Suggested Solutions to Midterm Problems

1. Prove by induction that the regions formed by a planar graph all of whose vertices have even degrees can be colored with two colors such that no two adjacent regions have the same color.

   Solution. The proof is by *strong* induction on the number m of edges in the graph.

   Base case: When $m = 0$ (i.e., the graph has one or more isolated vertices), there is only one region which can be colored by any of the two colors.

   Induction step: Consider a planar graph G with $m = k$ ($k \geq 1$) edges. The induction hypothesis says that "a planar graph with $< k$ edges can be *properly* colored with two colored (such that no two adjacent regions have the same color)".

   G must contain a simple cycle (a cycle that passes through a node at most once). Remove the cycle from G to obtain a graph G' that has $< k$ edges and all of whose edges have even degrees. By the induction hypothesis, G' can be properly colored with two colors. The removed cycle, when put back, divides G' into two areas: one inside the cycle and the other outside the cycle. The cycle also divides some of the regions of G' into smaller regions (it is possible that a region be divided into more than two smaller regions). Flip the colors of the regions inside the cycle and we get a proper coloring for graph G (why this is so follows from an argument similar to that for the example of regions divided by lines in general position that we discussed in class). $\Box$

2. For each of the following pairs of functions, decide if $f(n) = O(g(n))$ holds and if $f(n) = \Omega(g(n))$ holds? Justify your answers.

	$f(n)$	$g(n)$
(a)	$(\log n)^{\log n}$	$\frac{n}{\log n}$
(b)	$n^3 \cdot 2^n$	3^n

   Solution. (a) $(\log n)^{\log n} = \Omega(\frac{n}{\log n})$, but $(\log n)^{\log n} \neq O(\frac{n}{\log n})$. This can be proven by showing that $\frac{n}{\log n} = o((\log n)^{\log n})$, i.e., $\lim_{n\to\infty} \frac{\frac{n}{\log n}}{(\log n)^{\log n}} = 0$.
   Since f and g are monotonically increasing and diverge as n approaches the inifinity, it suffices to show that $\lim_{n\to\infty} \frac{\log \frac{n}{\log n}}{\log(\log n)^{\log n}} = 0$. $\lim_{n\to\infty} \frac{\log \frac{n}{\log n}}{\log(\log n)^{\log n}} = \lim_{n\to\infty} \frac{\log n - \log\log n}{\log n \log\log n} \leq \lim_{n\to\infty} \frac{\log n}{\log n \log\log n} = \lim_{n\to\infty} \frac{1}{\log\log n} = 0$.
   (b) $n^3 2^n = O(3^n)$, but $n^3 2^n \neq \Omega(3^n)$. It suffices to show that $n^3 2^n = o(3^n)$, i.e., $\lim_{n\to\infty} \frac{n^3 2^n}{3^n} = \lim_{n\to\infty} \frac{n^3}{(1.5)^n} = \lim_{n\to\infty} \frac{n^3}{e^{n\ln(1.5)}} = 0$. Applying L'Hospital's rule repeatedly, $\lim_{n\to\infty} \frac{n^3}{e^{n\ln(1.5)}} = \lim_{n\to\infty} \frac{3n^2}{\ln(1.5)e^{n\ln(1.5)}} = \lim_{n\to\infty} \frac{6n}{\ln(1.5)^2 e^{n\ln(1.5)}} = \lim_{n\to\infty} \frac{6}{\ln(1.5)^3 e^{n\ln(1.5)}} = 0$. $\Box$

3. In the towers of Hanoi puzzle, there are three pegs A, B, and C, with n (generalizing the original eight) disks of different sizes stacked in decreasing order on peg A. The objective is to transfer all the disks on peg A to peg B, moving one disk at a time (from one peg to one of the other two) and never having a larger disk stacked upon a smaller one.

(a) Give an algorithm to solve the puzzle. Explain how induction works here.

Solution.

```
Algorithm Towers_Hanoi(A,B,C,n);
begin
   if n=1 then
      pop x from A and push x to B
   else
      Towers_Hanoi(A,C,B,n-1);
      pop x from A and push x to B;
      Towers_Hanoi(C,B,A,n-1);
end;
```

Base case: When $n = 1$, we simply move the only disk from A to B.
Inductive step: To move $n(\geq 2)$ disks from A to B, we first move the top $n-1$ disks from A to C; we know how to do this from the induction hypothesis. The n-th and largest disk remains at the bottom of A and it is larger than any disk that is put on A throughout those moves. Therefore, all the moves meet the "size constraint" of never having a larger disk stacked upon a smaller one. We then move the last disk (which is the largest one) left on A to B (which is empty right before the move); this move also meets the size constraint. Finally, we move the $n-1$ disks from C to B; we know how to do this from the induction hypothesis. All the moves meet the size constraint like before. □

(b) Compute the total number of moves in the algorithm. Show the details of your calculation.

