Suggested Solutions to Midterm Problems

Problems

1. Prove by induction that a ring of even size can be colored with two colors and a ring of odd size with three colors such that no two adjacent nodes have the same color.

   Solution. The proof is by *strong* induction on the number n of nodes.

   Base case: A ring of two nodes can be colored with two colors and a ring of three nodes with three colors.

   Inductive step: Consider a ring of n (≥ 4) nodes. There are two cases: one when n is odd and the other when n is even.

   If n is odd, then we remove an arbitrary node from the ring to obtain a ring of $n-1$ (which is even) nodes. By the induction hypothesis, the smaller ring can be properly colored with two colors. The removed node can be colored with the third color.

   If n is even, then we remove two arbitrary but consecutive nodes to obtain a ring of $n-2$ (which is even) nodes. The smaller ring, again by the induction hypothesis, can be properly colored with two colors. The removed two nodes can be colored differently with the two colors and inserted back into the ring in such a way that no two adjacent nodes have the same color. □

2. Construct a gray code (which is open) of length $\lceil \log_2 13 \rceil$ ($= 4$) for 13 objects. Show how the gray code is constructed *systematically* from gray codes of smaller lengths.

   Solution. (楊智皓)

   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: $00, 01$.
   11 has not been used and differs from 01 by 1 bit.
   Code of length 2 for 3 objects: $00, 01, \mathbf{11}$ (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 #1 of length 4 for 6 objects: $0000, 0001, 0011, 0111, 0101, 0100$.
   Code #2 of length 4 for 6 objects: $1000, 1001, 1011, 1111, 1101, 1100$.
   Code of length 4 for 12 objects:
   $0000, 0001, 0011, 0111, 0101, 0100$,x$(1000, 1001, 1011, 1111, 1101, 1100)^R$
   $= 0000, 0001, 0011, 0111, 0101, 0100, 1100, 1101, 1111, 1011, 1001, 1000$.

1010 has not been used and differs from 1000 by 1 bit.

Code of length 4 for 13 objects:

$0000, 0001, 0011, 0111, 0101, 0100, 1100, 1101, 1111, 1011, 1001, 1000, \mathbf{1010}$ (which is open).

□

3. Solve the following recurrence relation with full history:

$$\begin{cases} T(1) = 0 \\ T(n) = (n-1) + \sum_{i=1}^{n-1} T(i), \quad n \geq 2 \end{cases}$$

You need to provide an exact, not just asymptotic, solution.

Solution. (陳郁方)

$T(n) = (n-1) + \sum_{i=1}^{n-1} T(i) \ldots (1)$

$T(n+1) = n + \sum_{i=1}^{n} T(i) \ldots (2)$

We can obtain $T(n+1) = 1 + 2T(n)$ from $(2) - (1)$.

$T(n) = 1 + 2T(n-1)$

$= 1 + 2 + 2^2 T(n-2)$

$= 1 + 2 + 2^2 + 2^3 T(n-3)$

$= ...$

$= 1 + 2 + 2^2 + 2^3 + ... + 2^{n-2}$

$= 2^{n-1} - 1$ □

4. Find the asymptotic behavior of the function $T(n)$ defined as follows:

$$\begin{cases} T(1) = 1 \\ T(n) = 2\sqrt{n} + T(n/2), \quad n = 2^i \ (i \geq 1) \end{cases}$$

You should try to solve this problem without resorting to the general theorem for divide-and-conquer relations discussed in class. The asymptotic bound should be as tight as possible. (Hint: guess and prove by induction.)

Solution. (陳郁方)

$T(n) = O(\sqrt{n})$.

We prove by induction that $T(n) \leq 7\sqrt{n}$.

Base case: $T(1) = 1 \leq 7 = 7\sqrt{1}$.

Inductive step: $T(2n) = T(n) + 2\sqrt{2n} \leq 7\sqrt{n} + 2\sqrt{2}\sqrt{n} = (7 + 2\sqrt{2})\sqrt{n} \leq 7\sqrt{2}\sqrt{n} = 7\sqrt{2n}$.

Note: How was the constant 7 determined? Suppose $T(n) \leq c\sqrt{n}$ for some c. Anticipating the proof obligation in the inductive step that $T(2n) = T(n) + 2\sqrt{2n} \leq c\sqrt{n} + 2\sqrt{2}\sqrt{n} \leq c\sqrt{2}\sqrt{n}$, we stipulate that the constant c should be $\geq 2\sqrt{2}/(\sqrt{2}-1) = 6.\cdots$; 7 is just one of such constants. But, how do we know in the first place that $T(n) = O(\sqrt{n})$ is a good guess?

