Suggested Solutions to Midterm Problems

Problems

1. Given a set of $n+1$ numbers out of the first $2n$ (starting from 1) natural numbers 1, 2, 3, ..., $2n$, prove *by induction* that there are two numbers in the set, one of which divides the other.

   Solution. The proof is by induction on n.

   Base case ($n = 1$): There is only one subset of 2 ($= n+1$) numbers from $\{1, 2\}$, which is the set $\{1, 2\}$ itself. 1 divides 2.
   Inductive step ($n = k+1 > 1$): We need to show that any selection (subset) of $k+2$ numbers from $\{1, 2, \cdots, 2k, 2k+1, 2k+2\}$ contains two numbers, one of which divides the other. If the selection includes $k+1$ numbers from $\{1, 2, \cdots, 2k\}$, then by the induction hypothesis we are done. Otherwise, the selection must contain both $2k+1$ and $2k+2$ and also include a selection S of other k numbers from $\{1, 2, \cdots, 2k\}$.
   Case one ($k+1 \in S$): $k+1$ divides $2k+2$.
   Case two ($k+1 \notin S$): If S happens to contain two numbers one of which divides the other, then we are done. Otherwise, from the induction hypothesis, S must contain a number that divides $k+1$. This is so, because (a) $k+1$ does not divide any number in $\{1, 2, \cdots, 2k\}$ and (b) $S \cup \{k+1\}$ is a selection of $k+1$ numbers from $\{1, 2, \cdots, 2k\}$ and by the induction hypothesis must contain two numbers one of which divides the other. The number that divides $k+1$ also divides $2k+2$. □

2. Construct a gray code of length $\lceil \log_2 14 \rceil$ ($= 4$) for 14 objects. Show how the gray code is constructed *systematically* from gray codes of smaller lengths.

   Solution. Let $(c_1, c_2, \ldots, c_n)^R$ denote the list $c_n, c_{n-1}, \ldots, c_1$.

   Code of length 1 for 2 objects: $0, 1$.
   Code of length 2 for 2 objects: $00, 01$.
   Code of length 2 for 3 objects: $00, 01, 11$ (which is open).
   Code #1 of length 3 for 3 objects: $000, 001, 011$.
   Code #2 of length 3 for 3 objects: $100, 101, 111$.
   Code of length 3 for 6 objects: $000, 001, 011, (100, 101, 111)^R$.
   Code of length 3 for 7 objects: $000, 001, 011, 111, 101, 100, 110$ (which is open).
   Code #1 of length 4 for 7 objects: $0000, 0001, 0011, 0111, 0101, 0100, 0110$.
   Code #2 of length 4 for 7 objects: $1000, 1001, 1011, 1111, 1101, 1100, 1110$.
   Code of length 4 for 14 objects:
   $0000, 0001, 0011, 0111, 0101, 0100, 0110, (1000, 1001, 1011, 1111, 1101, 1100, 1110)^R$. □

3. Let $T(h)$ denote the number of nodes in a smallest AVL tree of height h (smallest in the sense of having the least number of nodes); the height of an empty tree is defined to be 0.

   (a) Define a recurrence relation for $T(h)$ ($h \geq 0$). Be sure to cover the base cases (or marginal cases).

   Solution.

$$\begin{cases} T(0) = 0 \\ T(1) = 1 \\ T(h) = T(h-1) + T(h-2) + 1, \quad h \geq 2 \end{cases}$$

   □

   (b) Derive, based on the preceding recurrence relation, a lower bound for $T(h)$, showing that $T(h)$ grows at least exponentially with h. How do you infer, from the lower bound, the time complexity of performing a search operation on an AVL tree of size n?

   Solution. A precise solution to $T(h)$ may be derived by establishing the relation $T(h) = F(h+2) - 1$, where $F(n)$ is the n-th Fibonacci number (as defined in Chapter 3.5 of Manber's book) for which we already know the closed form; the proof is in fact quite simple by induction. However, we will prove directly a lower bound for $T(h)$, namely $\Omega((\frac{3}{2})^h)$, which is good enough to show its exponential growth. The proof is by induction on h, showing that $T(h) \geq \frac{2}{3}(\frac{3}{2})^h$, for $h \geq 1$.

