Please write your name on every page and enter your serial number in the box above. If you miss out any of these: -1 and no rechecking.

No pseudocode means: For every loop you use, you must explain what it will do to the input, you cannot write “for i = 1 to ...” or “while <condition>...”. You cannot write things like “x = x + y”, you must explain the significance of every step in words. In summary: if we need to interpret how your description will treat a particular input then it is pseudocode.

Q1. (Tree + heap = Treap. Total marks = 5).

Let us suppose we are given a set $S$ of the following 10 items, each is a tuple with two values, the first a key, the second a priority:

$$S = \{(22, 7), (20, 2), (2, 9), (10, 3), (35, 10), (40, 11), (15, 6), (30, 8), (25, 4), (5, 5)\}.$$

A Treap is a special kind of binary search tree which also has the property of a heap. Each item in a treap is a tuple of two values i.e. $x_i = (k_i, p_i)$ where $k_i$ is a key and $p_i$ is a priority. A treap is a binary search tree on the keys of the items and maintains the min-heap order property on the priorities i.e. the priority value of a node is less than the priority value of any children it may have. Note that the Treap does not have to have the structural property of heap i.e., it may not be a complete binary tree but must have the order property of heap.

Q1.1 (2 marks). Construct a Treap on the set $S$ given above. You do not need to show the steps, only the final treap.

![Treap Diagram]

Marks. For every minor error deduct 0.5 marks. If the basic structure is correct then deduct at most 1 mark for minor errors.
Q1.2 (3 marks). Given a set of key-priority pairs, like $S$ above, give an algorithm to construct a treap for $S$. Note, you are to assume the entire set is given in advance. No pseudocode.

**** Solution ****

A simple recursive procedure for this is as follows.

- The base case is: if there is only one pair then create a one-node treap containing that pair in the root. Otherwise,
- Extract the pair with the min priority value. Let’s say it is $(k, p)$. This will be the root of the treap.
- Separate the remaining pairs into two sets: those with keys $\leq k$ and those with keys $> k$.
- Recursively create a treap with the first of these two sets and make it the left subtree of $(k, p)$. Similarly, recursively create a treap with the pairs whose keys are $> k$ and make it the right subtree of the root $(k, p)$.

A much harder solution would involve putting the keys in one at a time and doing a BST insertion followed by adjustment to fix the heap order property. In case such a solution is offered you will have to read it carefully to see if it is correct.

Marks. Give at least 1.5 marks if the basic idea of starting with the minimum priority is there.

Q2. (Hastables. Total marks = 5) [Marks: ]

In this question we look at an open addressing scheme called *cuckoo hashing*. The set of keys we are trying to store is a subset of the natural numbers and we have a hash table $T$ of size $n$. We are given two hash functions $h_1$ and $h_2$ which map all natural numbers to the set $\{0, 1, \ldots, n-1\}$. In order to place a newly inserted key $x$ into the table we do the following:

1. Compute $l_1 = h_1(x)$.
2. If $T[l_1]$ is free then store $x$ in $T[l_1]$ and exit, else
3. Compute $l_2 = h_2(x)$.
4. If $T[l_2]$ free then store $x$ in $T[l_2]$ and exit else if $y$ is currently stored in $T[l_2]$. Remove $y$ from $T[l_2]$ and store $x$ there.
5. Now we need to find an alternate position for $y$. If $l_2$ was $h_1(y)$ then compute $l_3 = h_2(y)$ and go to step 4 else if $l_2$ was $h_2(y)$ then compute $l_3 = h_1(y)$ and go to step 4. The fifth time we have to go to step 4 we assume a cycle has occurred and an exception is thrown declaring the insertion unsuccessful.

Q2.1. (1 mark) Given a key $x$, explain in words how to check if $x$ is in the hash table.

**** Solution ****

Simply check both positions $h_1(x)$ and $h_2(x)$ and see if $x$ is in any one of them.
Name: 

![Cuckoo Hash Table with n = 11 and keys inserted.](image)

**Figure 1:** A cuckoo hash table with \( n = 11 \) and the keys 69, 58, 49, 18 and 89 inserted.

**Marks:** No partial credit. 0 or 1.

**Q2.2. (4 marks)** Suppose we are given \( n = 11 \) and \( h_1 = (x + 1) \mod n \) and \( h_2 = (2x + 5) \mod n \) and the following sequence of insertions starting from an empty table: 69, 58, 49, 18, 89, 55, 56, 28 is performed. The table after the insertions of 69, 58, 49, 18, 89 is shown in Figure 1. Show the state of the hash table after each of the following insertions 55, 56 and 28 using cuckoo hashing. Note you only have to show the state of the array at the very end of each operation, not the intermediate movements. In case you detect a cycle (i.e. you throw an exception and abort the insertion) please show the state of the hash table at the 5th time the “go to” statement is executed and indicate the value \( y \) which was last displaced.

