Midterm

Note

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

Problems

1. Find the error in the following proof that all horses are the same color.

   CLAIM: In any set of h horses, all horses are the same color.

   PROOF: By induction on h.

   Basis ($h = 1$): In any set containing just one horse, all horses clearly are the same color.

   Inductive step ($h > 1$): We assume that the claim is true for $h = k$ ($k \geq 1$) and prove that it is true for $h = k+1$. Take any set H of $k+1$ horses. We show that all the horses in this set are the same color. Remove one horse from this set to obtain the set H_1 with just k horses. By the induction hypothesis, all the horses in H_1 are the same color. Now replace the removed horse and remove a different one to obtain the set H_2. By the same argument, all the horses in H_2 are the same color. Therefore all the horses in H must be the same color, and the proof is complete.

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. Summations whose exact values are hard to compute may be easily and tightly bounded by integrals. For example, if $f(x)$ is monotonically *decreasing*, then

$$\int_1^{n+1} f(x)dx \leq \sum_{i=1}^{n} f(i) \leq f(1) + \int_1^n f(x)dx.$$

   (a) Show that the bounds for $\sum_{i=1}^{n} f(i)$ are indeed correct.

   (b) Prove, using this bounding technique, that $\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$.

4. The Knapsack Problem that we discussed in class is defined as follows: Given a set S of n items, where the ith 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 described in class an algorithm to solve the problem. Modify the algorithm to solve a variation of the knapsack problem where each item has an *unlimited* supply. In your algorithm, please change the type of $P[i,k].belong$ into integer and use it to record the number of copies of item i needed.

5. Show all intermediate and the final AVL trees formed by inserting the numbers 1, 2, 5, 7, 4, 3, and 6 (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.

6. Below is the Mergesort algorithm in pseudocode:

```
Algorithm Mergesort (X, n);
begin  M_Sort(1, n)  end

procedure M_Sort (Left, Right);
begin
    if Right − Left = 1 then
       if X[Left] > X[Right] then swap(X[Left], X[Right])
    else if Left ≠ Right then
            Middle := ⌈½(Left + Right)⌉;
            M_Sort(Left, Middle − 1);
            M_Sort(Middle, Right);
            // the merge part
            i := Left;  j := Middle;  k := 0;
            while (i ≤ Middle − 1) and (j ≤ Right) do
                    k := k + 1;
                    if X[i] ≤ X[j] then
                       TEMP[k] := X[i];  i := i + 1
                    else TEMP[k] := X[j];  j := j + 1;
            if j > Right then
               for t := 0 to Middle − 1 − i do
                    X[Right − t] := X[Middle − 1 − t]
            for t := 0 to k − 1 do
                X[Left + t] := TEMP[1 + t]
end
```

   Given the array below as input, what are the contents of array *TEMP* after the merge part is executed for the first time and what are the contents of *TEMP* when the algorithm terminates? Assume that each entry of *TEMP* has been initialized to 0 when the algorithm starts.

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

7. The partition procedure in the Quicksort algorithm chooses an element as the pivot and divide the input array $A[1..n]$ into two parts such that, when the pivot is properly placed in $A[i]$, the entries in $A[1..(i-1)]$ are less than or equal to $A[i]$ and the entries in $A[(i+1)..n]$ are greater than or equal to $A[i]$. Please design an extension of the partition procedure so that it chooses two pivots and divides the input array into three parts. Assuming the two pivots are eventually placed in $A[i]$ and $A[j]$ $(i < j)$ respectively, the entries in $A[1..(i-1)]$ are less than or equal to $A[i]$, the entries in $A[(i+1)..(j-1)]$ are greater than or equal to $A[i]$ and less than or equal to $A[j]$, and the entries in $A[(j+1)..n]$ are greater than or equal to $A[j]$.

   Please present your extension in adequate pseudocode and make assumptions wherever necessary. Give an analysis of its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem.

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

   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.

9. Two computers, each with a set of n integers, try to collaboratively find the n-th smallest element of the union of the two sets. The two computers can communicate by sending messages and they can perform any kind of local computation. A message can contain one element or one integer; a message with two numbers should be counted as two messages. Design an algorithm for the search task so that the number of messages exchanged is minimized. You can assume, for simplicity, that all the elements are distinct.

   Please present your algorithm in an adequate pseudo code and make assumptions wherever necessary. Give an analysis of its message complexity (the number of messages exchanged). The more efficient your algorithm is, the more points you will get for this problem.

10. Below is a variant of the bubble sort algorithm in pseudocode.

```
Algorithm Bubble_Sort (A, n);
begin
    i := n;
    repeat
        swapped := false;
        for j := 1 to i − 1 do
            if A[j] > A[j + 1] then
                swap(A[j], A[j + 1]);
```

```
                swapped := true;
            end if
        end for
        i := i − 1;
    repeat until (not swapped)
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 (not $A[1]$, $A[2]$, $A[3]$) to refer to the elements (in this order) of the original input array.

Appendix

- Below is an algorithm for determining whether a solution to the (original) 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
```

- Below is an alternative algorithm for partition in the Quicksort algorithm:

```
Partition (X, Left, Right);
begin
    pivot := X[left];
    i := Left;
    for j := Left + 1 to Right do
        if X[j] < pivot then i := i + 1;
                             swap(X[i], X[j]);
    Middle := i;
    swap(X[Left], X[Middle])
end
```