**Problem 1  Hash Functions I [5+4+4=13 marks]**

Assume that we have a hash table of size $M = 5$, we use the hash function $h(k) = k \mod 5$, and we use chaining for collision resolution. Furthermore, assume that our universe is $U = \{0, 1, 2, \ldots, 12\}$.

a) Demonstrate the insertion of the keys $0, 1, 2, \cdots, 12$ into the (initially empty) hash table (in that order). You just need to draw the state of the hash table after all insertions are done.

Answer: This is how the hash table looks like (when, for example, we apply chaining).
b) Is the uniform hashing assumption true for this example? Why or why not? **Answer:** Uniform hash assumption is true if items from the universe are inserted uniformly in the hash table. One way to look at it is that, if all items in the universe are inserted into the hash table, is it that all indices receive an equal number of items. If they do, the assumption holds; otherwise it does not.

For this particular question and universe universe, there is no way to insert all 12 items evenly into a hash table of size 5 (while it is possible for a hash table of size, e.g., 6). So, the uniform has assumption does not hold. Particularly, indices 0,1, and 2 receive more numbers than 3 and 4 on expectation.

The assumption is valid, however, when universe contains all integers. Note that the ordering of insertions has nothing to do with the uniform hash assumption (it assumes ‘all’ items in the universe are inserted to the table and then checks if they are inserted uniformly)

**Marking Scheme:** If you mention that the assumption is invalid (because of the above reason), you get the full mark. If you mention the assumption is true because the items are ‘almost’ uniformly distributed, you will get a partial mark up to 3 marks. If you mention the assumption does not hold but provide a wrong reason, you will get at most 2 out of 4 marks. Wrong reason includes mentioning of the ordering of insertions.

c) [bonus] Suppose only two insertions of elements in $U$ are made into the initially-empty hash table. If each pair of elements in $U$ is equally likely to be inserted, show that the probability that the second insertion caused a collision is $11/78$. Put a different way, given distinct $k_1$ and $k_2$ uniformly chosen from $U$, show that the probability that $h(k_1) = h(k_2)$ is $11/78$.

**Answer:** There are $\binom{13}{2} = 78$ ways to select two items from the universe. The number of ways they can be equal is $3 \times \binom{3}{2} = 9$ (when they both map to indices 0, 1, or 2) plus $2 \times \binom{2}{2} = 2$ (when they both map to indices 3 or 4). So, the chance of two elements being equal is $\frac{9+2}{78}$.

### Problem 2 Hash Functions II [6 marks]

Assume a hash scheme in which keys are selected uniformly at random from the Universe set $U = \{1, 2, 3, \ldots, 600\}$. Consider the following two hash functions: $h_1(k) = k \mod 6$ and $h_2(k) = 3k \mod 6$ Which hash function is better? Justify your answer.

**Answer:** The first function is much better. For any number $k$, the value of $3k \mod 6$ is either 0 or 3. So, this function maps all items in these two indices while other indices are left empty.

In general, a good hash function requires to depend on all parts of the input (that’s why we want hash tables have prime sizes; because division by a prime some how makes all digits involved). I contrast, multiplying keys by 3 in the above example causes overlooking parts of input.

**Marking Scheme:** You get 3 marks for correct answer and 3 for correct justification.

### Problem 3 Hash Functions III [6+6=12 marks]

Consider a hash table dictionary with universe $U = \{0, 1, 2, \ldots, 24\}$ and size $M = 5$. If items with keys $k = 21, 3, 16, 1$ are inserted in that order, draw the resulting hash table if we resolve collisions using:

a) Linear probing with $h(k) = (k + 1) \mod 5$

b) Cuckoo hashing with $h_1(k) = k \mod 5$ and $h_2(k) = \lfloor k/5 \rfloor$ **Answer:** see the figures below.
For Cuckoo hashing, the following steps are taken:

- 21 is placed at \( h_1(21) = 1 \).
- 3 is placed at \( h_1(3) = 3 \).
- 16 is placed at \( h_1(16) = 1 \). So, 21 is kicked out from index 1 and is placed at \( h_2(21) = 4 \).
- 1 is placed at \( h_1(1) = 1 \). So, 16 is kicked out from index 1 and is placed at \( h_2(16) = 3 \). So, 3 is kicked out from index 3 and is placed at index \( h_2(3) = 0 \).