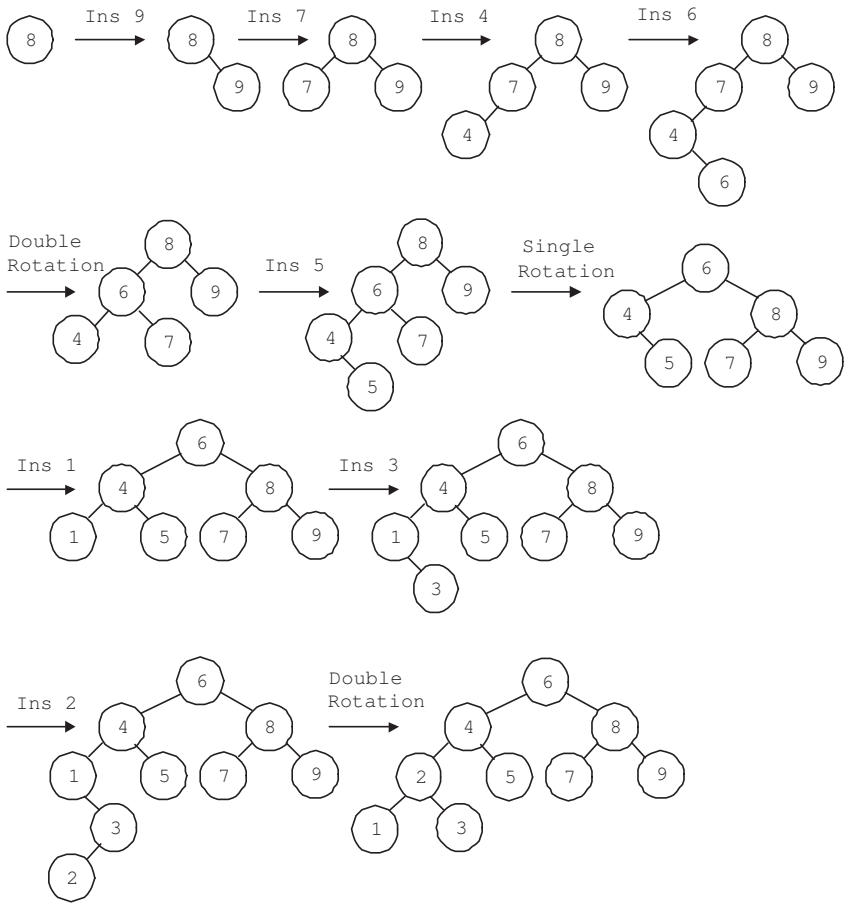
   Base case ($h = 1$ or $h = 2$): When $h = 1$, $\frac{2}{3}(\frac{3}{2})^h \leq 1 = T(h)$. When $h = 2$, $\frac{2}{3}(\frac{3}{2})^h = \frac{3}{2} \leq 2 = T(h)$.

   Inductive step ($h > 2$): $T(h) = T(h-1) + T(h-2) + 1 \geq \frac{2}{3}(\frac{3}{2})^{h-1} + \frac{2}{3}(\frac{3}{2})^{h-2} + 1 \geq (1 + \frac{2}{3})(\frac{3}{2})^{h-2} = (1 + \frac{2}{3})(\frac{3}{2})^{-2}(\frac{3}{2})^h = \frac{20}{27}(\frac{3}{2})^h \geq \frac{2}{3}(\frac{3}{2})^h$.

   Therefore, for an AVL tree of size n, its height h must be such that $\frac{2}{3}(\frac{3}{2})^h \leq T(h) \leq n$. It follows that $h \leq \frac{1}{\log 1.5} \log n + 1$, implying $h = O(\log n)$. Performing a search operation on the AVL tree takes time proportional to its height and hence the time complexity of a search operation is bounded from above by $O(\log n)$. □

4. Show all intermediate and the final AVL trees formed by inserting the numbers 8, 9, 7, 4, 6, 5, 1, 3, and 2 (in this order) into an empty tree. Please use the following ordering convention: the key of an internal node is larger than that of its left child and smaller than that of its right child. If a rotation is performed during an insertion, please also show the tree before the rotation. (15 points)

   Solution. (陳郁方)



□

5. Design an algorithm that solves the following variant of the towers of Hanoi problem (adapted from Exercise 5.24 of Manber's book): Like in the original problem, there are three pegs, each capable of holding up to n disks (for some given n). Initially, the n disks (of different sizes) are arbitrarily distributed among the three pegs, all in a decreasing order of sizes (from bottom to top). The goal is to move all the n disks, one at a time using only the pegs as temporary storage, to one of the three pegs, without violating the ordering constraint and with as few moves as possible. Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary. (15 points)

