Midterm

Note

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

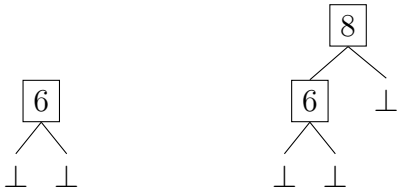
Problems

1. The set of all full binary trees that store non-negative integer key values may be defined inductively as follows.

   (a) $FBT(k, \perp, \perp, 0)$, for any non-negative integer k, is a full binary tree of height 0.

   (b) If t_l and t_r are full binary trees of height h, then $FBT(k, t_l, t_r, h+1)$, for any non-negative integer k, is a full binary tree of height $h+1$.

   Please give a similar inductive definition for the set of all complete binary trees (of the form $CBT(\cdot, \cdot, \cdot, \cdot)$) that store non-negative integer key values. For instance, $CBT(6, \perp, \perp, 0)$ is a single-node complete binary tree storing key value 6 and $CBT(8, CBT(6, \perp, \perp, 0), \perp, 1)$ is a complete binary tree with two nodes — the root and its left child, storing key values 8 and 6 repsectively. Pictorially, they may be depicted as below.



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. 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 (see the Appendix) to solve the problem. Please provide, for each of the following two requirements, a modification to the algorithm that meets the requirement.

(a) The values of $P[i,k].belong$ ($0 \leq i \leq n$ and $0 \leq k \leq K$) record the subset (if one exists) with the fewest items whose sizes sum to K.

(b) The type of $P[i,k].exists$ becomes integer and it gives the number of possible subsets of items from $S[1..i]$ whose sizes sum to exactly k. In this case, the values of $P[i,k].belong$ are not useful and can be omitted.

For each case, you may just indicate the changes that should be made to the original algorithm.

4. You are asked to design a schedule for a round-robin tennis tournament. There are $n = 2^k$ ($k \geq 1$) players. Each player must play every other player, and each player must paly one match per round for $n - 1$ rounds. Denote the players by $P_1, P_2, \ldots, P_n$. Output the schedule for each player. (Hint: use divide and conquer in the following way. First, divide the players into two equal groups and let them play within the groups for the first $\frac{n}{2} - 1$ rounds. Then, design the games between the groups for the other $\frac{n}{2}$ rounds.)

5. Consider the solutions to the union-find problem discussed in class. Suppose we start with a collection of ten elements: A, B, C, D, E, F, G, H, I, and J, and the following sequence of operations are performed: union(A,B), union(C,D), union(E,F), union(G,H), union(I,J), union(A,D), union(F,G), union(D,J), union(J,H).

   Assuming that both the balancing and the path compression techniques are used, draw (1) a diagram showing the grouping of these ten elements immediately after union(F,G) is performed and (2) another after the whole sequence of operations are performed. In the case of combining two groups of the same size, please always point the second group to the first.

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

7. Draw a decision tree of the Mergesort 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 A.

8. Construct a Huffman code tree for a text composed from seven characters A, B, C, D, E, F, and G with frquencies 18, 10, 3, 8, 24, 4, and 10 respectively. And then, list the codes for all the characters according to the code tree.

9. The *next* table is a precomputed table (for $B = b_1 b_2 \cdots b_m$) that plays a critical role in the KMP algorithm. Under what condition (regarding $b_1 b_2 \cdots b_i$) does $next[i]$ (for $i > 0$) get a 0? And under what condition can it be safely set to -1 (without missing a potential match)?

10. The Fibonacci word sequence of bit strings is defined as follows:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) \cdot F(n-2) & \text{if } n \geq 2 \end{cases}$$

Here $\cdot$ denotes the operation of string concatenation. The first six Fibonacci words (from $F(0)$ to $F(5)$) are: 0, 1, 10, 101, 10110, 10110101.

Design an algorithm that, given a bit pattern p and a number n, determines whether p occurs in $F(n)$. For instance, 1101 occurs in $F(5)$, but not $F(4)$. Please present your algorithm in adequate pseudocode. Explain why it is correct and give an analysis of its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem.

Appendix

- 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
```

- The Mergesort algorithm:

```
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
```

- The KMP algorithm (assuming *next*):

```
Algorithm String_Match (A, n, B, m);
begin
    j := 1;  i := 1;
    Start := 0;
    while Start = 0 and i ≤ n do
        if B[j] = A[i] then
            j := j + 1;  i := i + 1
        else
            j := next[j] + 1;
            if j = 0 then
                j := 1;  i := i + 1;
        if j = m + 1 then Start := i − m
end
```

- The algorithm for computing the *next* table in the KMP algorithm:

```
Algorithm Compute_Next (B, m);
begin
    next[1] := −1;  next[2] := 0;
    for i := 3 to m do
        j := next[i − 1] + 1;
        while B[i − 1] ≠ B[j] and j > 0 do
            j := next[j] + 1;
        next[i] := j
end
```