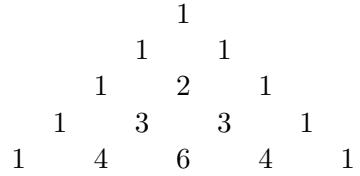
Suggested Solutions to Midterm Problems

1. Find an expression for the sum of the i-th row of the following triangle, which is called the **Pascal triangle**, and prove *by induction* the correctness of your claim. The sides of the triangle are 1s, and each other entry is the sum of the two entries immediately above it.



Solution. To be completed. □

2. 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 $1 \leq i \leq n$.

   Solution. We first observe that a complete binary tree of n nodes is obtained from another of $n-1$ nodes as follows. If node $n-1$ (according to the labeling from top to bottom and left to right) is the left child of a node, node n is simply added as the right child of the same node. If node $n-1$ is the right child of a node i ($i < n-1$), then node n is added as the left child of node $i+1$.

   The proof of compactness is by induction on n.

   Base case ($n = 1, 2$): when $n = 1$, the only element is the root which is labeled 1 and, according to the implicit representation, is stored in $A[1]$. When $n = 2$, the node labeled 1 is stored in $A[1]$ as in the case of $n = 1$. The node labeled 2 is the left child of node 1, i.e., $A[1]$, and according to the implicit representation, is stored in $A[2 \times 1]$, i.e., $A[2]$.

   Inductive step ($n > 2$): from the induction hypothesis and the observation stated in the beginning, the part of nodes 1 through $n-1$ are stored in $A[1..n-1]$ in the right order. We need to show that the last node, labeled n, will indeed be stored in $A[n]$ according to the implicit representation. There are two cases:

   When $n-1$ is even ($n-1 \geq 2$), node $n-1$, stored in $A[n-1]$ (from the induction hypothesis), is the left child of $A[\frac{n-1}{2}]$ (according to the implicit representation), i.e., node $\frac{n-1}{2}$ (from the induction hypothesis). Hence, node n should be the right child of node $\frac{n-1}{2}$ (or $A[\frac{n-1}{2}]$) and stored in $A[2 \times \frac{n-1}{2} + 1]$, i.e., $A[n]$.

When $n-1$ is odd ($n-1 \geq 3$), node $n-1$, stored in $A[n-1]$, is the right child of $A[\frac{n-1-1}{2}]$, i.e., node $\frac{n-1-1}{2}$. Hence, node n should be the left child of node $\frac{n-1-1}{2}+1$ (or $A[\frac{n-1-1}{2}+1]$) and stored in $A[2(\frac{n-1-1}{2}+1)]$, i.e., $A[n]$.

□

3. (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.

   Solution. The recurrence relation can be defined as follows:

$$\begin{cases} G(-1) & = 0 \\ G(0) & = 1 \\ G(h) & = G(h-1) + G(h-2) + 1, \quad h \geq 1 \end{cases}$$

   □

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

   Solution. A precise solution to $G(h)$ may be derived by establishing the relation $G(h) = F(h+3) - 1$, where $F(n)$ is the n-th Fibonacci number (as defined in Chapter 3.5 of Manber's book) for which we already know the closed form; the proof is in fact quite simple by induction. However, we will prove directly a lower bound for $G(h)$, namely $\Omega((\frac{3}{2})^h)$, which is good enough to show its exponential growth. The proof is by induction on h, showing that $G(h) \geq \frac{2}{3}(\frac{3}{2})^h$, for $h \geq 0$.
   Base case ($h = 0$ or $h = 1$): When $h = 0$, $\frac{2}{3}(\frac{3}{2})^0 = \frac{2}{3} \leq 1 = G(0)$. When $h = 1$, $\frac{2}{3}(\frac{3}{2})^1 = 1 \leq 2 = G(1)$.
   Inductive step ($h > 1$): $G(h) = G(h-1) + G(h-2) + 1$, which from the induction hypothesis $\geq \frac{2}{3}(\frac{3}{2})^{h-1} + \frac{2}{3}(\frac{3}{2})^{h-2} + 1 \geq (1+\frac{2}{3})(\frac{3}{2})^{h-2} = (1+\frac{2}{3})(\frac{3}{2})^{-2}(\frac{3}{2})^h = \frac{20}{27}(\frac{3}{2})^h \geq \frac{2}{3}(\frac{3}{2})^h$.
   Therefore, for an AVL tree of size n, its height h must be such that $\frac{2}{3}(\frac{3}{2})^h \leq G(h) \leq n$. It follows that $h \leq \frac{1}{\log 1.5}\log n + 1$ (base 2 logarithm), implying $h = O(\log n)$. □

4. (15 points) Consider the problem of merging two skylines, which is a useful building block for computing the skyline of a number of buildings. A skyline is an alternating sequence of x coordinates and y coordinates (heights), ending with an x coordinate (as discussed in class).

   (a) What is the resulting skyline from merging these two skylines: (1,**3**,3,**6**,6,**2**,8,**7**,10,**3**,14) and (2,**4**,6,**2**,8,**5**,11,**3**,12)?