The guess is motivated by considering two other recurrence relations:

"$T(n) = T(n/2) + 1$, $T(1) = 1$" ($T(n) = O(\log n)$) and "$T(n) = T(n/2) + n$, $T(1) = 1$" ($T(n) = O(n)$). □

5. 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?

   Solution.

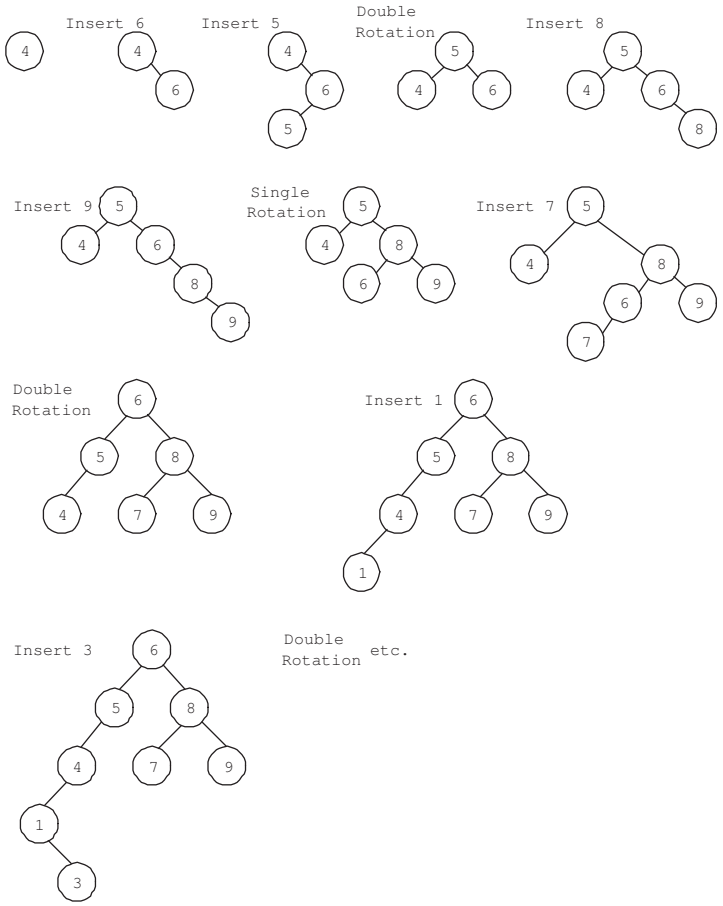
```
Algorithm Check_Branch_Sum(T, s);
begin
    if s >= 0 then
        answer := Check_Branch(T, s);
    else answer := false
end

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. □

6. Show all intermediate and the final AVL trees formed by inserting the numbers 4, 6, 5, 8, 9, 7, 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. (楊智皓)



□

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

Show the result after each element is added to the part of array that already satisfies the heap property. (15 points)

Solution. (陳郁方)

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

□

8. 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
```

□

9. A string $A = a_0a_1 \cdots a_{n-1}$ is a cyclic shift of another string $B = b_0b_1 \cdots b_{n-1}$ if there exists an index k, $0 \leq k \leq n-1$, such that $a_i = b_{(k+i) \bmod n}$ for all i, $0 \leq i \leq n-1$. For example, "`defgabc`" is a cyclic shift of "`abcdefg`". Suppose that you already have an algorithm called **Substring** that can determine whether a string is a substring of another. Design an algorithm using **Substring** that, given two strings A and B, determines whether A is a cyclic shift of B. Please present your algorithm in an adequate pseudo code and make assumptions wherever you feel necessary. Explain why your algorithm is correct and analyze its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem.

Solution. For two strings A and B of equal length, A is a cyclic shift of B if and only if B is a substring of AA (the concatenation of two copies of A) .

```
Algorithm Cyclic_Shift(A,n,B,m);
begin
   if n<>m then
```

```
      print "No"; halt;
   /* concatenate two copies of A */
   for i:=1 to n do
      C[i] := A[i];
      C[i+n] := A[i];
   if Substring(C,2n,B,m) then
      print "Yes"
   else print "No";
end
```

Suppose the time complexity of `Substring` is $O(f(n,m))$, where n is the length of the first string and m that of the second. The time complexity of the call `Substring(C,2n,B,m)` will then be $O(f(2n,m))$. Since $O(f(2n,m))$ is at least $O(n)$, the time for executing the for loop does not affect the asymptotic complexity. The time complexity of `Cyclic_Shift` hence is $O(f(2n,m))$. □