Suggested Solutions to Midterm Problems

Problems

1. Below is an algorithm for solving a variant of the Towers of Hanoi puzzle with an additional fourth peg D; `Towers_Hanoi` is an algorithm for the original puzzle.

```
Algorithm Four_Towers_Hanoi(A,B,C,D,n);
begin
   if n<=2 then
      Towers_Hanoi(A,B,C,n);
   else
      Four_Towers_Hanoi(A,D,B,C,n-2);
      Towers_Hanoi(A,B,C,2);
      Four_Towers_Hanoi(D,B,C,A,n-2);
end;
```

Let $T(n)$ denote the number of moves needed for n disks. Write a recurrence relation for $T(n)$ and solve it.

Solution. `Towers_Hanoi(A,B,C,1)` takes 1 move, while `Towers_Hanoi(A,B,C,2)` takes 3 moves. A recurrence relation for $T(n)$ is the following:

$$
\begin{aligned}
&T(1) = 1\\
&T(2) = 3\\
&T(n) = 2T(n-2) + 3, \text{for } n \geq 3
\end{aligned}
$$

We solve the recurrence relation by considering odd and even n's separately.

When n (≥ 3) is odd,

$$
\begin{array}{rl}
T(n) = & 2T(n-2) + 3\\
2T(n-2) = & 2(2T(n-4)+3) = 2^2T(n-4) + 2\times 3\\
2^2T(n-4) = & 2^2(2T(n-6)+3) = 2^3T(n-6) + 2^2\times 3\\
\cdots & \cdots\\
2^{\frac{n-3}{2}}T(3) = & 2^{\frac{n-3}{2}}(2T(1)+3) = 2^{\frac{n-1}{2}} + 2^{\frac{n-3}{2}}\times 3\\
\hline
T(n) = & 2^{\frac{n-1}{2}} + 3\times(2^{\frac{n-1}{2}} - 1)\\
= & 2^{\frac{n+3}{2}} - 3
\end{array}
$$

When n (≥ 3) is even,

$$\begin{array}{rl}
T(n) = & 2T(n-2)+3 \\
2T(n-2) = & 2(2T(n-4)+3) = 2^2T(n-4) + 2\times 3 \\
2^2T(n-4) = & 2^2(2T(n-6)+3) = 2^3T(n-6) + 2^2\times 3 \\
\cdots & \cdots \\
2^{\frac{n-4}{2}}T(4) = & 2^{\frac{n-4}{2}}(2T(2)+3) = 3\times 2^{\frac{n-2}{2}} + 2^{\frac{n-4}{2}}\times 3 \\
\hline
T(n) = & 3\times 2^{\frac{n-2}{2}} + 3\times(2^{\frac{n-2}{2}}-1) \\
= & 3\times 2^{\frac{n}{2}} - 3
\end{array}$$

□

2. Consider binary trees where each node stores a non-negative integer. Design an algorithm that, given such a tree T and a non-negative integer k as input, determines whether T contains a branch (from the root to a leaf) such that the sum of all numbers stored on the nodes of the branch equals k. The more efficient your algorithm is, the more points you will be credited for this problem. Is there a possibility that your code may overflow? Have you avoided the problem? (15 points)

Solution.

```
Algorithm Check_Branch_Sum(T, s);
begin
    if s >= 0 then
        answer := Check_Branch(T, s);
    else answer := false
end

procedure Check_Branch(T, s);
begin
    if T = nil then return(false);
    if (T^.left <> nil) or (T^.right <> nil) then
        if T^.value <= s then
            s := s - T^.value;
            if Check_Branch(T^.left, s) or Check_Branch(T^.right, s) then
                return(true);
    else if T^.value = s then return(true);
    return(false)
end
```

It is assumed that in the evaluation of a boolean condition "A or B", B will not be evaluated when A has been evaluated to true. It is also assumed that the value stored in each node is non-negative and hence not checked. □

3. Modify the following code for determining the sum of the maximum consecutive subsequence so that it also records the start and end indices of the subsequence.

Algorithm Max_Consec_Subseq (X, n);
begin
  Global_Max := 0;
  Suffix_Max := 0;
  for i := 1 **to** n **do**
    if $x[i]$ + *Suffix_Max* > *Global_Max* **then**
      Suffix_Max := *Suffix_Max* + $x[i]$;
      Global_Max := *Suffix_Max*
    else if $x[i]$ + *Suffix_Max* > 0 **then**
        Suffix_Max := *Suffix_Max* + $x[i]$
    else *Suffix_Max* := 0
end

Solution. (蔡明憲)