![Cuckoo Hash Table Solution](image)

**Marks:** For the first two it is 1 mark each. These are 0/1 questions, no partial credit. For the third one see partial credit suggestion as above.

![Additional Cuckoo Hash Table](image)
Q3. (Binary Search Trees. Total marks = 6) We have a Binary Search Tree implementation in which each node contains an extra value along with the key that we call special. This is some pseudocode for insertion into this BST:

```plaintext
function insert(T, x)
    temp = CreateNode(x)
    if (T is empty)
        Set temp.special = 1
        Set root of T to be temp
    else
        curr = getRoot(T)
        Set curr.special = (curr.special + 1) mod 2
        if (x <= curr.key)
            if curr.leftSubtree.isEmpty()
                Set left child of curr to be temp
            else
                insert(curr.leftSubtree, x)
        else
            if curr.rightSubtree.isEmpty()
                Set right child of curr to be temp
            else
                insert(curr.rightSubtree, x)
```

Q3.1. (1 mark) What does the special value stored at a node represent?

**** Solution ****

It stores 1 if the number of nodes in the subtree (including the root) is odd and 0 otherwise. Alternately, we can say that it stores 1 if the number of nodes in the subtree (excluding the root) is even and 0 otherwise. Note that the procedure above has an error. In the case where curr is a leaf, we should have set the special flag before inserting, i.e., where I wrote:

```plaintext
if curr.leftSubtree.isEmpty()
{
    Set left child of curr to be temp
}
```

I should have written:

```plaintext
if curr.leftSubtree.isEmpty()
{
    Set temp.special = 1
    Set left child of curr to be temp
}
```

and similarly for the right subtree case.

Marks. We can be lenient here. The main thing is the understanding that it is distinguishing odd and even in the subtrees. If someone has pointed out the error and not given any other answer then we can give full marks.
Q3.2. (5 marks) Describe in words (No pseudocode) the delete function that removes key $x$ from a tree $T$ while maintaining the special value at every node.

**** Solution ****

1. The delete function, like in the normal case, begins with a find operation. Except here, we flip the value of special at each node we visit (since the number of nodes in its subtree is going to decrease by 1).

2. When we find $x$ we have to consider the three cases involved in deleting from a BST

   (a) $x$ is a leaf. Simply delete $x$ and exit.

   (b) $x$ is an internal node with one child (say $y$). Remove $x$ and attach $y$ to the parent of $x$, i.e., set parent($y$) to parent($x$) and exit.

   (c) $x$ has two children.

      • First we flip the value of special at the current node (the one whose data value is $x$).

      • In this case we have to go down the tree to either the inorder predecessor or inorder successor of $x$. The important thing is that as we go down we have to flip the value of special at every node we visit since the deletion is going to happen at below the nodes we visit.

      • When we reach the inorder successor (which is reached by going to the right child of $x$ and then continuing leftwards till we cannot go left anymore) then we take its data value, say $z$, and put it in the node where the value $x$ used to be. Then we delete the node where $z$ used to be, which will be either of the earlier two cases. For inorder predecessor we do the same thing except we first go left then continuously right. They have to do either one or the other.

Marks. Delete 1 mark if they have not flipped special on the way down as they find $x$. Delete 0.5 if they have omitted the leaf deletion case or written it wrong, similarly delete 0.5 for the one child case. Delete 1.5 if they have not traversed down the tree flipping special on the way down in the two children case. Note that these instructions are for the case where the procedure is basically correct. If they do not have the basic BST deletion procedure correct then give at most 2 marks depending on what has been written. Note one can also delete a first and then travel upselection special.

Q4. (Short answer. Total marks = 4).

The following statements are false. You have to construct an example that proves that they are false.

Q4.1. (1 mark) Given 15 priority values, 8 of which are 0 and 7 of which are 1, it is not possible to organise them in a min-heap such that there is a priority value 1 in a node of depth 1 (i.e. a child of the root).

**** Solution ****

![Diagram of a min-heap with a root node and two child nodes, one with a value of 0 and the other with a value of 1, illustrating the impossibility of having a priority value 1 in a node of depth 1.]
Q4.2. (1 mark) The pre-order traversal of a binary search tree is never in sorted order. (Note: Your example must have at least 4 keys).

Marks. This is 0/1. No partial credit.

Q4.3. (2 marks) It is not possible to make an AVL tree on 10 or more nodes such that the left subtree of the root contains less than one-third of the nodes that the right subtree contains.

Marks. This is 0/2. No partial credit.