"Transform and Conquer"

Often useful, before solving a problem, to transform its instance

1. by simplifying it, or
2. into another representation, or
3. into an instance of another problem

3. is typical for considering the solvability or complexity (esp. NP-completeness) of problems: the target of the transformation is seen to be at least as difficult as the original problem

Sorting is often a useful simplifying transformation:

**Example** Removal of duplicates

Task: find/eliminate repeating values in sequence \((a_1, \ldots, a_n)\)

\[ B = \text{Sort}(a_1, \ldots, a_n); \]

Scan \(B\),

report/eliminate adjacent identical items;

It can be proved that comparison-based elimination of duplicates also requires time \(\Omega(n \log n)\)

Many algorithms of computational geometry also utilize sorting (of the set of points that form the instance)

**Example** (Multiplication via logarithms)

256 \times 4096 =

The Fourier transform, similarly, transforms convolution into inner product

\(\rightarrow\) efficient multiplication of polynomials

**Example** (Horner rule)

Evaluate polynomial

\[ P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \]

at a given point

\[ = (((a_n x + a_{n-1}) x + a_{n-2}) x + \ldots + a_1) x + a_0 \]

Directly \(\rightarrow 2n - 1 \times \text{MUL} \text{ and } n \times \text{ADD}\)

The second form \(\rightarrow n \times \text{MUL} \text{ and } n \times \text{ADD}\)
**2-3-4 trees**

Implementation of a balanced search tree:

All leaves are at the same depth;
Each node carries 1, 2, or 3 keys;
Inner node has # of keys + 1 child pointers;
Order btw keys \((x, y, z)\) and subtrees \(A, B, C, D\):
\(A \leq x < B \leq y < C \leq z < D\)

Nodes with 1 or 2 keys similarly

Nodes of a **B tree** have \(t - 1 \ldots 2t - 1\) keys;
2-3-4 tree is a B tree with \(t = 2\);
In disk indexes of databases \(t\) typically
200 \ldots 1000

**Search**: branch to the appropriate subtree

**Insertion**: Search the leaf; place key there.
While going down, eliminate full nodes:

(ii) & (iii): symmetric variants also

(i)\(\rightarrow\) the insert position is never full
(i) \(\Rightarrow\) tree gets deeper at the root only
(\(\rightarrow\) remains balanced)

Insert chars of "esimerkkime" into a 2-3-4 tree:

\[\text{Th.} \ 2-3-4 \text{ trees support lookup, insertion and deletion of keys in time } O(\log n)\]

**Proof**: All leaves are at the same depth \(h\)

Operations are local along root-to-node path \(\rightarrow\) their complexity is \(O(h)\)

\(n\) keys \(\rightarrow\) tree has at most \(n\) nodes

\(h \leq \log_2 n\) because each inner node has at least 2 children

\(\blacksquare\)