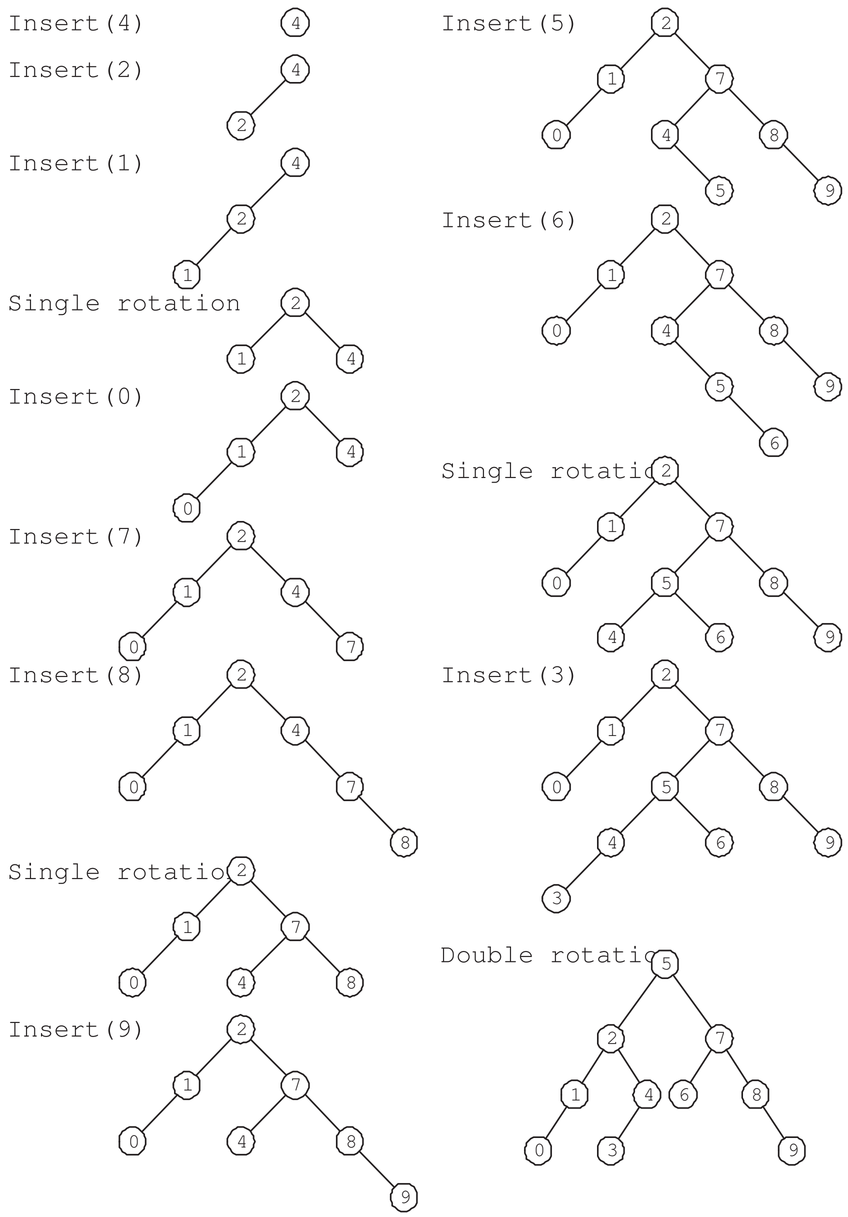
```
Algorithm Max_Conseq_Subseq(X, n);
begin
    Global_Max := 0;
    Suffix_Max := 0;
    Suffix_Start_Index := 0;
    Global_Start_Index := 0;
    Global_End_Index := 0;
    for i := 1 to n do
        if x[i] + Suffix_Max > Global_Max then
            Suffix_Max := Suffix_Max + x[i];
            Global_Max := Suffix_Max;
            Global_Start_Index := Suffix_Start_Index;
            Global_End_Index := i;
        else if x[i] + Suffix_Max > 0 then
            Suffix_Max := Suffix_Max + x[i];
        else
            Suffix_Max := 0
            Suffix_Start_Index := i + 1;
end
```

□

4. Show all intermediate and the final AVL trees formed by inserting the numbers 4, 2, 1, 0, 7, 8, 9, 5, 6, and 3 (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. (蔣孟儒)



□

5. Please present the union-find algorithm with balancing and path compression in a suitable pseudocode. (20 points)

Solution.

```
Algorithm Union_Find_Init(A,n);
```

```
begin
    for i := 1 to n do
        A[i].parent := nil;
        A[i].size := 1
end

Algorithm Union(a,b);
begin
    x := Find(a);
    y := Find(b);
    if x <> y then
        if A[x].size > A[y].size then
            A[y].parent := x;
            A[x].size := A[x].size] + A[y].size;
        else A[x].parent := y;
             A[y].size := A[y].size] + A[x].size
end

Algorithm Find(a);
begin
    if A[a].parent <> nil then
        A[a].parent := Find(A[a].parent);
        Find := A[a].parent;
    else Find := a
end
```

□

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

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

Solution. (黃俊誌)

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

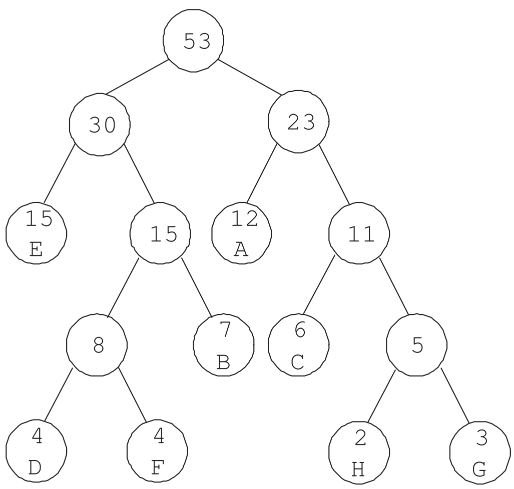
□

7. Design an algorithm that determines whether two sets of numbers (represented as arrays) are disjoint; the more efficient your algorithm is, the more points you will be credited for this problem. State the time complexity of your algorithm in terms of the sizes m and n of the given sets. Be sure to consider the case where m is substantially larger than n.

   Solution. The basic idea is to sort one of the two sets and use binary search to check if some member of the other set appears in it. A little calculation shows that it is better to sort the smaller set, resulting in a complexity of $O((m+n)\log n)$. □

8. Draw a Huffman tree for a text with the following frequency distribution: A : 12, B : 7, C : 6, D : 4, E : 15, F : 4, G : 3, and H : 2.

   Solution. (黃俊誌)



□