Solution. (楊智皓)

```
Algorithm main(n,A,B,C)
begin
   des := find_the_disk(n,A,B,C);
   {S_1,S_2} := {A,B,C} - des;
   New_Hanoi_tower(n-1,des,S_1,S_2);
end
```

```
Algorithm New_Hanoi_tower(n,A,B,C)
begin
   src := find_the_disk(n,A,B,C);
   des := {B,C} - src;
   if src = A then
      New_Hanoi_tower(n-1,A,B,C);
   else if n = 1 then
      move n to A;
   else if A.top > n then
      move n to A;
      New_Hanoi_tower(n-1,A,B,C);
   else
      New_Hanoi_tower(n-1,des,src,A);
      move n to A;
      Hanoi_tower(n-1,des,A,src);
end

Algorithm find_the_disk(n,A,B,C)
begin
   return which peg contains n;
end
```

□

6. Design an efficient algorithm that, given an array A of n integers and an integer x, determine whether A contains two integers whose sum is exactly x. Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary. Give an analysis of its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem.

   Solution. The straightforward solution of trying every pair in A would take $O(n^2)$ time, as there are $\frac{n(n-1)}{2}$ possible pairs. When A is sorted (in increasing order), finding the pair (if it exists) can be done much more efficiently as follows: If $A[1] + A[n] < x$, then $A[1]$ cannot be one of the pair we are looking for, as $A[1] + A[j]$ will be smaller than x for any $j \leq n$. On the other hand, if $A[1] + A[n] > x$, then $A[n]$ cannot be one of the pair we are looking for, as $A[i] + A[n]$ will be greater than x for any $i \geq 1$. In either case, we can eliminate one element. So, we sort A first with, for example, the heapsort algorithm and then invoke the procedure below.

   procedure Find_Pair (A, n, x);
   begin
   $i := 1$;

```
    j := n;
    while i < j do
        if A[i] + A[j] = x then
            break;
        if A[i] + A[j] < x then
            i := i + 1;
        else j := j − 1;
    if i < j then
        print i, j;
    else print "no solution"
end
```

The while loop will be executed at most $n-1$ times, hence the running time of the procedure is $O(n)$. Together with the sorting part, the whole algorithm will run in $O(n \log n)$ time. □

7. Rearrange the following array into a (max) heap using the bottom-up approach.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	8	3	5	9	14	7	6	1	4	10	13	15	12	11

Show the result after each element is added to the part of array that already satisfies the heap property.

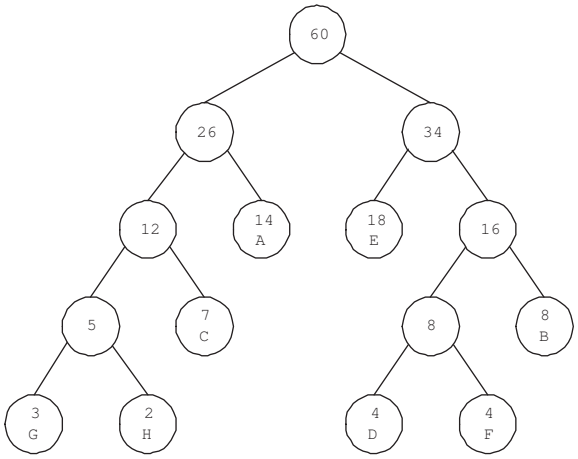
Solution. (陳郁方)

2	8	3	5	9	14	7	6	1	4	10	13	15	12	11
2	8	3	5	9	14	**12**	6	1	4	10	13	15	**7**	11
2	8	3	5	9	**15**	12	6	1	4	10	13	**14**	7	11
2	8	3	5	**10**	15	12	6	1	4	**9**	13	14	7	11
2	8	3	**6**	10	15	12	**5**	1	4	9	13	14	7	11
2	8	**15**	6	10	**14**	12	5	1	4	9	13	**3**	7	11
2	**10**	15	6	**9**	14	12	5	1	4	**8**	13	3	7	11
15	10	**14**	6	9	**13**	12	5	1	4	8	**2**	3	7	11

□

8. Draw a Huffman tree for a text with the following frequency distribution: $A : 14$, $B : 8$, $C : 7$, $D : 4$, $E : 18$, $F : 4$, $G : 3$, and $H : 2$.

Solution. (陳郁方)



□

9. Solve one of the following two problems:

   (a) Suppose that you are given an algorithm as a *black box* (you cannot see how it is designed) that has the following properties: If you input any sequence of real numbers and an integer k, the algorithm will answer "yes" or "no," indicating whether there is a subset of the numbers whose sum is exactly k. Show how to use this black box to find the subset whose sum is k, if it exists. You should use the black box $O(n)$ times (where n is the size of the sequence).

