$

Algorithm *InsertionSort*($A[0..n-1]$)

//Input: An array $A[0..n-1]$ of orderable elements

//Output: Array $A[0..n-1]$ sorted in ascending order

for $i \leftarrow 1$ to $n-1$ do

$curr \leftarrow A[i]$

$pos \leftarrow i$

 while $pos > 0$ and $A[pos-1] > curr$ do

$A[pos] \leftarrow A[pos-1]$

$pos \leftarrow pos-1$

$A[pos] \leftarrow curr$

Merge Sort

- A recursive algorithm
- Continuously splits the list in half
- When the list is empty or size 1, it is sorted.
- Combine the two sorted list to create a single list
 - Called Merge operation.

Searching

- Find a given value, called a **search key**, in a given set.
- Examples of searching algorithms
 - Sequential search
 - Binary search ...
 - Hash search
 -

String Processing

- A string is a sequence of characters from an alphabet.
- Text strings: letters, numbers, and special characters.
- String matching:
 - Searching for a given word/pattern in a text.
- Use it in daily applications
 - Finding key words in text files, programs etc.

Graph Problems

- Informal definition
 - A graph is a collection of nodes called vertices
 - Nodes are connected by edges.
- Modeling real-life problems
 - Modeling world wide web
 - Communication networks
- Examples of graph algorithms
 - Graph traversal algorithms
 - Shortest-path algorithms
 - Topological sorting

Combinatorial Problems

- Find a combinatorial object (e.g. permutation, combination)
 - Satisfying some constraints and
 - Satisfies a desired property (max value or min cost)
- Generally, the most difficult problem in CS
 - Num of combinatorial objects grows exponentially
 - No known algo exist to solve in acceptable time.
- Examples
 - Travelling Salesman problem
 - Graph coloring problem
 - Time table generation

Geometric Problems

- Deals with geometric objects
 - Points, lines, polygons...
- Example problems
- Closest pair problem
 - Given n points in a plane, find the closest pair among them.
- Convex Hull problem
 - Find the smallest convex polygon that would include all points of a given set

Numerical Problems

- Problems involving mathematical objects
- Solving equations, function evaluations, ..
 - Problems can be solved approximately
 - These problems requires real number representation
 - Can be represented only approximately
 - Accumulation of round off errors can lead to errors
 -

Summary

- Problems important types
 - Sorting
 - Searching
 - String processing
 - Graph problems
 - Combinatorial problems
 - Geometric problems
 - Numerical problems