Midterm

Note

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

Problems

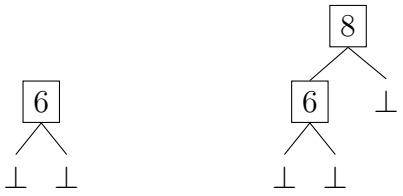
1. Prove *by induction* that every natural number greater than or equal to 12 is a non-negative linear combination of 4 and 5, i.e., for every $n \in \mathbb{N}$, if $n \geq 12$, then there exist $a, b \in \mathbb{N}$ s.t. $n = 4a + 5b$ (where $\mathbb{N}$ is the set of all natural numbers, including 0).

2. 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; you may reuse FBT in parts of your definition. 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.



3. Consider bounding summations by integrals. We already know that, if $f(x)$ is monotonically *increasing*, then

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

(a) The sum may also be bounded from below as follows:

$$\int_0^n f(x)dx \leq \sum_{i=1}^{n} f(i).$$

Show that this is indeed the case.

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

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

5. Below is the pseudocode of the binary search algorithm we discussed in class. Would the code still be correct if we change the assignment "$Middle := \lceil \frac{Left+Right}{2} \rceil$" to "$Middle := \lfloor \frac{Left+Right}{2} \rfloor$" for *Middle* to take instead the largest integer less than or equal to $\frac{Left+Right}{2}$? Please justify your answer. If the modified code is incorrect, what other changes must be made accordingly?

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

Algorithm Binary_Search (X, n, z);
begin
    Position := Find(z, 1, n);
end
```

6. Given the array below as input [to the Mergesort algorithm], 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
8	3	2	6	5	9	10	7	1	12	4	11

7. 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.

8. We have studied in class an algorithm, outlined again below, for finding the minimum and the maximum of a sequence of numbers.

   > Compare the first two numbers (assuming the input sequence is of even length). Set *min* to be the smaller of the two and *max* the larger. Compare the next pair of numbers and then compare the smaller with *min* and the larger with *max*. Update *min* and *max* accordingly. Continue until we have exhausted the sequence.

   Draw a decision tree of the algorithm for the case of an input sequence of four *distinct* numbers. In the decision tree, you must indicate (1) which two elements of the original sequence are compared in each internal node and (2) the output (the values of *min* and *max* respectively) in each leaf. Please use X_1, X_2, X_3, X_4 to refer to the numbers (in this order) in the original input sequence.

9. Consider the text data compression problem we have discussed in class; the problem statement is given below.

   > Given a text (a sequence of characters), find an encoding for the characters that satisfies the prefix constraint and that minimizes the total number of bits needed to encode the text.

   Prove that the two characters with the lowest frequencies must be among the deepest leaves (farthest from the root) in the final code tree. (Hint: proof by contradiction.)

10. Consider the *next* table as in the KMP algorithm for string $B[1..9] = abaababaa$.

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	0	1	1	2	3	2	3

    Suppose that, during an execution of the KMP algorithm, $B[6]$ (which is an a) is being compared with a letter in A, say $A[i]$, which is not an a and so the matching fails. The algorithm will next try to compare $B[next[6]+1]$, i.e., $B[3]$ which is also an a, with $A[i]$. The matching is bound to fail for the same reason. This comparison could have been avoided, as we know from B itself that $B[6]$ equals $B[3]$ and, if $B[6]$ does not match $A[i]$, then $B[3]$ certainly will not, either. $B[5]$, $B[8]$, and $B[9]$ all have the same problem, but $B[7]$ does not.

    Please adapt the computation of the *next* table, so that such wasted comparisons can be avoided. Also, please give the values of the *next* table for the string $B[1..9] = abbaabbaa$, according to the adaptation.

Appendix

- 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
```