Transform and Conquer

This group of techniques solves a problem by a transformation to

- a simpler/more convenient instance of the same problem (*instance simplification*)
- a different representation of the same instance (*representation change*)
- a different problem for which an algorithm is already available (*problem reduction*)

checking if all elements are distinct
computing the median
Instance simplification - Presorting

Solve a problem’s instance by transforming it into another simpler/easier instance of the same problem.

Presorting

Many problems involving lists are easier when list is sorted, e.g.

- searching
- computing the median (selection problem)
- checking if all elements are distinct (element uniqueness)

Also:

- Topological sorting helps solving some problems for dags.
- Presorting is used in many geometric algorithms.
How fast can we sort?

Efficiency of algorithms involving sorting depends on efficiency of sorting.

**Theorem** (see Sec. 11.2): \([\log_2 n!] \approx n \log_2 n\) comparisons are necessary in the worst case to sort a list of size \(n\) by any comparison-based algorithm.

Note: About \(n \log_2 n\) comparisons are also sufficient to sort array of size \(n\) (by mergesort).
Searching with presorting

Problem: Search for a given \( K \) in \( A[0..n-1] \)

Presorting-based algorithm:

Stage 1  Sort the array by an efficient sorting algorithm

Stage 2  Apply binary search

Efficiency: \( \Theta(n\log n) + O(\log n) = \Theta(n\log n) \)

Good or bad?

Why do we have our dictionaries, telephone directories, etc. sorted?
Element Uniqueness with presorting

- Presorting-based algorithm
  Stage 1: sort by efficient sorting algorithm (e.g. mergesort)
  Stage 2: scan array to check pairs of adjacent elements

Efficiency: $\Theta(n\log n) + O(n) = \Theta(n\log n)$

- Brute force algorithm
  Compare all pairs of elements

Efficiency: $O(n^2)$

Another algorithm? Hashing
Instance simplification – Gaussian Elimination

Given: A system of $n$ linear equations in $n$ unknowns with an arbitrary coefficient matrix.

Transform to: An equivalent system of $n$ linear equations in $n$ unknowns with an upper triangular coefficient matrix.

Solve the latter by substitutions starting with the last equation and moving up to the first one.

\[
\begin{align*}
    a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\
    a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\
    \vdots \\
    a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n \\
\end{align*}
\]

\[
\begin{align*}
    a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\
    a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\
    \vdots \\
    a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n \\
    a_{nn}x_n &= b_n
\end{align*}
\]

Gaussian Elimination (cont.)

The transformation is accomplished by a sequence of elementary operations on the system's coefficient matrix (which don't change the system's solution):

for $i \leftarrow 1$ to $n-1$ do
    replace each of the subsequent rows (i.e., rows $i+1, \ldots, n$) by a difference between that row and an appropriate multiple of the $i$-th row to make the new coefficient in the $i$-th column of that row 0
Example of Gaussian Elimination

Solve:
\[
\begin{align*}
2x_1 - 4x_2 + x_3 &= 6 \\
3x_1 - x_2 + x_3 &= 11 \\
x_1 + x_2 - x_3 &= -3
\end{align*}
\]

Gaussian elimination:

\[
\begin{align*}
2 &\quad -4 &\quad 1 &\quad 6 \\
3 &\quad -1 &\quad 1 &\quad 11 \\
1 &\quad 1 &\quad -1 &\quad -3
\end{align*}
\]

\[
\begin{align*}
\text{row2} &\quad - (3/2) \times \text{row1} \\
0 &\quad 5 &\quad -1/2 &\quad 2 \\
\text{row3} &\quad - (1/2) \times \text{row1} \\
0 &\quad 3 &\quad -3/2 &\quad -6
\end{align*}
\]

\[
\begin{align*}
2 &\quad -4 &\quad 1 &\quad 6 \\
0 &\quad 5 &\quad -1/2 &\quad 2 \\
0 &\quad 0 &\quad -6/5 &\quad -36/5 \\
\text{row3} &\quad - (3/5) \times \text{row2}
\end{align*}
\]

Backward substitution:

\[
x_3 = (-36/5) / (-6/5) = 6 \\
x_2 = (2 + (1/2) \times 6) / 5 = 1 \\
x_1 = (6 - 6 + 4 \times 1) / 2 = 2
\]
Pseudocode and Efficiency of Gaussian Elimination

Stage 1: Reduction to an upper-triangular matrix
for \(i \leftarrow 1\) to \(n-1\) do
  for \(j \leftarrow i+1\) to \(n\) do
    for \(k \leftarrow i\) to \(n+1\) do

Stage 2: Back substitutions
for \(j \leftarrow n\) downto \(1\) do
  \(t \leftarrow 0\)
  for \(k \leftarrow j +1\) to \(n\) do
    \(t \leftarrow t + A[j, k] \times x[k]\)
    \(x[j] \leftarrow (A[j, n+1] - t) / A[j, j]\)

Efficiency: \(\Theta(n^3) + \Theta(n^2) = \Theta(n^3)\)
Searching Problem

Problem: Given a (multi)set $S$ of keys and a search key $K$, find an occurrence of $K$ in $S$, if any

- Searching must be considered in the context of:
  - file size (internal vs. external)
  - dynamics of data (static vs. dynamic)

- Dictionary operations (dynamic data):
  - find (search)
  - insert
  - delete
Taxonomy of Searching Algorithms

- List searching
  - sequential search
  - binary search
  - interpolation search

- Tree searching
  - binary search tree
  - binary balanced trees: AVL trees, red-black trees
  - multiway balanced trees: 2-3 trees, 2-3-4 trees, B trees

- Hashing
  - open hashing (separate chaining)
  - closed hashing (open addressing)
Binary Search Tree

Arrange keys in a binary tree with the binary search tree property:

Example: 5, 3, 1, 10, 12, 7, 9
Dictionary Operations on Binary Search Trees

Searching – straightforward
Insertion – search for key, insert at leaf where search terminated
Deletion – 3 cases:
  deleting key at a leaf
  deleting key at node with single child
  deleting key at node with two children

Efficiency depends of the tree's height: \( \lfloor \log_2 n \rfloor \leq h \leq n-1 \),
with height average (random files) be about \( 3 \log_2 n \)

Thus all three operations have
- worst case efficiency: \( \Theta(n) \)
- average case efficiency: \( \Theta(\log n) \)

Bonus: inorder traversal produces sorted list