   Solution. Let `Find_Subset` denote the given algorithm, which takes as input an array of real numbers, the size of the array (these two together representing the sequence of real numbers), and an integer.

```
Algorithm Print_Subset(S,n,k);
begin
   if Find_Subset(S,n,k)="no" then
      print "No suitable subset"; halt;
   print "Below is a suitable subset:";
   sum := 0.0;
   i := 1;
   while sum<k do
      this := S[i];
      S[i] := 0;
      if Find_Subset(S,n,k)="no" then
         print this;
         sum := sum + this;
         S[i] := this;
      i := i + 1;
end
```

□

(b) Prove that the sum of the heights of all nodes in a complete binary tree with n nodes is at most $n-1$. (A complete binary tree with n nodes is one that can be compactly represented by an array A of size n, where the root is stored in $A[1]$ and the left and the right children of $A[i]$, $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$, are stored respectively in $A[2i]$ and $A[2i+1]$. Notice that, in Manber's book a complete binary tree is referred to as a balanced binary tree and a full binary tree as a complete binary tree. Manber's definitions seem to be less frequently used. Do not let the different names confuse you.)

Solution. Let $G(n)$ denote the sum of the heights of all nodes in a complete binary tree with n nodes. For a full binary tree (a special case of complete binary trees) with $n = 2^{h+1} - 1$ nodes where h is the height of the tree, we already know that $G(n) = 2^{h+1} - (h+2) = n - (h+1) \leq n - 1$. With this as a basis, we prove the general case of arbitrary complete binary trees by induction on the number n (≥ 1) of nodes.

Base case ($n = 1$ or $n = 2$): When $n = 1$, the tree is the smallest full binary tree with one single node whose height is 0. So, $G(n) = 0 \leq 1 - 1 = n - 1$. When $n = 2$, the tree has one additional node as the left child of the root. The height of the root is 1, while that of its left child is 0. So, $G(n) = 1 \leq 2 - 1 = n - 1$.

Inductive step ($n > 2$): If n happens to be equal to $2^{h+1} - 1$ for some $h \geq 1$, i.e., the tree is full, then we are done; note that this covers the case of $n = 3 = 2^{1+1} - 1$. Otherwise, suppose $2^{h+1} - 1 < n < 2^{h+2} - 1$ ($h \geq 1$), i.e., the tree is a "proper" complete binary tree with height $h + 1 \geq 2$. We observe that at least one of the two subtrees of the root is full, while the other is complete (possibly full). There are three cases to consider:

Case 1: The left subtree is full with n_1 nodes and the right one is complete but not full with n_2 nodes (such that $n_1 + n_2 + 1 = n$). In this case, both subtrees much be of height h and $n_1 = 2^{h+1} - 1$. From the special case of full binary trees and the induction hypothesis, $G(n_1) = 2^{h+1} - (h+2) = n_1 - (h+1)$ and $G(n_2) \leq n_2 - 1$. $G(n) = G(n_1) + G(n_2) + (h+1) \leq (n_1 - (h+1)) + (n_2 - 1) + (h+1) = (n_1 + n_2 + 1) - 2 \leq n - 1$.

Case 2: The left subtree is full with n_1 nodes and the right one is also full with n_2 nodes. In this case, the left subtree much be of height h and $n_1 = 2^{h+1} - 1$, while the right subtree much be of height $h - 1$ and $n_2 = 2^h - 1$. From the special case of full binary trees, $G(n_1) = 2^{h+1} - (h+2) = n_1 - (h+1)$ and $G(n_2) = 2^h - (h+1) = n_2 - h$. $G(n) = G(n_1) + G(n_2) + (h+1) \leq (n_1 - (h+1)) + (n_2 - h) + (h+1) = (n_1 + n_2 + 1) - (h+1) \leq n - 1$.

Case 3: The left subtree is complete but not full with n_1 nodes and the right one is full with n_2 nodes. In this case, the left subtree much be of height h, while the right subtree much be of height $h - 1$ and $n_2 = 2^h - 1$. From the induction hypothesis and the special case of full binary trees, $G(n_1) \leq n_1 - 1$ and $G(n_2) = 2^h - (h+1) = n_2 - h$.

$G(n) = G(n_1)+G(n_2)+(h+1) \leq (n_1-1)+(n_2-h)+(h+1) = (n_1+n_2+1)-1 = n-1.$
□