**Instructions.** Collaboration is permitted in groups of size at most three. You must write the names of your collaborators on your solutions and must write your solutions alone.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

(20) 1. **Fibonacci Numbers**

Suppose instead of using powers of two, we now represent integers as the sum of Fibonacci numbers. That is, rather than representing a number as an array of bits, we keep an array of “fibbits” so that \( (x_kx_{k-1} \ldots x_1)_F \) denotes the number \( \sum_{i=1}^{k} x_i F_i \). As an example, the Fibonacci number \( (1101)_F = F_4 + F_3 + F_1 = 1 + 2 + 3 = 6 \). Recall that the Fibonacci numbers satisfy the recurrence \( F_0 = 0, F_1 = 1 \) and \( F_{n+2} = F_{n+1} + F_n \).

a) Show that every positive integer \( n \) can be represented as a Fibonacci number.

b) Give an algorithm to increment a Fibonacci number in constant amortized time.

(20) 2. **Variation of a Fibonacci Heap**

The maximum degree of any node in an \( n \)-node Fibonacci heap is \( O(\log n) \). Suppose that we modified the cut rule for Fibonacci heaps, so that instead of cutting a node from its parent as soon as it lost its 2nd child, we cut the node from its parent as soon as it lost its \( k \)th child (for some integer constant \( k \)). For what values of \( k \) is the maximum degree still \( O(\log n) \)?

(20) 3. **Scanning a BST**

Given a BST \( T \) of size \( n \), give an efficient algorithm for accessing its keys in order that uses \( O(1) \) space. The upper bound on the number of comparisons should hold independent of the tree \( T \).
4. **Maintaining a List with Reversals**

Consider a data structure that represents an ordered list of elements under the following three types of operations:

- **access**\((k)\): Return the \(k\)th element of the list (in its current order).
- **insert**\((k, x)\): Insert \(x\) (a new element) after the \(k\)th element in the current version of the list.
- **reverse**\((i, j)\): Reverse the order of the \(i\)th through \(j\)th elements.

For example, if the initial list is \([a, b, c, d, e]\), then \(\text{access}(2)\) returns \(b\). After \(\text{reverse}(2, 4)\), the represented list becomes \([a, d, c, b, e]\), and then \(\text{access}(2)\) returns \(d\).

Show how to modify splay tree construction so that each operation runs in \(O(\log n)\) amortized time, where \(n\) is the (current) number of elements in the list. The list starts out empty.

Hint: First consider how to implement access and insert using splay trees. Then think about a special case of reverse in which the \([i, j]\) range is represented by a whole subtree. Use these ideas to solve the real problem. Remember, if you store extra information in the tree, you must state how this information can be maintained under various restructuring operations.

5. **Potential Function for Splay Trees**

In analyzing splay trees, we used the potential function \(\Phi(T) = \sum_{x \in T} \log(s(x))\) where \(s(x)\) is the number of nodes (inclusive) in the subtree rooted at \(x\).

Let \(\mathcal{T}_n\) denote the family of all distinct BSTs of size \(n\). Calculate

\[
\sum_{T \in \mathcal{T}_n} \exp(-\Phi(T)).
\]