   Solution. To be completed. □

   (b) To obtain a systematic procedure for merging two skylines, one may focus on determining the first two coordinate values to be output and reducing (by two) the length of one of the input skylines. Suppose the two input skylines are $(a_1, a_2, \cdots, a_{2m+1})$ and $(b_1, b_2, \cdots, b_{2n+1})$. How can the first two coordinate values in the output be determined?

   Solution. To be completed. □

(c) Following the thought in the previous subproblem, assume that the first two coordinate values that should be output are a_1 and a_2. We are now looking at two skylines $(a_3, a_4, \cdots, a_{2m+1})$ and $(b_1, b_2, \cdots, b_{2n+1})$, the first being shorter (by two) than in the original input. How can the next two coordinate values in the output be determined?

Solution. To be completed. □

5. 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.

We have discussed in class two approaches to implementing a solution that we designed by induction: one uses dynamic programming (see the Appendix), while the other uses recursive function calls.

Please present the recursive approach in suitable pseudocode.

Solution. 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(S, K, n).

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

□

6. Show all intermediate and the final AVL trees formed by inserting the numbers 7, 6, 3, 1, 4, 5, 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 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.

Solution. To be completed. □

7. The input is a set S with n real numbers. Design an $O(n)$ time algorithm to find a number that is *not* in the set. Prove that $\Omega(n)$ is a lower bound on the number of steps required to solve this problem.

Solution. When there is just one number (i.e., $n = 1$) in S, it is trivial to find a real number different from the number in S, e.g., by adding one to or subtracting one from the existing number. When there are exactly two numbers, we simply take their average which will be different from both numbers, since by the definition of a set, all numbers in S are distinct. When there are more than two numbers, we proceed as follows.

Store the first two numbers as a pair and read the remaining numbers one by one. When the next number read falls between the pair, we replace the smaller of the pair by the number just read. At the end we will have a pair of real numbers and none of the numbers

in S falls between the pair. We take the average of the pair which will be different any number in S.

We next argue for the lower bound $\Omega(n)$. This is quite straightforward, since every number in S must be read and there are n numbers. Any algorithm that skips a number may return a wrong result, as the result may happen to be equal to the number that is skipped. □

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

Solution. To be completed. □

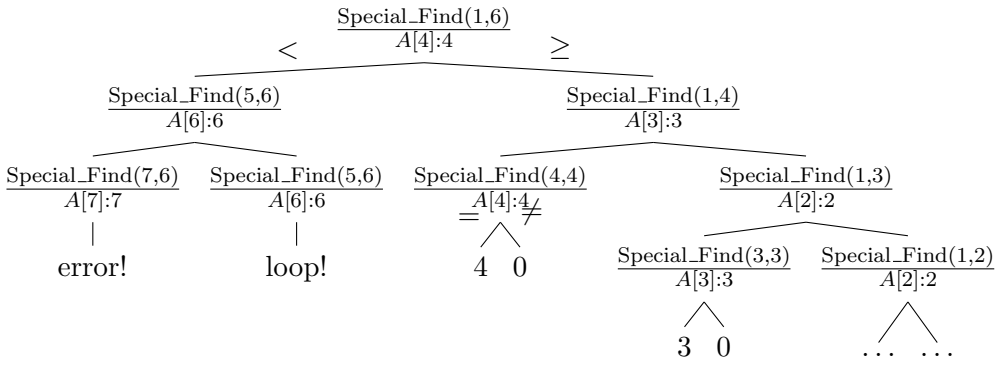
9. Consider the following algorithm that, given a sorted sequence of *distinct* integers $a_1, a_2, \cdots, a_n$ (stored in an array A), determines whether there exists an index i such that $a_i = i$.

```
Algorithm Special_Binary_Search (A, n);
begin
    Position := Special_Find(1, n);
end

function Special_Find (Left, Right) : integer;
begin
    if Left = Right then
       if A[Left] = Left then Special_Find := Left
       else Special_Find := 0
    else
        Middle := ⌈(Left+Right)/2⌉;
        if A[Middle] < Middle then
           Special_Find := Special_Find(Middle + 1, Right)
        else
           Special_Find := Special_Find(Left, Middle)
end
```

Draw a decision tree of the algorithm for the case of input size six, i.e., $n = 6$.

Solution. The given algorithm turns out to be erroneous (as was noticed by some of you during the exam). According to the algorithm, a decision tree for the case of $n = 6$ is as follows.



□

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 are several basic generating functions.

generating func.	power series	generated sequence
$\frac{1}{1-z}$	$1 + z + z^2 + \cdots + z^n + \cdots$	$1, 1, 1, \cdots, 1, \cdots$
$\frac{c}{1-az}$	$c + caz + ca^2z^2 + \cdots + ca^nz^n + \cdots$	$c, ca, ca^2, \cdots, ca^n, \cdots$
$\frac{1}{(1-z)^2}$	$1 + 2z + 3z^2 + \cdots + nz^{n-1} + \cdots$	$1, 2, 3, \cdots, n, \cdots$
$\frac{z}{(1-z)^2}$	$z + 2z^2 + 3z^3 + \cdots + nz^n + \cdots$	$0, 1, 2, 3, \cdots, n, \cdots$

- 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] ≥ 0 then
                    if P[i-1,k-S[i]].exist then
                       P[i,k].exist := true;
                       P[i,k].belong := true
end
```