Algorithms -1
Tutorial 2
Solution Sketch

1. Insert the numbers 39, 33, 26, 7, 14, 3, 71, 73, 67, 65 in this order in an initially empty R-B Tree. Draw the tree after each insertion. (Practice more such examples).

   Routine.

2. You are given with keys 10, 22, 31, 4, 15, 28, 17, 88, 59. You need to insert these keys into a hash table of length $m = 11$ using open addressing with the hash function $h(k) = k \bmod m$. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $c_1 = 1$ and $c_2 = 3$, and using double hashing with $h_1(k) = k \bmod m$ and $h_2(k) = 1 + (k \bmod (m - 1))$. Now, repeat the same procedure for $m = 22$. (Practice more such examples).

   Routine.

3. Given a singly linked list $A_1, A_2, A_3, A_4,...,A_{n-2}, A_{n-1}, A_n$, design an algorithm to convert it to the singly linked list $A_1, A_n, A_2, A_{n-1}, A_3, A_{n-2},...$ and so on in O(n) time and O(1) space.

   H = head pointer of the list
   M = pointer to the middle element of the list (this can be found with two traversals, one to know the number of nodes in the list and the second to go to the middle).
   prevM = pointer to the node previous to M (can be remembered while finding M)
   Reset next pointer of prevM to NULL
   So now there are two lists, one with head pointer H and one with M, with n/2 elements in each
   Traverse the list H, inserting one element from M in between two elements of H (easy)

4. Given an array of n integers, design an O(n) time algorithm to find the next larger element for every element (next larger element of an element x is the first element that occurs after x in the array that is larger than x). (Hint: Use a stack).

   Keep stack S. Assume that you have a function top(S) that gives the top-of-stack element without removing it (easy to implement with one pop() and then push())
   Push first element in stack.
   For all elements x from 2nd to last
   If (stack not empty and top(S) ≥ x)
   push(S, x)
   Else while (stack not empty and top(S) < x)
   Y= pop(S);
   Print x as next larger element of y
   push(S, x)
   While (S not empty)
   y = pop(S);
   printf -1 as next larger element of y

5. In a binary tree, there is exactly one path between any two nodes (a path between two nodes informally is any sequence of edges that you can follow from one node to the other in the graphical representation of the tree). The path-sum of a path is the sum of the values of the nodes in the path. Design an O(n) time algorithm to find the maximum path-sum of any path in a given binary tree.

   For every node x, store the following values:
   (a) The value at the node
   (b) The length of the maximum path sum through left child + this node
   (c) The length of the maximum path sum through right child + this node
   (d) The length of the maximum path sum through left child + this node + max path through right child

   No think how you can use them. Program this, it is interesting. Note that the maximum path-sum path in a binary tree need not go through root, so your recursive calls will need to pass back something to take care of this.

6. A Cartesian tree is a binary tree such that the value at any node x is greater than the value of any other node in the subtree rooted at x. Given an inorder traversal of a Cartesian tree, construct the tree. Is the tree unique? Justify your answer.

   Do this recursively
   1. Find the max element in the input array
   2. Set max as the root
   3. Construct the left subtree from the elements lying on the left side of the max element
   4. Construct the right subtree from the elements lying on the right side of the max element

   Yes, given a sequence the Cartesian tree is unique. This is because max must be in the root anyway. Then, for the inorder sequence to produce all elements to the left of the max in the sequence before x, all those elements must be in the lft subtree of x (or inorder traversal will not print them befor x). Same for right subtree. This argument will now hold recursively for subtrees, and they trivially hold for 1-element sequences (leaf node, base case). Can prove more formally using this idea using induction.

7. Given an array A of n integers such that there is at most k distinct values in A, design an O(nlog k) algorithm to sort the array. You can use at most O(k) space.

   We create a balanced BST. At each node, we also maintain a *count*, which is the number of elements in the array with key equal to this node. We insert all A[i] in the BST. If A[i] was not present in the BST before, we create a new node and initialize its *count* to be 1. If it's already present, we just increment *count* by 1. So there are at most k nodes in the BST, so O(k) space. After all the elements have been inserted, we can do an inorder traversal and get the sorted sequence.

8. Given an array A of n integers an a positive integer k, design an algorithm to construct another array B such that $B[i] = \max\{A[j] \mid \max(i-k+1), 1) \leq j \leq i\}$ (i.e., B[i] is the sum of the last k elements of the array ending at i). Your algorithm should run in O(nlog k) time.

```
Create a balanced BST
For i = 1 to n
    If i < k
        insert A[i] in the BST
        B[i] = max element in the BST
    Else
        delete A[i-k] from the BST
        insert A[i] in the BST
        B[i] = max element in the BST
```

Note that insert, findMax, and delete can all be done on O(lg n) time.

9. Given an array A of n integers, design an algorithm to find the total number of subarrays A[i..j] (part of A from index i to index j) such that $\sum A[k]$, $i \leq k \leq j$ is 0. Your algorithm should run in expected O(n) time.

Create an array P such that P[i] gives the prefix sum till index i (sum of elements from A[1] to A[i]) (can be done easily recursively: set P[1] = A[1]; P[i] = P[i-1] + A[i])

Note that if P[i] = P[j], j > i, then the subarray sum of A[i..j] is 0.

Create a hash table to store tuples (x, count), where x is the key hashed, count is associated (satellite) data stored with it. X will store P[i]'s, count will store number of times a value P[i] occurs in P

```
Set answer = 0;
For i = 1 to n
        Search for P[i] in the hash table.
        If not found insert (P[i], 1) in hash table
        Else
                Let the value found be (P[i], c)
                answer += c
                Update the value to (P[i], c+1);
Finally, the value of answer gives the required number of subarrays
```