Midterm

Note

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

Problems

1. Consider the geometric series: 1, 2, 4, 8, 16, ... Prove *by induction* that any positive integer can be written as a sum of distinct numbers from this series.

2. Prove *by induction* 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.)

3. In the so-called implicit representation of a binary tree, the tree nodes are stored in an array, say A, such that

   (a) the root is stored in $A[1]$ and

   (b) the left child of (the node stored in) $A[i]$ is stored in $A[2i]$ and the right child in $A[2i+1]$. (Note: a nonexistent child may be indicated by a special mark/value in the corresponding cell for storing the child.)

   Prove *by induction* that, for complete binary trees, the implicit representation is compact in the sense that, if we label the tree nodes from top to bottom and left to right with the numbers 1 through n (where n is the number of nodes in the tree), then the node labeled i is stored in $A[i]$ for $1 \leq i \leq n$.

4. (15 points) 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.

   (a) (5 points) Please give a recurrence relation that characterizes (fully defines) G.

   (b) (10 points) Based on the recurrence relation, prove that the height of an AVL tree with n nodes is $O(\log n)$.

5. (15 points) Consider the Knapsack Problem: 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.

   Below is a recursive version of the algorithm for determining whether a solution to the Knapsack Problem exists, where we have ignored the tag values for recording the subset of items that constitute the solution. The algorithm should be invoked with Knapsack(n, K).

```
**Algorithm Knapsack**$(m, k)$;
begin
    if $k = 0$ then return $true$;
    if $m = 0$ then return $false$;
    if Knapsack$(m - 1, k)$ then return $true$
    else if $k - S[m] \geq 0$ then return Knapsack$(m - 1, k - S[m])$
        else return $false$;
end
```

(a) (5 points) Given an input, Knapsack(m, k) may be invoked with the same combination of m and k at different points of execution. Why? Please give an example.

(b) (10 points) How will you propose to avoid duplicate invocations? Please revise the code to incorporate your proposal. (Hint: use an array to memorize the result of an invocation.)

6. Show all intermediate and the final AVL trees formed by inserting the numbers 6, 1, 2, 5, 4, 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. Apply the Quicksort algorithm to the following array. Show the contents of the array after each partition operation. If you use a different partition algorithm (from the one discussed in class), please describe it.

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

8. Please present in suitable pseudocode the algorithm (discussed in class) for rearranging an array $A[1..n]$ of n integers into a max heap using the *bottom-up* approach.

9. Below is a variant of the insertion sort algorithm.

```
Algorithm Insertion_Sort $(A, n)$;
begin
    for $i := 2$ to $n$ do
        $x := A[i]$;
        $j := i$;
        while $j > 1$ and $A[j - 1] > x$ do
            $A[j] := A[j - 1]$;
            $j := j - 1$;
        end while
        $A[j] := x$;
    end for
end
```

Draw a decision tree of the algorithm for the case of $A[1..3]$, i.e., $n = 3$. In the decision tree, you must indicate (1) which two elements of the original input array are compared in each internal node and (2) the sorting result in each leaf. Please use X_1, X_2, X_3 to refer to the elements (in this order) of the original input array.

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 a non-recursive 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
```