**Marking Scheme:** We will look at the final hash table and deduce marks for each mistake.

**Problem 4  Binary Tree Traversals [5+5+5 = 15 marks]**

**a)** Assume you are given the pre-order and in-order traversals of a binary tree \( T_1 \). Note that \( T_1 \) is not necessarily a BST. Explain how we can reconstruct \( T_1 \) given these two traversals.  

**Answer:** (scheme) The first item (index \( lo \)) in the pre-order traversal indicates the root. Provide with this, we find the index of the root at the inorder traversal; let that index be \( i \). So, vertices at indices \([lo..i-1]\) from inorder traversal and indices \([lo+1..i+1]\) in the pre-order traversal form the left subtree. We recursively form this subtree. Similarly, vertices \([i+1..hi]\) in both inorder and pre-order traversal indicate vertices on the right subtree; so, we can recursively form the right subtree. Provided with the left and right subtree, we can just create a vertex with root’s key and let it point to the left and right subtree. The base of reduction is when \( lo = hi \) (in which case we just create a tree with one node).

**Marking Scheme:** 1 mark for mentioning the root being the first index. 2 mark for mentioning that we can construct the left subtree by looking at index of the root in pre-order (it is essential to mention the index). 2 marks for other details (including the mention of recursion).

**b)** Assume you are given the pre-order and post-order traversal of a binary tree \( T_2 \). Note that \( T_2 \) is not necessarily a BST. Provide an example that shows \( T_2 \) cannot be reconstructed using these traversals. For that, you need to show two different trees with the same pre- and post-order traversals.

**Answer:** Consider the following two trees. In both, the pre-order traversal is \{ a, b, c \} and post-order traversal is \{ c, b, a \}.
Marking Scheme: Basically, any two ‘chain-like’ have similar pre- and post-order traversals. There are many such examples that you can provide. But an example is required (a simple explanation does not suffice).

c) Assume $T_3$ is a binary search tree and its post-order traversal is given. Can we reconstruct $T_3$ from this traversal? justify your answer.

Answer: The answer is similar to part (a). Given the post-order traversal of a BST, we know its root is the last element. Provided with the root we can separate the left subtree (those smaller than the root) from the right subtree (those larger than the root). This involves two recursive calls, one with input parameters $\text{[lo}, i \text{]}$ (for creating the left subtree) and one with $\text{[i+1,hi-1]}$ (for creating the right subtree). Here $i$ is the index of the predecessor of the root in the array.

Problem 5  BST operations $[4+4+6 = 14 \text{ marks}]$

This problem will concern operations on the binary search tree shown in the following figure.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{bst_tree.png}
\end{figure}

a) Indicate what nodes are visited when we apply operation $\text{search}(19)$  \hspace{1cm} Answer: $50, 17, 23, 19$

a) Draw the tree after performing operations $\text{insert}(49)$ and then $\text{insert}(13)$.  \hspace{1cm} Answer: See the following figure.
b) Draw the original tree after performing operations delete(9) and then delete(50).

Answer: See the following figures. Note that depending on switching 50 with its successor or predecessor any of the following trees can be achieved (and both are fine).

![Diagram showing original tree after operations](image)

Problem 6 Insertion in BSTs [5 marks]

We saw in the class that rotations can be used to keep a tree balanced (short and fat). A clever student named Bob claims that it is possible to design a binary search tree that balances itself in a way that inserting any item takes time asymptotically less than \( \log n \). Prove that Bob’s claim is wrong.

Hint: use the lower bound for comparison-based sorting.

Answer: Preferred solution: For the sake of contradiction, let’s assume Bob’s claim is correct. Given an array of \( n \) items, we can insert them, one by one, into an initially empty binary search tree of Bob. According to Bob’s claim, this can be done in time asymptotically less than \( O(\log n) \), that is, all items are inserted in time asymptotically less than \( O(n \log n) \). Then we can in-order traverse the tree, in \( O(n) \), to get the sorted array. So, we could sort an input array in time less than \( O(n \log n) \) which we know is not possible. So, our initial assumption was wrong and insertion into a binary search tree takes at least \( O(\log n) \) time.