Solution. We count "pop x from A and push x to B" as one move. Let $T(n)$ denote the number of moves required for n disks.

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

Solving the equation, we get $T(n) = 2^n - 1$, for $n \geq 1$. □

4. Construct a gray code of length $\lceil \log_2 12 \rceil$ $(= 4)$ for 12 objects. Show how the gray code is constructed from gray codes of smaller lengths. Your construction should be systematic.

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 #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:
$0000, 0001, 0011, 0111, 0101, 0100, (1000, 1001, 1011, 1111, 1101, 1100)^R$. □

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

Solution. See the attached. □

6. Apply the quicksort algorithm to the following array. Show the result after each partition operation.

7	1	5	11	14	12	2	15	8	3	13	4	10	9	16	6

Solution.

7	1	5	11	14	12	2	15	8	3	13	4	10	9	16	6
2	1	5	6	4	3	**7**	15	8	12	13	14	10	9	16	11
1	**2**	5	6	4	3	**7**	15	8	12	13	14	10	9	16	11
1	**2**	5	6	4	3	**7**	15	8	12	13	14	10	9	16	11
1	**2**	4	3	**5**	6	**7**	15	8	12	13	14	10	9	16	11
1	**2**	3	**4**	**5**	6	**7**	15	8	12	13	14	10	9	16	11
1	**2**	3	**4**	**5**	6	**7**	11	8	12	13	14	10	9	**15**	16
1	**2**	3	**4**	**5**	6	**7**	10	8	9	**11**	14	13	12	**15**	16
1	**2**	3	**4**	**5**	6	**7**	9	8	**10**	**11**	14	13	12	**15**	16
1	**2**	3	**4**	**5**	6	**7**	8	**9**	**10**	**11**	12	13	14	**15**	16
1	**2**	3	**4**	**5**	6	**7**	8	**9**	**10**	**11**	12	13	**14**	**15**	16
1	**2**	3	**4**	**5**	6	**7**	8	**9**	**10**	**11**	**12**	13	**14**	**15**	16

□

7. Rearrange the following array into a heap using the buttom-up approach.

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

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

Solution.

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

□

8. Write a program (or modify the following code) to recover the solution to a knapsack problem using the *belong* flag. You should make your solution as efficient as possible. (Note: The knapsack algorithm that appeared in the original problem statement has been removed.)

Solution.

Procedure Print_Solution (S, P, n, K)**;**
begin
    if $\neg P[n, K].exist$ **then**
        print "no solution"
    else $i := n$;
         $k := K$;
         while $k > 0$ **do**
             if $P[i, k].belong$ **then**
                 print i;
                 $k := k - S[i]$;
             $i := i - 1$
end

□

9. Compute the *next* table as in the KMP algorithm for the string *ababaababab*. Show the details of your calculation.

Solution.

$next[1] = -1$.
$next[2] = 0$.
$next[3] = 0$: $B_{3-1} = B_2 = b$, while $B_{next[3-1]+1} = B_1 = a$; $B_{3-1} \neq B_{next[3-1]+1}$. As $next[next[3-1]+1]+1 = next[1]+1 = 0$, $next[3] = 0$.

$next[4] = 1$: $B_{4-1} = B_3 = a$, while $B_{next[4-1]+1} = B_1 = a$; $B_{4-1} = B_{next[4-1]+1}$. So, $next[4] = next[4-1]+1 = 1$.

$next[5] = 2$: $B_{5-1} = B_4 = b$, while $B_{next[5-1]+1} = B_2 = b$; $B_{5-1} = B_{next[5-1]+1}$. So, $next[5] = next[5-1]+1 = 2$.

$next[6] = 3$: $B_{6-1} = B_5 = a$, while $B_{next[6-1]+1} = B_3 = a$; $B_{6-1} = B_{next[6-1]+1}$. So, $next[6] = next[6-1]+1 = 3$.

$next[7] = 1$: $B_{7-1} = B_6 = a$, while $B_{next[7-1]+1} = B_4 = b$; $B_{7-1} \neq B_{next[7-1]+1}$. $B_{next[4]+1} = B_2 = b$; $B_{7-1} \neq B_{next[4]+1}$. $B_{next[2]+1} = B_1 = a$; $B_{7-1} = B_{next[2]+1}$. So, $next[7] = 1$.

And so on.

1	2	3	4	5	6	7	8	9	10	11
a	b	a	b	a	a	b	a	b	a	b
-1	0	0	1	2	3	1	2	3	4	5

□

10. Explain why the time complexity of the KMP algorithm is $O(n)$, where n is the length of string A.

Solution. See Page 154 of Manber's book. □