Midterm

Note

This is a closed-book exam. Each problem accounts for 10 points, unless otherwise marked.

Problems

1. Reprove the following theorem which we have proven (mostly) in class. This time you must apply the *reversed induction* principle, or a variant of it, in some part of the proof.

   There exist Gray codes of length $\lceil \log_2 k \rceil$ for any positive integer $k \geq 2$. The Gray codes for the *even* values of k are *closed*, and the Gray codes for *odd* values of k are *open*.

2. Consider the following two-player game: given a positive integer N, player A and player B take turns counting to N. In his turn, a player may advance the count by 1 or 2. For example, player A may start by saying "1, 2", player B follows by saying "3", player A follows by saying "4", etc. The player who eventually has to say the number N loses the game.

   A game is *determined* if one of the two players always has a way to win the game. Prove that the counting game as described is determined for any positive integer N; the winner may differ for different given integers. You must use induction in your proof. (Hint: think about the remainder of the number N divided by 3.)

3. For each of the following pairs of functions, determine whether $f(n) = O(g(n))$ and/or $f(n) = \Omega(g(n))$. Justify your answers.

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

4. The Knapsack Problem is defined as follows: Given a set S of n items, where the i-th item has an integer size $S[i]$, and an integer K, find a subset of the items whose sizes sum to exactly K or determine that no such subset exists.

Now consider a variant where we want the subset to be as large as possible (i.e., to be with as many items as possible). How will you adapt the algorithm (see the Appendix) that we have studied in class? Your algorithm should collect at the end the items in one of the best solutions if they exist. Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary (you may reuse the code for the original Knapsack Problem). Give an analysis of its time complexity. The more efficient your algorithm is, the more points you will get for this problem.

5. Let x_1, x_2, …, x_n be a set of integers, and let $S = \sum_{i=1}^{n} x_i$. Design an algorithm to partition the set into two subsets of equal sum, or determine that it is impossible to do so. When the partitioning is possible, your algorithm should also give the two subsets of integers. The algorithm should run in time $O(nS)$.

6. Show all intermediate and the final AVL trees formed by inserting the numbers 6, 5, 4, 1, 2, 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 re-balancing operations are performed, please also show the tree before re-balancing and indicate what type of rotation is used in the re-balancing.

7. Let $G(h)$ denote the least possible number of nodes contained in an AVL tree of height h. Let us assume that the empty tree has height -1 and a single-node tree has height 0. Please give a recurrence relation that characterizes (fully defines) G. Based on the recurrence relation, prove that the height of an AVL tree of n nodes is $O(\log n)$.

8. The *Partition* procedure for the Quicksort algorithm discussed in class is as follows, where *Middle* is a global variable.

```
**Partition** $(X, Left, Right)$;
**begin**
    $pivot := X[left]$;
    $L := Left$;  $R := Right$;
    **while** $L < R$ **do**
        **while** $X[L] \leq pivot$ and $L \leq Right$ **do** $L := L + 1$;
        **while** $X[R] > pivot$ and $R \geq Left$ **do** $R := R - 1$;
        **if** $L < R$ **then** $swap(X[L], X[R])$;
    $Middle := R$;
```

```
        $swap(X[Left], X[Middle])$
    end
```

Find an adequate loop invariant for the main while loop, which is sufficient to show that after the execution of the last two assignment statements the array is properly partitioned by $X[Middle]$. Please express the loop invariant as precisely as possible, using mathematical notation.

9. Consider rearranging 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	7	1	5	2	14	6	11	8	4	10	15	13	12	9

   Please show the result (i.e., the contents of the array) after a new element is added to the current collection of heaps (at the bottom) until the entire array has become a heap.

10. Prove that the sum of the heights of all nodes in a complete binary tree with n nodes is at most $n-1$. You may assume it is known that the sum of the heights of all nodes in a *full* binary tree of height h is $2^{h+1}-h-2$. (Note: a single-node tree has height 0.)

Appendix

- The solution of the recurrence relation $T(n) = aT(n/b) + cn^k$, where a and b are integer constants, $a \geq 1$, $b \geq 2$, and c and k are positive constants, is as follows.

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } a > b^k \\ O(n^k \log n) & \text{if } a = b^k \\ O(n^k) & \text{if } a < b^k \end{cases}$$

- Below is an algorithm for determining whether a solution to the Knapsack Problem exists.

```
Algorithm Knapsack $(S, K)$;
begin
    $P[0,0].exist := true$;
    for $k := 1$ to $K$ do
        $P[0,k].exist := false$;
    for $i := 1$ to $n$ do
        for $k := 0$ to $K$ do
```

```
          $P[i,k].exist := false$;
          **if** $P[i-1,k].exist$ **then**
            $P[i,k].exist := true$;
            $P[i,k].belong := false$
          **else if** $k - S[i] \geq 0$ **then**
                **if** $P[i-1,k-S[i]].exist$ **then**
                  $P[i,k].exist := true$;
                  $P[i,k].belong := true$
**end**
```