Alternative proof: a perfectly balanced BST on \( n \) nodes has height at least \( \log n \). So, an adversary can always select an item that needs to be inserted as a child of a leaf with depth \( \log n \). This requires at least traversing the height of the tree. In this proof, you should mention that a very certain insert causes the time complexity being at least \( \log n \).
Problem 7  2-3 Trees [5+6+6=17 marks]

This problem will concern operations on the 2-3 tree \( T \) shown in the following figure.

\[
\begin{align*}
&T \quad \text{50} \\
&\quad \text{10|30} \\
&\quad \quad \text{3|7} \quad \text{20} \quad \text{40} \\
&\quad \quad \quad 1 \quad 4|6 \quad 8 \quad 15 \quad 26 \quad 39 \quad 45|48 \\
&\quad \quad \quad 60 \quad 66 \quad 75|76 \quad 90|95 \\
&\quad \quad \quad 70 \quad 82
\end{align*}
\]

a) Draw the tree after performing operations \text{insert(49)} and \text{insert(5)}.  
Answer:  See the following figure.

\[
\begin{align*}
&T \quad \text{10|50} \\
&\quad \text{5} \\
&\quad \quad \text{3|7} \quad \text{20} \quad \text{40|48} \\
&\quad \quad \quad 1 \quad 4|6 \quad 8 \quad 15 \quad 26 \quad 39 \quad 45|49 \\
&\quad \quad \quad 60 \quad 66 \quad 75|76 \quad 90|95 \\
&\quad \quad \quad 70 \quad 82
\end{align*}
\]

b) Draw the \textit{original tree} after performing operations \text{delete(76)} and then \text{delete(55)}.  
See the following figure.

\[
\begin{align*}
&T \quad \text{30} \\
&\quad \text{10} \\
&\quad \quad \text{3|7} \quad \text{20} \\
&\quad \quad \quad 1 \quad 4|6 \quad 8 \quad 15 \quad 26 \quad 39 \quad 45|48 \\
&\quad \quad \quad 40 \quad 49 \\
&\quad \quad \quad 70 \quad 82 \\
&\quad \quad \quad 75 \quad 90|95
\end{align*}
\]

Answer:  Here are some details for those of you who are still confused about delete in the 2-3 trees. For delete, first we swap the target item with its predecessor or successor so that we can delete the node as a leaf. In this case, the deleted nodes are already leaves. After deleting 76, its node still has a key (75) and is not underloaded. So we can move on. Deleting 55 causes its node to have no key; looking at the direct sibling, we see it has only one key (66). So, we merge the two siblings, borrowing their parent’s key. This creates a new node with keys 60 and 66. The parent who borrowed 60 is now underloaded. To fix this, we look at its direct sibling and we see it only has one key (82). So, we merge them again and borrow node 70 from the parent. the result will be a
node with keys 70 and 82. Now the parent who borrowed 70 is underloaded. Looking at its sibling, we see it has two keys (10, 32). So, the parent borrows its key 50 to the empty and borrows a key 30 from the direct sibling. The pointer to 40 is updated to preserve the 2-3 tree structure.

**Marking Scheme:** The TAs will look at your final trees; if they are correct you get the complete mark; otherwise, you get zero or partial marks depending on how close to the correct solution your tree is.

c) Consider an arbitrary 2-3 tree of height $h$. Provide exact upper and lower bounds for the number of KVPs in such a tree (as a function of $h$)? Justify your answer. **Answer:**

**Lower bound:** The number of KVPs in a 2-3 tree is minimized when every node has exactly 1 key and 2 children. In this case, the number of nodes at level $i$ will be $2^i$ and the total number of nodes will be $1 + 2 + \ldots + 2^h = 2^{h+1} - 1$. Since there is one key per node, this gives a lower bound of $2^{h+1} - 1$ for the number of keys in a 2-3 tree of height $h$.

**Upper bound:** The number of KVPs in a 2-3 tree is minimized when every node -including the root- has exactly 2 key and 3 children. In this case, the number of nodes at level $i$ will be $3^i$ and the total number of nodes will be $1 + 3 + \ldots + 3^h = (3^{h+1} - 1)/2$. Since there are two keys per node, this gives an upper bound of $(3^h - 1)/2 \times 2 = 3^{h+1} - 1$ for the number of keys in a 2-3 tree of height $h$.

**Marking Scheme:** if you provide a right formula and justification, you will get the full mark; solving the formula to get the closed formula is not important.