A Brief Review of Algorithm Design Concepts

1 Binary Search

Problem:
Given an array \( A(1..n) \) of \( n \) sorted numbers, and a key value \( x \), determine whether \( x \) is in \( A \) (output: 0 if answer is “no”, or \( j \) if \( x = A[j] \) i.e. answer is “yes”).

Solution:
procedure \( BS(A, a, b, x) \)
{
  if \( a > b \) then return 0;
  \( mid := \lfloor \frac{a+b}{2} \rfloor \);
  if \( x = A[mid] \) then return \( mid \)
  else
    if \( x < A[mid] \) then return \( BS(A, a, mid - 1, x) \) else return \( BS(A, mid + 1, b, x) \)
}

The initial call is \( BS(A, 1, n, x) \). This is an example of divide-and-conquer, where we recursively solve a subproblem of half size.

Time complexity: \( T(n) = 1 + T(\lfloor \frac{n}{2} \rfloor) \).
2 Tower of Hanoi

Problem:
Given 3 rods, $R_1$, $R_2$ and $R_3$, with $n$ rings on $R_1$, and none on $R_2$ and $R_3$. The rings on $R_1$ are stacked in order of size with the smallest on top. The permitted operations are: move the top ring on any rod to any other rod provided that at no time a larger ring is on top of a smaller ring. The goal is to move the rings from $R_1$ to $R_2$ with minimum number of moves.

Solution:
procedure $H(n, \text{FromRod, ToRod, SpareRod})$
{
    if $n > 0$ then {
        (a) $H(n - 1, \text{FromRod, SpareRod, ToRod})$;
        (b) move the top ring of FromRod to ToRod;
        (c) $H(n - 1, \text{SpareRod, ToRod, FromRod})$
    } 
}

Figure 1 shows an example of the recursion of this algorithm.

Figure 1: Recursion of Hanoi procedure for $n = 3$. The states of rods after step (b) at all recursive levels.

Technique used: divide-and-conquer. Total effort (number of moves):

$$T(n) = \begin{cases} 
0, & \text{if } n = 0 \\
1, & \text{if } n = 1 \\
2T(n - 1) + 1, & \text{otherwise}
\end{cases}$$

Solving $T(n)$, we obtain $T(n) = 2^n - 1$. 
3 Insertion Sort

Problem:
Given an array $A[1..n]$ of $n$ numbers, we want to sort the numbers into non-decreasing order.

Solution:
procedure InsertionSort($A, n$)
{
    for $j = 2$ to $n$ do {
        $key := A[j]$;
        $i := j - 1$;
        while $i > 0$ and $A[i] > key$ do {
            $i := i - 1$
        }
        $A[i + 1] := key$
    }
}

Effort (number of moves/comparisons):
In the worst case,

$$T(n) = \sum_{j=2}^{n} = 1 + 2 + \cdots + n - 1 = \frac{n(n - 1)}{2} = O(n^2).$$

In the best case (all input numbers are already sorted), $T(n) = n - 1$.

If we use binary search to determine the location of $A[j]$ in $A[1..j-1]$ to insert and use a height-balanced binary search tree to represent $A[1..j-1]$, the worst case is

$$T(n) = \sum_{i=1}^{n-1} \log_2 i = O(n \log n).$$
4 MergeSort

Problem:
Given an array $A[1..n]$ of $n$ numbers, we want to sort the numbers into non-decreasing order.

Solution:

procedure $MergeSort(A, p, r)$
{
  if $p < r$ then {
    $q := \lfloor \frac{p+r}{2} \rfloor$;
    $MergeSort(A, p, q)$;
    $MergeSort(A, q + 1, r)$;
    $Merge(A, p, q, r)$
  }
}

procedure $Merge(A, p, q, r)$
{
  $n_1 := q - p + 1$;
  $n_2 := r - q$;
  /* create arrays $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$ */ (note: we use a pair of |* and *| as delimiters for comments.)
  for $i = 1$ to $n_1$ do $L[i] := A[p + i - 1]$;
  for $j = 1$ to $n_2$ do $R[j] := A[q + j]$;
  $L[n_1 + 1] := R[n_2 + 1] := \infty$;
  $i := j := 1$;
  for $k = p$ to $r$ do
    if $L[i] \leq R[j]$ then {
      $A[k] := L[i]; i++$
    } else {
      $A[k] := R[j]; j++$
    }
}

Initial call: $MergeSort(A, 1, n)$.

Effort: $T(n) = 2 \cdot T(\frac{n}{2}) + c \cdot n = O(n \log n)$, where $c$ is a constant.
5 Tiling

Problem:
Given a square board divided into $2^{2k}$ unit squares, with each row/column contains $2^k$ unit squares. An arbitrary square is distinguished as special. Also given a supply of L-shape tiles, each looks like a $2 \times 2$ board with one square removed. Can we cover the board (except the special square) with tiles? If we can, provide a method to do it.

Solution:
For $k = 0$ and $k = 1$, a solution is shown in Figure 2(a) and (b), respectively.
Consider $k = m > 1$, and assume that for $k = m - 1$ the problem is solvable.
Divide the board into 4 equal square sub-boards, and shown in Figure 2(c).
The special square belongs to exactly one sub-board. Place a tile on the middle so as to cover one unit square of each of the other sub-boards. Then, we obtain four $2^{m-1} \times 2^{m-1}$ sub-boards, each containing one special square (i.e. we have 4 sub-problems). By the induction hypothesis, each of these sub-problems can be solved. Then, the final solution is the combination of central tile and the solutions of the 4 sub-problems.

![Figure 2: Solution for the tiling problem: (a) $k = 0$; (b) $k = 1$; (c) $k = 2^m$, where $m > 1$.](image)

Observations:
- This is a divide-and-conquer method (or algorithm);
- This is also a constructive proof (i.e. proof by construction).